AN ALGORITHM FOR THE SOLUTION OF NON-LINEAR OPTIMIZATION PROBLEMS AND ITS APPLICATION IN RADIATION PROTECTION

A thesis submitted for the award of the degree of Doctor of Philosophy of the University of London

by

A.D. Rowe

B Sc DURHAM UNIVERSITY

M Sc LONDON UNIVERSITY (QUEEN MARY COLLEGE)

Nuclear Power Section

Mechanical Engineering Department

Imperial College of Science and Technology

November 1974

ABSTRACT

A review of some of the more effective methods for the solution of non-linear optimization problems is given. The mode of development of a new algorithm for the solution of constrained non-linear optimization problems is described and criteria by which to assess the relative merits of different algorithms are discussed.

The algorithm presented is based on the well-known method of successive linear approximations. At each stage the object function and constraints are linearized and a standard implementation of the revised simplex technique is used to solve the resulting linear programs. A new method - the displaced origin technique - is used to set up the linearized problem for solution. This permits substantial reductions in both computer core storage and execution time requirements. An efficient convergence acceleration technique which employs a cubic curve fitting procedure has been developed. The computer code SLA, which implements the algorithm, has been tested against a wide variety of standard test problems. Comparisons are made with published data wherever possible. The code is shown to work well with numerical approximations to the first derivatives of the object function and constraints. This reduces the amount of work necessary to prepare problems for solution by numerical techniques. An easy-input self-documenting user version of the code SLA is described.

SLA has been applied to the problem of determining minimum weight radiation shielding. Constraints have been applied to limit the maximum integrated dose which any one of a set of radiation workers may absorb whilst following a particular work schedule. Results obtained for a 25 element shield are presented and their physical significance discussed.

ACKNOWLEDGEMENTS

The author would like to thank both the Science Research Council and the Safety and Reliability Directorate (UKAEA, Culcheth) who provided financial support for the work.

Thanks are due to Dr. A.J.H. Goddard who supervised the project.

The author would like to express his appreciation to Dr. J.R.A. Lakey of the Royal Naval College, Greenwich who provided many stimulating discussions on the problems involved in radiation shield optimization.

The author is also indebted to Mr. D.S.H. Rosenthal for his greatly valued expert advice on computing matters.

Finally the author wishes to thank his wife Shahla, for painstakingly reading the manuscript and providing much needed critical comment, and for her patience.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Over the last 20 years the use of mathematical programming techniques to determine optimal solutions to complex problems has increased enormously. This vigorous growth has only been made possible by the parallel development of high-speed electronic computers, since only by using such machines can practical problems be solved with any reasonable expenditure of human effort. As the power and size of computers has increased so has the range of processes which may be effectively modelled and optimized. Mathematical programming techniques are now being applied to problems in many fields of activity such as financial planning, economic modelling, industrial process optimization, scheduling, stock control and many more besides.

The term mathematical programming is the generic name for a set of techniques which are used for solving a wide variety of problems. Integer programming, linear programming, network flow analysis, dynamic programming and non-linear programming are all examples of mathematical programming techniques. The work described here belongs to the branch of non-linear programming. The problems within this field are conveniently divided into two types. There are non-linear programming problems which involve the minimization of non-linear functions on which no restrictions are placed. These are known as unconstrained problems, and the independent variables defining the object function, may assume any real value, positive or negative. The other type of non-linear programming problem is referred to as constrained, since the independent variables are not free to assume any value. Conditions, in the form of constraint equations, are imposed on the values which the independent variables may assume. These conditions define an area (the feasible

region) where acceptable solutions to the problem may be located. Such solutions are known as feasible solutions.

There are a number of very efficient techniques available for minimizing unconstrained functions and most problems of this type may be tackled with every confidence of success. However the situation as regards constrained problems is not so reassuring. A variety of methods exist which are capable of solving non-linearly constrained problems, but no one algorithm has demonstrated a clear superiority. The development of techniques for solving problems of this type remains an active field of research.

The algorithm presented here solves non-linear programming problems by replacing them with a series of linear programming problems which are easier to solve. At each stage both the object function (*i.e.* the function to be minimized) and the constraint functions are replaced by appropriate first order Taylor-Series approximations. This process is referred to as linearization and is achieved using either explicit formulae for the required first derivatives (*i.e.* analytically) or by use of numerical approximations to these derivatives (*i.e.* numerically). The linear programming problems produced are solved by a standard algorithm. The solution vector found for each linear program is used as the linearization point (*i.e.* the point at which derivatives are evaluated) for each ensuing linear program.

There are several important advantages which accrue from this approach.

(1) With very few exceptions indeed all linear programming problems may be solved in a finite number of steps by existing techniques. Thus there is every reason to believe that an algorithm based on successive linear approximations will be robust.

(2) The simplex method for the solution of linear programs was devised more than 20 years ago. As a result, the algorithms available to-day are the end products of a long period of intensive development and are thus very efficient and numerically stable. Problems involving several hundred variables and constraints may be solved as a matter of routine. Therefore it is unlikely that an algorithm developed by testing on small problems will break down on larger ones.

(3) The method is conceptually simple. This makes the development of an effective algorithm simpler. It also makes it less likely that an implementation of the algorithm will be incorrectly applied by an inexperienced user.

(4) The linear programming technique treats constraints directly. Many methods for solving non-linear constrained problems are in fact unconstrained algorithms on to which a strategy for dealing with constraints has been subsequently added. This is not always a successful union.

(5) The technique of successive linear approximation has been used successfully before. This demonstrates that this approach to non-linear programming is fundamentally sound.

The algorithm described here is shown to be a substantial improvement on other algorithms using linear approximations and that, in terms of computing time, it is as efficient as the best non-linear algorithms currently available. This performance has been achieved by:

(1) Reformulating the linear program so as to avoid the necessity of doubling the dimensionality of the problem. (The independent variables of linear programs must be non-negative. This has meant that the independent variables of the non-linear problem have had to be expressed as the difference between two non-

negative variables. In this way negative values could be treated by the linear programming algorithm. By avoiding this 'variable splitting' approach, substantial savings in computer core storage requirements and calculation time have been achieved.)

(2) Developing a sophisticated step adjustment strategy which utilizes cubic fitting procedures to accelerate convergence.

A brief review of non-linear programming techniques is given in the following chapter. This is followed by an account of the mode of development of the algorithm and a discussion of criteria by which to assess competing algorithms. A description of the theoretical basis of the algorithm and details of the way in which it has been realized are provided. This is followed by the results obtained with the computer code developed for a series of standard test problems. Where possible comparisons are made with published results. The algorithm is then used to evaluate the minimum weight configuration of a radiation shield, subject to constraints on the maximum dose rate allowed at a set of dose points. Constraints are also applied to limit the maximum radiation dose absorbed by a set of workers whilst observing a prespecified work schedule. The physical implications of the results obtained for this shield system and work schedule are discussed.

CHAPTER 2

A SURVEY OF NON-LINEAR OPTIMIZATION TECHNIQUES

In this chapter a review of methods for solving non-linear optimization problems is given. Unconstrained techniques are discussed first since in many cases they form the basis of the constrained techniques which are considered subsequently. It is not possible to consider here all of the methods available for non-linear optimization so this survey should be regarded as indicative rather than comprehensive.

2-1 Unconstrained Techniques

The unconstrained problem may be stated as:

$$\text{Minimize: } f(\mathbf{x}) , \quad \mathbf{x} \in E^n \tag{2-1}$$

In general it will be assumed that the first derivatives of $f(\mathbf{x})$ are finite and continuous but this is not a necessary pre-requisite for the search methods. A necessary, but not sufficient, condition for $f(\mathbf{x})$ to have a minimum is that the elements of the gradient of $f(\mathbf{x})$ shall all be zero.

$$\frac{\partial f}{\partial x_i} = 0 \quad (i = 1,n) \tag{2-2}$$

For such a point to be a minimum the matrix of second partial derivatives (the Hessian) must be positive definite, *i.e.*

$$\mathbf{p}^T H \mathbf{p} > 0, \quad \text{for all} \quad \mathbf{p} \neq 0 \tag{2-3}$$

- where the elements of the matrix H are:

$$h_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

It is convenient to divide the methods for solving equation 2-1 into two groups. The first group of methods makes use of gradient information (Gradient Methods) while the second group requires only function values

(Search Methods).

2-1-1 Gradient Methods

(1) Steepest descent

This method uses the negative gradient, $-g$, of $f(x)$ as the direction of search. The negative gradient gives the direction of greatest local function decrease. Thus, if x_K is the starting point, the next point in the search, x_{K+1}, is given by:

$$x_{K+1} = x_K + \lambda_K s_K \qquad (2\text{-}4)$$

- where s_K is a unit vector in the direction $-g$

λ_K is a scalar.

The method of steepest descent supplies a search direction, s_K, but not the magnitude of the step to be taken, λ_K. In general the approach usually adopted is to perform a linear search for a minimum in the direction s_K. Thus the value of λ_K is determined such that $f(x_K + \lambda_K . s_K)$ is a minimum. A new point is generated using equation 2-4 and the gradient is re-evaluated. A new linear search ensues. The great attraction of the method is its extreme simplicity. However the concept of steepest descent is largely illusory since by rescaling the independent variables a new gradient vector and hence a new steepest descent direction is found. A more serious drawback lies in the performance of the method on functions having valley structures. In such cases the method characteristically pursues an inefficient zig-zag path. Since several more efficient techniques are available the method of steepest descent is not widely used.

(2) Conjugate gradient

Most functions are well approximated by a quadric close to a minimum and so many minimization methods are based on techniques which are guaranteed to locate the minimum of a quadric in a finite number of steps.

The method of conjugate gradients is one such method. The method was originally developed by Hestenes and Steifel [Ref. 1]. Fletcher and Reeves [Ref. 2] developed the ideas and prepared a computer program to implement the algorithm. The method involves a series of linear searches in a direction defined by a linear combination of the current gradient and the previous search direction. The search direction on the k^{th} stage is given by:

$$s_K = -g_K + \beta_{K-1} s_{K-1} \tag{2-5}$$

- where $\beta_{K-1} = (g_K^T . g_K) / (g_{K-1}^T . g_{K-1})$

$\beta_0 = 0$

This procedure for selecting the search directions ensures that they are mutually conjugate. Two directions p and q are said to be mutually conjugate with respect to a quadric if, and only if:

$$p^T H q = 0 \tag{2-6}$$

- where H is the Hessian of the quadric.

Conjugate directions are chosen since it may be shown that for a quadric with a unique minimum, this minimum is found in at most n steps by minimizing in turn along n conjugate directions, starting from any point. Thus by performing the linear searches in n mutually conjugate directions the required guaranteed convergence for a quadric is achieved. Fletcher points out that in order to preserve the quadratic convergence properties of the method on more general functions it is necessary to restart the direction generation procedure every $n+1$ iterations with the steepest descent direction. A disadvantage of this method is the requirement for accurate location of the line search minima, without which the quadratic convergence is lost. The method is however economical with computer core storage requiring space for only three n-

vectors.

(3) Davidon-Fletcher-Powell (DFP)

The DFP technique is a second order method in that it uses information about the second partial derivatives of the function. However the second order derivatives are not evaluated directly but are approximated using first order information only. If the function $f(\mathbf{x})$ is expanded in a Taylor Series about the point $\mathbf{x}_K$ then:

$$f(\mathbf{x}_{K+1}) = f(\mathbf{x}_K) + \mathbf{p}_K^T \mathbf{g}_K + \tfrac{1}{2}\mathbf{p}_K^T \mathbf{H}_K \mathbf{p}_K \qquad (2\text{-}7)$$

- where $\mathbf{x}_{K+1} = \mathbf{x}_K + \mathbf{p}_K$

$\mathbf{g}_K$ is the gradient of $f(\mathbf{x})$, evaluated at $\mathbf{x}_K$

$\mathbf{H}_K$ is the Hessian, also evaluated at $\mathbf{x}_K$

It was noted earlier that a necessary condition for a stationary point of $f(\mathbf{x})$ was that the gradient should vanish. Thus by taking the gradient of $f(\mathbf{x}_{K+1})$ and setting it to zero the value of $\mathbf{p}_K$, the perturbation required to move from $\mathbf{x}_K$ to $\mathbf{x}_{K+1}$ may be found. If $f(\mathbf{x})$ is a quadric then $\mathbf{H}$ is constant and the minimum may be reached in one move. If $f(\mathbf{x})$ is a more general function then $\mathbf{p}_K$ will provide a direction for a line search. Since $\mathbf{p}_K$ is based on a second order information it may be expected to be a superior search direction to the negative gradient. Formally, the gradient of $f(\mathbf{x}_{K+1})$ is:

$$\mathbf{g}_{K+1} = \mathbf{g}_K + \mathbf{H}_K \cdot \mathbf{p}_K \qquad (2\text{-}8)$$

Setting $\mathbf{g}_{K+1}$ to zero:

$$\mathbf{p}_K = -\mathbf{H}_K^{-1} \mathbf{g}_K \qquad (2\text{-}9)$$

The DFP technique builds up an approximation to the inverse of the Hessian (*i.e.* $\mathbf{H}_K^{-1}$ in equation 2-9) using gradient information only. Initially the direction of search is chosen to be that of steepest descent

and a unit matrix is used for the initial inverse Hessian approximation. However as more gradient information is generated the inverse Hessian approximation is steadily updated using a formula developed by Davidon [Ref. 3]. The method was implemented by Fletcher and Powell in 1963 and is probably the most powerful unconstrained optimization technique using first derivative information. However the method does not perform reliably when numerical approximations to the first derivatives are used.

2-1-2 Search Methods

(1) Simplex

The simplex search method, which is not to be confused with the simplex method for the solution of linear programming problems, was originally devised by Spendley, Hext and Himsworth [Ref. 4] and was later improved by Nelder and Meade [Ref. 5]. The method begins with the construction of a simplex in the n-space. This is a set of N+1 points distributed, initially at least, so as to be mutually equidistant. In two dimensions the simplex is a triangle and in three dimensions a tetrahedron. The search proceeds by repeatedly reflecting the vertex having the highest function value about the centroid of the simplex. As proposed by Spendley *et al.* the simplex remained regular and was uniformly reduced in size if one of the vertices remained unchanged for a pre-specified number of iterations. The method was modified by Nelder and Meade to allow the simplex to assume an irregular shape by allowing expansion and contraction as well as reflection about the centroid. The method is quite efficient in finding the general location of the minimum but is generally slow to converge. The performance of the method is also significantly affected by the choice of initial simplex.

(2) Rosenbrock's method

This method starts with a set of orthonormal directions, usually the co-ordinate axes. Searches are then made along each direction in turn. The method of search adopted is a simple probe technique. A step, λ_i, is taken in the i^{th} direction. If a lower function value is found λ_i is multiplied by 3 and the new value for the independent variable, x_i, is retained. If a higher function value is found λ_i is multiplied by -0.5 and the old value of x_i is retained. When each direction has been searched in this way a further probe is made in the direction defined by the vector whose elements are the individual perturbations in each search direction; *i.e.* the resultant vector. After this final probe the direction between the initial point and the final point is used as one of a new set of orthonormal directions. This new set of directions is generated by the Gram-Schmidt procedure. The original procedure is due to Rosenbrock [Ref. 6] but was modified by Davies, Swann and Campey [Ref. 7] who replaced the probe procedures by linear searches. This refined method is known by the acronym DSC. Neither the original Rosenbrock method nor its derivative DSC, is regarded as efficient in terms of the number of function evaluations required to obtain a solution.

(2) Powell's method

The method due to Powell [Ref. 8] is probably the most powerful method currently available for unconstrained minimization of functions for which analytical first derivatives are unavailable. Like the method of conjugate gradients it is quadratically convergent in that it minimizes a quadric in a finite number of steps. The basis of the method is to set up a series of mutually conjugate directions, minimizing along them as they are set up. A quadric is minimized by n such conjugate line minimizations. However it takes $n+1$ single variable minimizations to

set up each conjugate direction so in general to find the minimum of a quadric will require $n(n+1)$ line minimizations. The method begins with an arbitrary set of n linearly independent search directions $s_1, s_2, \ldots, s_n$. Linear searches are performed along each direction in turn. This is followed by a linear search in the resultant direction:

$$x_1 = x_0 + \sum_{i=1}^{n} \lambda_i s_i \qquad (2\text{-}10)$$

$$x_2 = x_1 + \lambda_{n+1} (x_1 - x_0) \qquad (2\text{-}11)$$

- where x_0 is the starting point

  x_1 is the point obtained after n linear searches

  x_2 is the point obtained after a linear search in the resultant direction.

Tests are than made to see whether the resultant vector $(x_1 - x_0)$ can replace one of the initial search directions, s_i. These tests are necessary to ensure the preservation of the linear independence of the search directions thus avoiding the possibility of finding the minimum in a sub-space only. Badly scaled problems can cause the resultant direction to the rejected frequently and this reduces the efficiency of the algorithm.

2-2 *Constrained Techniques*

The methods used for solving the general constrained non-linear optimization problem may be divided into three groups. The first two are, as for the unconstrained cases, gradient methods and search methods. The third group covers methods based on linear programming techniques. The main features of some of the more successful methods in the first two groups are described below. Methods based on linear programming are discussed in Chapter 4.

2-2-1 Gradient Methods

(1) Projection methods

The common feature of projection techniques is that a search direction, chosen by some rule, is projected on to a set of 'active' linear or linearized constraints. This has the effect of reducing components of the search direction orthogonal to the constraints to zero, whilst those parallel are unaffected. A linear search can thus be made in the projected search direction which will never leave the feasible region, provided the constraints are linear. If the constraints are non-linear then a correction procedure must follow the linear search to enable feasibility to be regained. An early example of a projection technique is due to Rosen [Ref. 9] who used the negative gradient as the basic search direction. New points are determined as:

$$x_{K+1} = x_K - \lambda_K P g_K \qquad (2\text{-}12)$$

- where P is a projection matrix

λ_K is chosen so as to minimize $f(x_{K+1})$

The method incorporates strategies for adding or dropping constraints from the set of active constraints. Goldfarb and Lapidus [Ref. 10] have devised a method whereby the basic search direction is chosen as in the DFP algorithm. The updated approximation to the inverse Hessian and the projection matrix are combined.

Projection methods are very efficient in terms of the number of function evaluations required. However, for larger problems the computing time required to calculate and manipulate projection matrices can be substantial, and unless the evaluation of the object function is lengthy this can cancel out time savings due to the reduced number of function calls. A further difficulty with projection techniques is their sensitivity to small errors in derivatives calculated numerically

making the provision of analytical derivatives virtually essential. There are a number of methods for treating non-linearly constrained problems by projection techniques currently under active development.

(2) Reduced gradient technique

There are many points of similarity between projection and reduced gradient techniques. In fact some authors refer to the reduced gradient method as a projection method. The method has been developed by Abadie and Carpentier [Ref. 11] from an algorithm due to Wolfe. Inequality constraints are treated by converting them to equality constraints using non-negative slack variables. Thus the constraint:

$$\Phi_i(\mathbf{x}) \geq 0$$

- becomes

$$\Phi_i(\mathbf{x}) - x_s^2 = 0 \qquad (2\text{-}13)$$

- where x_s is the additional slack variable.

($x_s = 0$ implies Φ_i is an active constraint)

The constraints are linearized and the dimensionality of the problem reduced by eliminating p of the independent variables using the p constraint equations. An unconstrained search is made in the reduced space using a conjugate gradient algorithm. A correction procedure follows the unconstrained step to restore feasibility. The computer code implementing the generalized reduced gradient algorithm is known by the acronym GRG and is generally regarded as one of the most effective methods available for non-linear programming. The main drawback to its use lies in the necessity to provide explicit formulae for analytical first derivatives of both object function and constraints.

(3) Penalty function methods

These methods rely on converting a constrained problem to a series of unconstrained problems by the addition of a penalty function to the

object function. The penalty function is constructed from the constraints on the problem and a wide variety of formulations have been used. A typical form of penalty function is:

$$\bar{f}(x,r) = f(x) + r \sum_{i=1}^{m} \frac{1}{\Phi_i(x)} + \frac{1}{r} \sum_{i=m+1}^{p} [\Psi_i(x)]^2 \qquad (2\text{-}14)$$

- where $f(x)$ is the original constrained object function.

$\Phi_i(x)$ are the m inequality constraints.

$\Psi_i(x)$ are the $(p-m)$ equality constraints.

r is an adjustable parameter.

It is apparent from the form of the penalty function that, as the inequality constraints are approached and the $\Phi_i(x)$ tend to zero, the value of the first penalty term increases sharply. This steep barrier prevents an unconstrained minimization procedure violating the constraint. The second penalty term in equation 2-14 takes a minimum value of zero when the equality constraints $\Psi_i(x)$ are precisely satisfied. This ensures that $\bar{f}(x,r)$ decreases as the equality constraints are more nearly satisfied. The method begins with an initial value for r, selected by experience or empirical rule. The modified unconstrained function $\bar{f}(x,r)$ is then minimized. Any efficient unconstrained minimization technique may be used for this stage. The value of r is then decreased by typically a factor of ten, and a second unconstrained minimum obtained. The whole process is repeated, each time reducing r, and with each new cycle beginning from the minimum of the previous one. The steady reduction of r makes the penalty function barriers progressively steeper allowing the unconstrained minima to approach the active constraints to within an arbitrary tolerance. Acceleration procedures based on the trajectory of the successive minima are sometimes used. Fiacco and McCormick [Ref. 12] have developed the method inten-

sively and their SUMT (Sequential Unconstrained Minimization Technique) program is widely used. The use of penalty functions has the disadvantage of converting the original problem to a series of predictably difficult unconstrained minimizations. The unconstrained minimizations are difficult because the penalty function produces steep-sided asymmetric valleys. These are not generally well approximated by quadrics and thus cause difficulties for algorithms based on such approximations.

2-2-2 Search Methods

(1) The complex method

This method is due to Box [Ref. 13] and is based on the unconstrained simplex method described earlier. In the complex method the simplex with n+1 vertices is replaced by a 'complex' which has k (>n+1) vertices. Box suggests a value of 2n for k but points out this is too large when n is greater than ten. It is necessary to use more than n+1 points in order to prevent the configuration from collapsing prematurely into a subspace. Only one initial feasible point is required. The remaining k-1 feasible points required to set up the complex are determined by a random number procedure. Any infeasible point generated by this procedure is rejected and a new trial point selected. Once the complex is set up the search proceeds as in the simplex method. The vertex having the highest function value is reflected about the centroid of the complex. Box provides a set of empirical rules for the manipulation of the complex. The method has been used successfully but its efficiency falls off rapidly as the number of independent variables increases.

(2) Flexible tolerance

This method was devised by Paviani and Himmelblau [Ref. 14]. It has some points in common with the penalty function technique in that the original problem is replaced by a related problem which, as a limiting case,

has the same solution as the primary problem. However in the Flexible Tolerance method the reformulation affects the representation of the constraints only. The object function remains unchanged. The basis of the method lies in combining all the constraints of the original problem (inequality and equality) into one gross constraint. The object function is then minimized as an unconstrained problem, using the simplex method of Nelder and Meade, until the gross constraint is violated. A separate unconstrained minimization is performed on a function defined by the constraints of the primary problem in order to regain feasibility with respect to the single gross constraint of the reformulated problem. In this way the processes of function reduction and constraint satisfaction are effectively separated. If the primary problem is:

$$\left.\begin{array}{lll} \text{Minimize :} & f(\mathbf{x}) \quad , & \mathbf{x} \in E^n \\ \text{Subject to :} & \Phi_i(\mathbf{x}) \geq 0 & (i = 1,m) \\ & \Psi_i(\mathbf{x}) = 0 & (i = m+1,p) \end{array}\right\} \qquad (2\text{-}15)$$

The reformulated problem is then:

$$\left.\begin{array}{lll} \text{Minimise :} & f(\mathbf{x}) \quad , & \mathbf{x} \in E^n \\ \text{Subject to :} & L^K - T(\mathbf{x}) \geq 0 & \end{array}\right\} \qquad (2\text{-}16)$$

The function $T(\mathbf{x})$ is an amalgamation of all the inequality and equality constraints of the primary problem. It is defined as:

$$T(\mathbf{x}) = \left[\sum_{i=1}^{m} \delta_i \Phi_i^2(\mathbf{x}) + \sum_{i=m+1}^{p} \Psi_i^2(\mathbf{x})\right]^{\frac{1}{2}} \qquad (2\text{-}17)$$

- where $\delta_i = 0$ if $\Phi_i \geq 0$

$\delta_i = 1$ if $\Phi_i < 0$

Note that $T(\mathbf{x}) = 0$ when $\mathbf{x}$ is a feasible point with respect to the primary problem and also that $T(\mathbf{x})$ cannot be negative. The function L^K

is the 'tolerance criterion' for the k^{th} stage of the search. It depends on the average distance of the vertices of the simplex from the centroid of that simplex. It is so defined as to be positive and monotonically decreasing at each successive stage. Each new point generated by the search procedure is classified as one of three types:

(1) Feasible, if $T(x) = 0$

(2) Near Feasible, if $0 \leq T(x) \leq L^K$

(3) Non-Feasible, if $T(x) > L^K$.

Since the minimum of $T(x)$ is zero, as L^K decreases, the tolerance on near feasible points is correspondingly reduced. In the limit as L^K tends to zero points must be generated such that $T(x)$ tends to zero also. Such points are feasible with respect to the primary problem.

The Flexible Tolerance algorithm is quite reliable and, being based on the simplex search technique, requires no derivative information. However convergence is frequently very slow.

CHAPTER 3

DESIGN, DEVELOPMENT AND ASSESSMENT OF AN OPTIMIZATION ALGORITHM

3-1 Algorithm Design

Algorithms used for multivariate optimization are invariably implemented on high speed electronic computers. In terms of both efficiency and reliability the way in which an algorithm is programmed for the computer can be as important as the initial choice of algorithm. Seemingly small changes in the way an algorithm is programmed can produce very significant effects on the rate of convergence. Colville, [Ref. 15] who conducted a survey into the efficiency of 34 different computer codes, states that "... it appears that the efficiency and performance of a non-linear programming code can be greatly affected by the method of implementing it on a computer."

Bearing this in mind the algorithm developed here - subsequently referred to as SLA - has from the outset been designed for maximum numerical reliability and computational efficiency. Where these two considerations have been in conflict, reliability has been considered of more importance than sheer computational speed. Numerical reliability has been achieved by careful preparation of logic flow diagrams for all subroutines and by detailed consideration of possible failure cases. Particular attention has been paid to the problems associated with division by very small numbers. This approach has been supported by a great deal of testing against standard test problems. This was largely done using the test program N1OC, described in the following section. The coding of SLA is in FORTRAN which although not as efficient as some programming languages is widely used and understood.

The principal aims considered in the design of SLA are summarized below.

(1) The algorithm should be reliable, *i.e.* it should solve real world problems most of the time.

(2) It should be a computationally efficient algorithm; *i.e.* it should not make excessive demands either on central processor time or central memory storage.

(3) The algorithm should be conceptually simple so allowing future modification and easy maintenance.

(4) The amount of input data and option selection should be kept to a minimum to ensure the code is easy to use.

(5) To produce a code which will function well using numerical estimates for the partial derivatives of both object function and constraints.

(6) To produce a modular code; *i.e.* one built up of well-defined independent sections, so that program modifications do not entail wholesale rewriting of the code.

(7) To produce a fully documented 'user package' to enable non-specialists to use SLA.

The extent to which SLA fulfils these specifications will, it is hoped, become apparent in the following chapters. In any event these aims will be reexamined in Chapter 8.

3-2 Method of Program Development

The code was developed by linking it to a special purpose test program, N1OC. This test program runs interactively via a time-sharing teletype on the Imperial College CDC 6400 computer. The code N1OC allows any one of a series of thirteen standard test problems to be selected by typing in the appropriate problem number. For each problem there is a pre-programmed set of standard data which may be inspected and changed at will. Data items which may be changed include the initial starting vector, convergence criterion and initial step lengths. The user may choose to use analytical derivatives or numerical approximations, in which

case the user must supply a pertubation value to use in the forward difference formula. Several different output options are available. A block of iterations, say, from iteration I1 to iteration I2 may be specified for output and at the same time every N^{th} iteration may be printed. The user controls whether the step length for each variable is printed at each iteration and there is also the option of a comprehensive output which gives all the data supplied to the linear programming routine as well as details of the iteration progress within the linear program itself.

The test problems used by N1OC are rather small and none has more than four variables. The chief reason for this is the maximum central memory allocation of 25000 words. This storage allocation is not dedicated to the one teletype but is shared between anything up to 40 other teletypes. This means that at times of peak load, with many teletypes connected to the computer, response times can be unacceptably long, unless demands on the central processor are kept to a minimum. This necessitates the use of small test problems. Care was taken in the construction of N1OC to ensure that the optimization routines comprising SLA remained distinct and immediately separable. Figure 3-1 shows the logic flow within N1OC. The test problems are described fully in Appendix 1. These problems are quite varied encompassing unconstrained, linearly constrained and non-linearly constrained problems.

Once the code N1OC had been developed it became a relatively straightforward matter to test new strategies and variations in program arrangement against the test problems. By plotting out the progress of the search for two-variable problems insight was gained into the behaviour of the linear approximation method. Causes of program failure were identified and eliminated. It was found essential to test modifications against as many test problems as possible since too much concentration

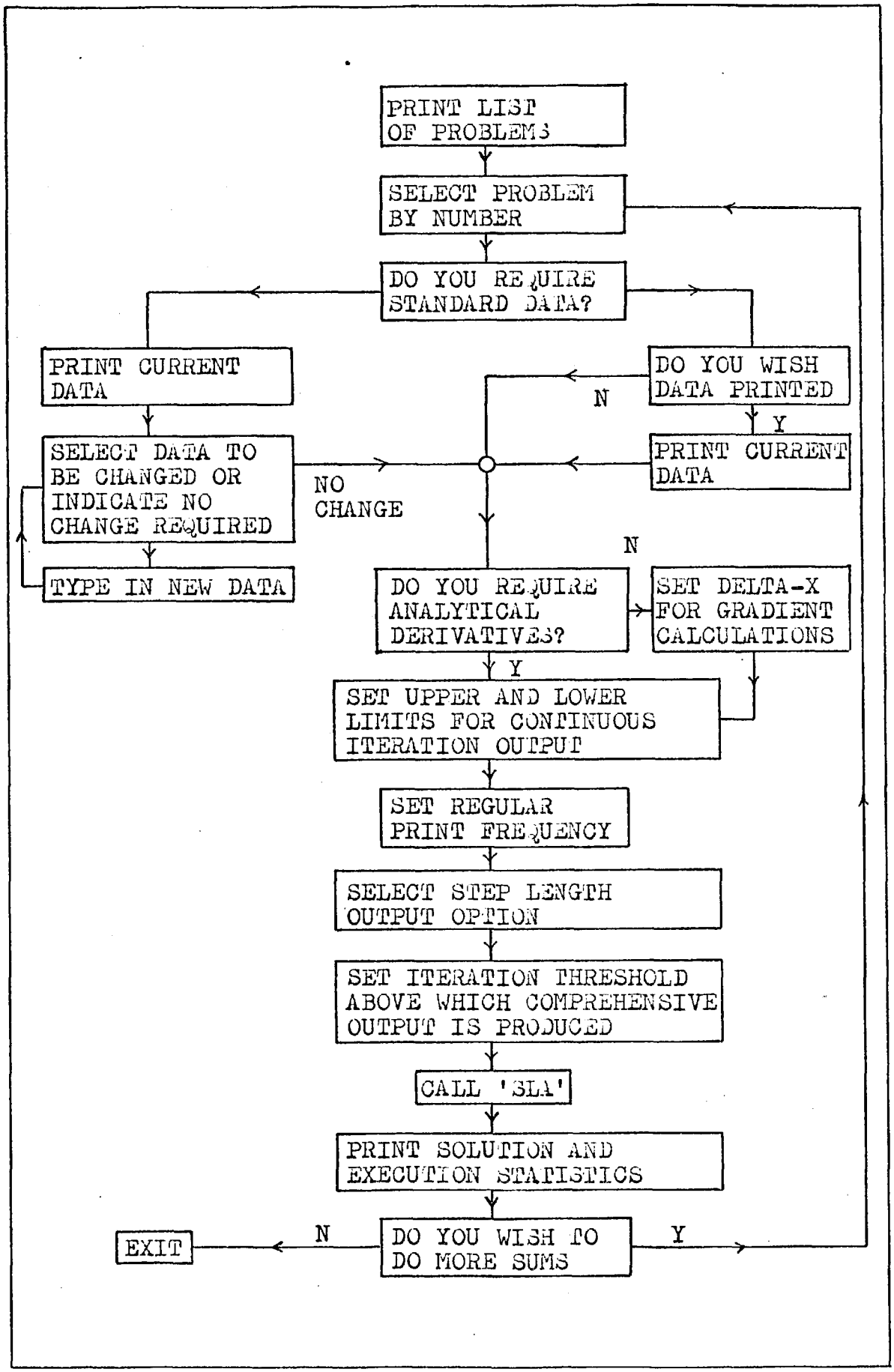

FIG. 3-1. The Test Program N10C

on one particular test case held the danger of producing a problem specific algorithm. Modifications which decreased the reliability of the algorithm were invariably rejected regardless of gains in computational efficiency. Acceptable modifications were required to have a positive effect on computational efficiency, for example, a reduction in core storage or execution time, and at least a neutral effect on reliability.

3-3 Algorithm Assessment

An optimization algorithm is required to solve a great variety of problems in a wide range of disciplines. As a consequence of this the demands made upon the algorithm are many and varied. Therefore there can be no single criterion of effectiveness. Since no one algorithm has proved to be superior to all others in all respects, it is necessary to assess the merits of alternative algorithms by classifying the various attributes a good method should possess. An attempt may then be made to quantify each of these attributes. However it is for the problem solver or 'user' himself to decide on the relative importance to be attached to these individual factors. Each user works under different external constraints of time, finance and computer resources. Consequently each will attach different importance to these factors.

A list of four criteria against which an algorithm may be assessed is given below. This does not purport to be an exhaustive list, but does include items of major importance for most problem solving situations.

Criteria for Algorithm Assessment

(1) Computer resource requirements.

(2) Accuracy of solution.

(3) Ease of use.

(4) Algorithm reliability.

The importance and measurement of each of these criteria is discussed

below.

3-3-1 Computer Resource Requirements

There are several ways in which demands on the computer system may be monitored. The most obvious is perhaps the central processor time required for solution. If non-linear programming algorithms were implemented in the same programming language with equal regard to computational efficiency, compiled by the same compiler and finally executed on the same type of computer running under the same operation system then, and only then, would accurate comparison of algorithms on the basis of central processor time be possible. Of course this idealized situation is not attainable in practice and a compromise must be sought. Colville [Ref. 16] used a standard timing program in an attempt to provide normalization factors for various computers. All solution times were divided by the central processor time required by the timing program. However the timing program, which performs a series of matrix inversions, can fail to reflect adequately the relative speeds of different computing machines. Factors of two or three in the standardized times for the same code solving the same problem on different machines are not to be unexpected. However, despite this serious drawback to intercomparison, standardized central processor times are still widely used for algorithm assessment.

A second measure of computer resource demands is that of central memory storage requirements. Again there are difficulties in making comparisons. The most important of these is the different word length found on different machines; *i.e.* the number of bits (binary digits) which represent each word in the machine. For example, the core requirements for SLA quoted in Section 5-1-2 refer to the 60 bit words found on the CDC 6000 series machines. To obtain equivalent accuracy on another machine might require all variables to be represented as double

precision variables; *i.e.* each variable is represented by a combination of two ordinary (single precision) words. Apart from considerations of computer word length it is well to bear in mind that different compilers will produce machine code in varying degrees of compactness and that different operating systems will supply system routines (*e.g.* input and output routines) which are of varying lengths. Despite these drawbacks, provided the precision requirements are specified (*i.e.* the word length) meaningful comparisons can be made. Unfortunately, core storage requirements are frequently not quoted in published work, even though storage limitations, unlike central processor time limits are virtually impossible to circumvent without a major restructuring of the computer code.

A third and widely used method of measuring computational efficiency is that of recording the number of times the object function is evaluated during the problem solution. There immediately arises a problem when comparing two computer codes one of which uses analytical derivatives and the other numerical derivatives. Other factors being equal, the code using analytical derivatives will require fewer function evaluations for solution than the other code, since to evaluate the n partial derivatives of an n-variable function requires at least n+1 function calls. Some authors do however express their results in terms of 'effective function evaluations' allowing an extra n+1 function evaluations for each analytical gradient calculation. The results quoted elsewhere in this work are all in terms of effective function evaluations (e.f.e.). A much more serious drawback to the use of function evaluations for computer resource monitoring is that for many large constrained problems the computing effort required to evaluate the object function may be small compared to the effort required to compute the next point. Under these circumstances it is clearly good policy to ensure that the expensive

(in computing resources) step once taken is the best possible. Excessive concern to minimize the number of function evaluations by the use of elaborate search procedures can incur an overall penalty in terms of central processor time. Similar comments also apply to assessments of efficiency based on the number of constraint evaluations for solution. Colville in his extensive optimization code survey says that the number of function and constraint evaluations were 'not at all useful in analysing the results obtained' [Ref. 15]. Despite this, function evaluations are almost the universal currency of optimization code comparisons and for this reason some of the results presented here are given in this way.

A fourth possibility for the measurement of computer resource demand is that of financial cost. This economic approach is attractive since a minimum cost solution to a problem is always highly desirable. However there are distinct practical difficulties involved. Charging formulae are installation dependent and are usually designed to penalize excessive use of resources which are scarce at the particular installation. Moreover much code development work is done in academic institutions where little effort is made to assess the commercial value of the computer facilities used. Further difficulties arise when comparing commercial rates between different countries.

In this work results are presented in terms of standardized central processor time and, for the smaller problems, effective function evaluation requirements. Also, estimate of computer core requirements are given in Section 5-1-2.

3-3-2 Accuracy of Solution

The accuracy desired for a particular problem solution may be expressed in a number of different ways. It may be stated in terms of the precision required in the satisfaction of constraints both inequality and

equality, in the precision of the object function or in the accuracy required in each of the elements of the solution vector. When comparing computer codes it is clearly desirable to compare them on an equal accuracy basis, but because of variations in modes of convergence this is rarely possible. There are no absolute standards against which to rank solutions of varying degrees of accuracy since a level of precision which to one user is a vital necessity may be an expensive luxury to another.

The code SLA accepts solution vectors as feasible if the absolute values of all the equality constraints are less than 10^{-6} and if the inequality constraints are violated by not more than the same amount. The accuracy achieved for the elements of the solution vector and object function value are largely determined by the mode of convergence adopted. The three ways in which convergence is recognized for constrained problems by SLA are described in detail in Section 5-6. Of these three modes of convergence two depend upon input tolerances on the solution vector and the other depends on the object function remaining unchanged to within 1 part in 10^6 in ten iterations. Therefore usually the solution vector obtained will be within the user-supplied tolerances but, if the object function is relatively insensitive to changes in the solution vector it may not be.

3-3-3 Ease of Use

It is clear that from a user's point of view the less preparation required to use a computer code the better. It is however, difficult to quantify the effort required to translate a real problem into a form suitable for a numerical optimization procedure. Attempts have been made to quantify preparation times required by different codes for the same problems but the results must be dependent upon the skill and experience of the user. Even though it is difficult to draw quantitative

comparions some useful qualitative conclusions may be drawn. Computer implementations of algorithms which require analytical derivatives to be supplied certainly require more preparation time than those using numerical derivatives. First, the derivatives have to be formed either by hand or by a symbolic manipulation program which will of course require preparation itself. These derivatives must then be translated into some programming language and supplied to the computer, usually through the medium of punched cards. All this work is unnecessary for algorithms not requiring analytical derivatives. It may be that analytical derivatives cannot be formed at all for the object function or constraints. This difficulty arises in problems which involve the solution of a complex numerical model to determine the object function. The format in which data is required by a program is also of importance. If the user has a lot of preliminary programming work and a complex data format to work to, the number of human errors will be large. Even one such error can cost a lot of effort and computer time to track down. This emphasises the comments made earlier about the desirability of codes which work with numerical derivatives, since the less input there is the fewer errors which can be made.

3-3-4 Algorithm Reliability

The reliability of a computer code is most convincingly demonstrated by the successful solution of a wide variety of problems. Proofs of convergence, subject to various restrictions, are available for some unconstrained algorithms. Proofs for constrained minimization algorithms are more difficult to formulate due to the very general nature of the object function and constraints likely to be encountered. SLA is essentially an heuristic technique and as such is not amenable to convergence analysis. However the linear programming routine is known to be extremely reliable and numerically robust, and the routines comprising SLA have been care-

fully designed and tested. This gives good confidence of reliability, but finally the wide variety of test problems successfully solved by SLA serve to indicate the reliability of the method.

3-4 General Requirement for an Optimization Algorithm

Bearing in mind the assessment criteria discussed above, a good algorithm should meet the following three broad objectives.

(1) It must be reliable in that it can be expected to solve to the required accuracy most problems posed most of the time.

(2) It must be easy to prepare for the computer, preferably not requiring explicit analytical derivatives.

(3) It should make reasonable demands on computer resources.

Codes which are unreliable or which require extensive or complex preparation will eventually lose any competitive edge they may have by waste of computer resources. It is for the user to decide what constitutes a reasonable demand on his computer resources. This decision is likely to be based upon the size of the problem to be solved and the value of the solution to him as well as of course the total computer resources he has available.

It should be noted that the first two objectives listed above are as important as the last. Unfortunately, because computer resource requirements can be assigned numerical values they are frequently regarded as the sole and absolute criteria of algorithm assessment. In view of the uncertainties surrounding the relative assessment of computer resource requirements, as outlined in Section 3-3-1, standardized times and function evaluation requirements are more properly regarded as qualitative indications of computer code efficiency.

CHAPTER 4

THEORETICAL BASIS OF THE ALGORITHM

In this chapter the general formulation of the non-linear programming problem is given and the range of problems to which SLA is applicable defined. A description is given of the form of a linear programming problem and the revised simplex method of solution is outlined. Two formulations are given for the application of successive linear approximations to non-linear problems. The original method as described by Griffith and Stewart [Ref. 17] is given first and then the technique developed for SLA is presented. Much of the early work on the SLA search procedure was in fact done using the Griffith and Stewart formulation. Finally a description is given of two techniques, cubic fitting and pattern moves, which are used to accelerate convergence. Cubic fitting is the key element in the step adjustment strategy.

4-1 An Overview of the Algorithm

The code SLA arrives at a solution to the general non-linear programming problem by solving a sequence of linear programs. At each step both object function and constraints are linearized. The linearization is achieved either by evaluating numerically the first partial derivatives of the object function and constraints or by use of explicit user-supplied formulae for the required partial derivatives. The solution point of one linear program is used as the linearization point for the next linear program. Each linearization and ensuing linear program constitutes one iteration. Constraints are imposed on the maximum change allowed in any variable at each iteration. These maximum step length limitations ensure, among other things, that the linear approximations to both the object function and constraints remain valid to within acceptable error bounds. A large part of the logic in SLA is concerned with the adjustment of these

maximum step length limitations.

Thus the essential function of the algorithm is to control and direct a linear programming routine. The results described here were obtained using the subroutine LA01A from the Harwell Subroutine Library. This routine is a good example of the computationally refined algorithms currently available. SLA is coded so that any numerically stable linear programming routine may be used. This means that SLA can easily be upgraded as more efficient linear programming algorithms become available. It also means that a user can employ a subroutine of his own choice if he wishes. However when choosing a different linear programming routine to use with SLA it is more important to use a reliable algorithm than to find the fastest available. Computer core requirements vary between algorithms and this too must be taken into account particularly when attempting large problems.

4-2 The Non-linear Programming Problem

The general non-linear programming problem may be stated formally as:

$$\left.\begin{array}{lll} \text{Minimize :} & f(x) & , \quad x \in E^n \\ \text{Subject to :} & \Phi_i(x) \geq 0 & (i = 1,m) \\ & \Psi_i(x) = 0 & (i = m+1,p) \end{array}\right\} \qquad (4\text{-}1)$$

The formulation given above covers a range of problem types. If some or all of the elements of x are restricted to integer values the problems are properly described as either mixed integer or integer programming problems. If f, Φ_i and Ψ_i are all linear functions the problem is a linear programming problem. If f is a quadric and Φ_i and Ψ_i linear then it is a quadratic programming problem. The subset of problems of concern here and which SLA is designed to solve are those for which f, Φ_i and Ψ_i are continuous non-linear functions. It is

assumed subsequently that these functions also possess finite, continuous first derivatives. The algorithm cannot deal with integer variables but can solve quadratic programming problems and of course linear programming problems.

4-3 Linear Programming

The linear programming problem takes a similar form to that given in equations 4-1 with the obvious restriction that f, Φ_i and Ψ_i are all linear functions. However the linear programming problem is conventionally stated in a different form and this form is given below.

$$\left.\begin{aligned} &\text{Minimize :} && \sum_{i=1}^{n} c_i x_i \\ &\text{Subject to :} && \sum_{i=1}^{n} a_{ij} x_i = b_j \qquad (j = 1,p) \\ & && x_i \geq 0 \qquad (i = 1,n) \end{aligned}\right\} \tag{4-2}$$

Linear programs may be solved by the revised simplex technique [Ref. 18] of which there are numerous computer implementations. The revised technique differs from the original only in the details of the computational procedure. It is more efficient but the underlying concepts remain the same. The technique allows the solution of problems with inequality constraints by the addition of slack variables to the inequality constraint equations. This computational device transforms the inequalities into strict equalities. The set of constraint equations defines a convex polyhedron or a convex region which is unbounded in some direction. Within this convex region all points are feasible with respect to all constraints. It may be shown [Ref. 18, pp 52-54] that a linear function, defined over the convex polyhedron described by the constraint set, takes its minimum value at one or more of the extreme points of the

convex set. If the minimum lies at more than one extreme point then the linear function takes the same value for every convex combination of the minimal extreme points.

The method of solution is iterative and is split into two phases. Phase one determines an initial feasible solution, if one exists, which satisfies all constraints. That is to say, it locates an extreme point of the convex polyhedron defined by the constraints. Phase two involves starting at the initial feasible solution generated by phase one and moving to other extreme point solutions, each time testing for a function decrease. When no further decrease is possible the procedure terminates. Since the convex polyhedron has a finite number of extreme points the procedure will always find a minimum in a finite number of moves. Possible cases of failure are:

(1) No feasible solution exists; *i.e.* the convex polyhedron is void.

(2) The solution is unbounded; *i.e.* at the minimum one or more variables assume infinite values.

(3) Special cases where the simplex procedure breaks down.

The first failure mode is the most important as far as SLA is concerned since a highly non-linear problem, once linearized may have no feasible solution even though the original problem does have a feasible region. There are two ways to proceed should this happen. One is to use a different linearization point. This may produce a new linear problem which does have a feasible solution. The other way is to solve a subsidiary unconstrained problem whose object function is of the form:

$$T(x) = \left[\sum_{i=1}^{m} \delta_i \Phi_i^2(x) + \sum_{i=m+1}^{p} \Psi_i^2(x) \right]^{\frac{1}{2}} \qquad (4\text{-}3)$$

- where $\delta_i = 0$, if $\Phi_i \geq 0$

$\delta_i = 1$, if $\Phi_i < 0$

This is of course the same function which is used by the Flexible Tolerance algorithm to regain feasibility. Other forms are possible. By minimizing $T(x)$ a feasible point will be found if one exists. (Note: It is possible that $T(x)$ will have one or more local minima which may prevent the determination of a feasible solution in this way. These cases should be rare.) The second failure mode cannot occur with SLA since the maximum step length limitation constraints ensure the feasible region is bounded. The third failure mode is very rare indeed and, although examples have been given in the literature [Ref. 18, pp 130-133] they have almost invariably been specially constructed problems designed to cause such failures. The routine LA01A as never suffered such a failure while being used with SLA.

4-4 Successive Linear Approximation

Griffith and Stewart [Ref. 17] were first to describe a method for the solution of non-linear optimization problems involving the solution of a series of linear programs. The method involves the replacement of all non-linear functions by the appropriate first order Taylor Series approximations. In this form equations 4-1 become:

$$\text{Minimize :} \quad f(x_K) + \nabla^T f(x_K).(x - x_K)$$

$$\text{Subject to :} \quad \Phi_i(x_K) + \nabla^T \Phi_i(x_K).(x - x_K) \geq 0, \qquad (i=1,m)$$

$$\Psi_i(x_K) + \nabla^T \Psi_i(x_K).(x - x_K) = 0, \qquad (i=m+1,p)$$

- where x_K is the point at which linearization takes place.

The substitution $\delta x = (x - x_K)$ is made and the equations rearranged as:

$$\text{Minimize :} \quad \nabla^T f(x_K)\ \delta x$$

$$\text{Subject to :} \quad \nabla^T \Phi_i(x_K)\ \delta x \geq -\Phi_i(x_K), \qquad (i=1,m) \tag{4-4}$$

$$\nabla^T \Psi_i(x_K)\ \delta x = -\Psi_i(x_K), \qquad (i=m+1,p)$$

(Note: The term $f(x_K)$ has been dropped from the object function since the solution vector is unchanged by the addition or subtraction of a constant.) Equations 4-4 are now in a form suitable for solution by the revised simplex method of linear programming. Comparison with equations 4-2 show that it is necessary to set:

$$\frac{\partial f}{\partial x_i} = c_i$$

$$\frac{\partial \Phi_i}{\partial x_j} = a_{ij} \qquad (i = 1,m)$$

$$\frac{\partial \Psi_i}{\partial x_j} = a_{ij} \qquad (i = m+1,p)$$

$$\delta x_i = x_i \qquad (i = 1,n)$$

This poses the problem in the conventional form of equations 4-2. Two difficulties arise from this formulation:

(1) It is possible that a well-behaved problem, once linearized, will have an unbounded solution. This is illustrated for a two-variable problem in figure 4-1. The constraints Φ_1 and Φ_2 are linearized at the point x_K to yield the two constraints Φ_1' and Φ_2'. The original problem has a unique minimum at $\hat{x}$ where the two non-linear inequalities intersect. However the linearized problem has no finite solution.

(2) The perturbations (δx in equations 4-4) may be positive or negative, whereas the variables in a linear program are confined to positive values.

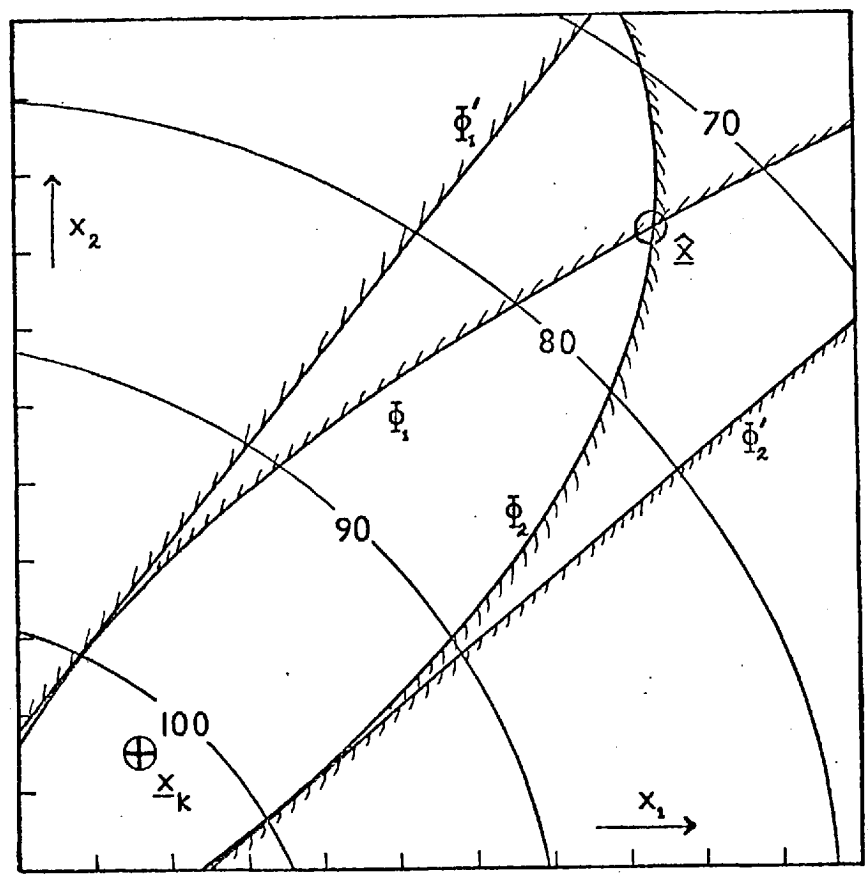

FIG. 4-1 Linearized Problem with Unbounded Solution

To solve the first problem Griffith and Stewart introduce step length limitation constraints of the form:

$$-\text{STEP}_i \leq \delta x_i \leq \text{STEP}_i$$

This prevents an unbounded solution but introduces further difficulties in that the STEP_i values have to be gradually reduced as the calculation proceeds to ensure convergence. It turns out that the manipulation of the STEP_i values as the calculation progresses is one of the most critical components of a successful algorithm using linear approximations.

The second difficulty was resolved by Griffith and Stewart by expressing each perturbation, δx_i, as the difference between two positive variables.

$$\delta x_i = \delta x_i^+ - \delta x_i^-$$

Since any real number, positive or negative, may be expressed as the difference between two positive real numbers, this satisfies the linear programming requirement for positive variables. However this solution is very costly in terms of both computer storage requirements and central processor time in that the dimensionality of any problem is automatically doubled. For problems with more than about ten variables the computing penalty rises very steeply indeed. In SLA this variable splitting technique is not used and examples are given in Section 6-3 to show the substantial savings in both computer time and core storage requirements for a fifteen variable problem.

4-5 The Displaced Origin Technique

As noted in Section 4-4 the chief disadvantage of the Griffith and Stewart formulation lies in the necessity to double the dimensionality of every problem. This disadvantage is removed in SLA using a coordinate transformation. Figure 4-2 shows a two-variable problem which

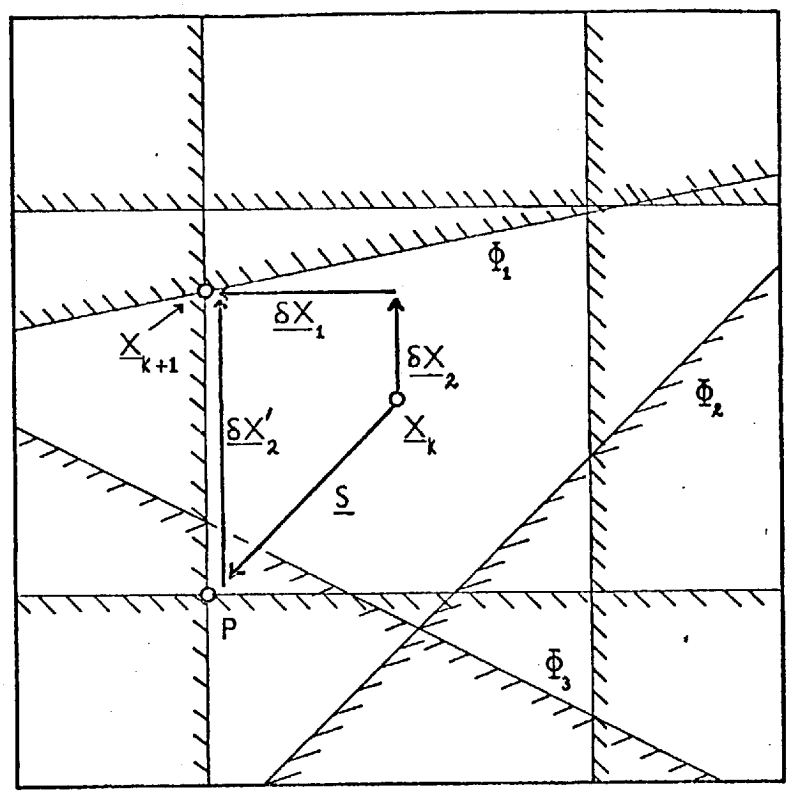

FIG. 4-2 Illustration of the Displaced Origin Technique.

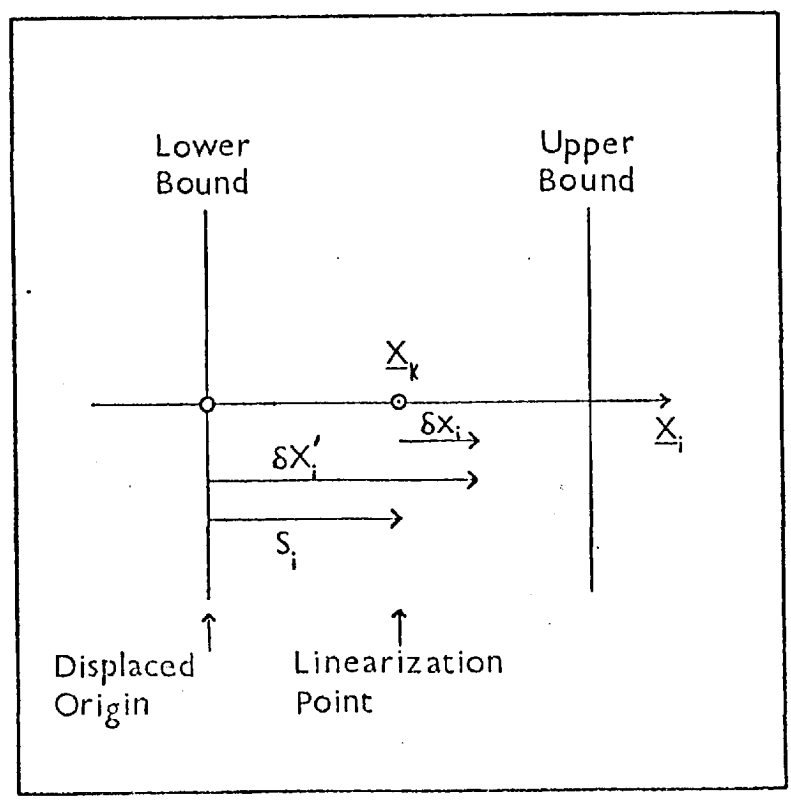

FIG. 4-3 Determination of the Elements of the Displacement Vector, S.

is subject to three inequality constraints (Φ_1, Φ_2 and Φ_3) and four step length limitations. Both object function and constraints have been linearized at the point $\mathbf{x}_K$. The solution to the linear problem is assumed to lie at $\mathbf{x}_{K+1}$ and thus $\delta x_1 < 0$ and $\delta x_2 > 0$. If, however, the problem is linearized as before at $\mathbf{x}_K$ and then transformed to a new set of variables $\delta x_1'$, $\delta x_2'$ such that point P in figure 4-2 becomes the origin, then the solution is still at $\mathbf{x}_{K+1}$ but the perturbations $\delta x_1'$ and $\delta x_2'$ are both greater than or, as for $\delta x_1'$ equal to zero. This is the basis of the displaced origin technique. There is no need to split the new variables as in the Griffith and Stewart formulation, moreover, since the linear programming algorithm automatically produces solutions which are greater than or equal to zero the lower bound step length limitations do not need to be incorporated explicitly. In general this means that for an n-variable problem there will be n fewer constraint equations. For large problems this in itself can lead to significant computational advantage over the original formulation.

From figure 4-3 it is clear that the origin displacement required for each independent variable is of the form:

$$\delta x_i + s_i = \delta x_i' \qquad (4\text{-}5)$$

- where s_i are the elements of the displacement vector $\mathbf{s}$ shown in figure 4-2.

Substituting from equation 4-5 into equations 4-4 and using the less compact but more explicit summation notation, the problem becomes:

$$\text{Minimize :} \quad \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} (\delta x_j' - s_j)$$

$$\text{Subject to :} \quad \sum_{j=1}^{n} \frac{\partial \Phi_i}{\partial x_j} (\delta x_j' - s_j) \geq -\Phi_i(x_K), \qquad (i = 1,m)$$

$$\sum_{j=1}^{n} \frac{\partial \Psi_i}{\partial x_j} (\delta x_j' - s_j) = -\Psi_i(x_K), \qquad (i = m+1,p)$$

These equations may be further simplified as:

$$\text{Minimize :} \quad \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \delta x_j' - K_1 \qquad (4\text{-}6)$$

$$\text{Subject to :} \quad \sum_{j=1}^{n} \frac{\partial \Phi_i}{\partial x_j} \delta x_j' \geq K_2^i - \Phi_i(x_K), \qquad (i = 1,m) \qquad (4\text{-}7)$$

$$\sum_{j=1}^{n} \frac{\partial \Psi_i}{\partial x_j} \delta x_j' = K_3^i - \Psi_i(x_K), \qquad (i = m+1,p) \qquad (4\text{-}8)$$

$$\text{- where} \quad K_1 = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} s_j \qquad (4\text{-}9)$$

$$K_2^i \quad \sum_{j=1}^{n} \frac{\partial \Phi_i}{\partial x_j} s_j, \qquad (i = 1,m) \qquad (4\text{-}10)$$

$$K_3^i \quad \sum_{j=1}^{n} \frac{\partial \Psi_i}{\partial x_j} s_j, \qquad (i = m+1,p) \qquad (4\text{-}11)$$

(Note: In SLA the constant K_1 is dropped from the object function since only the solution vector and not the value of the object function itself is of interest.)

Thus in order to set up a linearized problem using the displaced region technique all that is required is the displacement vector, S, and the partial derivatives of both object function and constraints. The

procedure for determining S is described in detail in Section 5-3-1.

4-6 Cubic Fitting

The most characteristic feature of algorithms using successive linearization is the oscillatory nature of the search. Unless positive action is taken it is possible for an algorithm to oscillate indefinitely (see figure 4-4) or make slow zig-zag progress (see figure 4-5). Several authors [*e.g.* Ref. 19] make use of the sign of the x increments (δx in equations 4-4) to detect an oscillatory variable. A succession of positive x-increments shows steady progress in the positive x-direction while a succession of negative x-increments shows progress in the negative x-direction. Alternating signs on x-increments indicate oscillation. Once oscillation is detected action is taken to reduce the maximum step length (*i.e.* the maximum value of $|\delta x_i|$) of the oscillating variable. There is usually provision for increasing the maximum step length by a constant factor following several steps in the same direction.

In SLA oscillation is recognised by more precise criteria than alternating increment signs. Also, step reduction is accompanied by a cubic fitting procedure which attempts to locate a minimum, if any, between the points of oscillation. This minimum is used as the linearization point for the following iteration. The motive for doing this is to locate the axis of the valley causing oscillation. On the axis the object function gradient is directed along the line of the valley which enables progress to be made down the valley without oscillation across it. A variable, x_i, is recognised as oscillating if, and only if, two conditions are met:

(1) The value of the variable at iteration number k is the same, within a small tolerance, as the value it had at iteration number k-2.

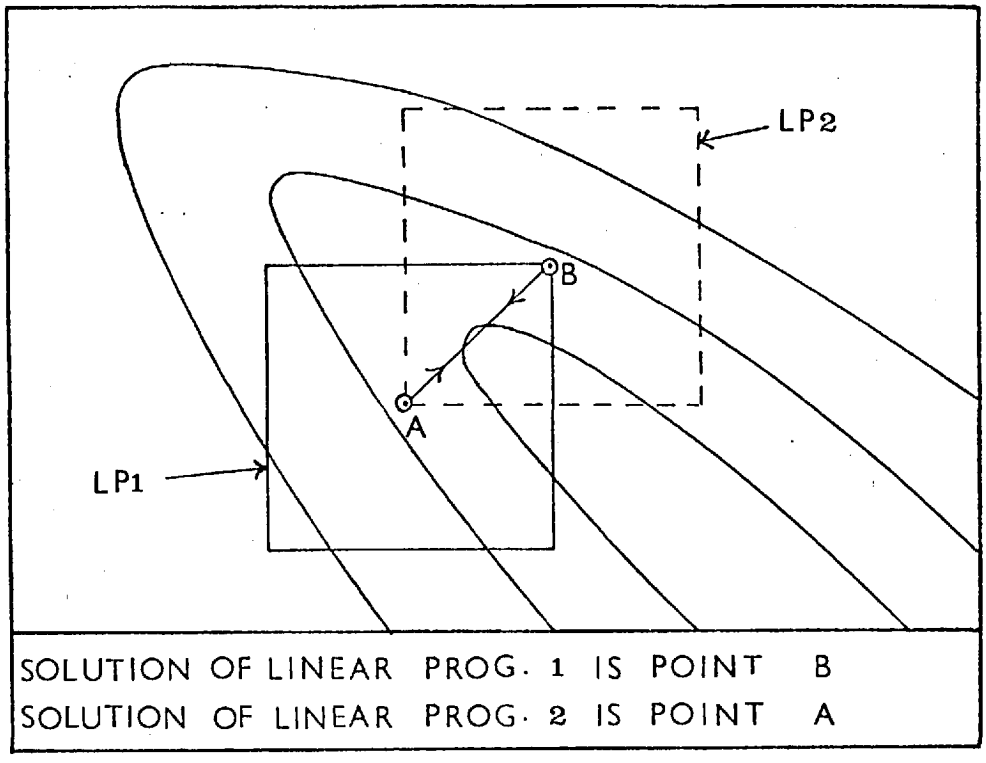

FIG. 4-4 Indefinite Search Oscillation.

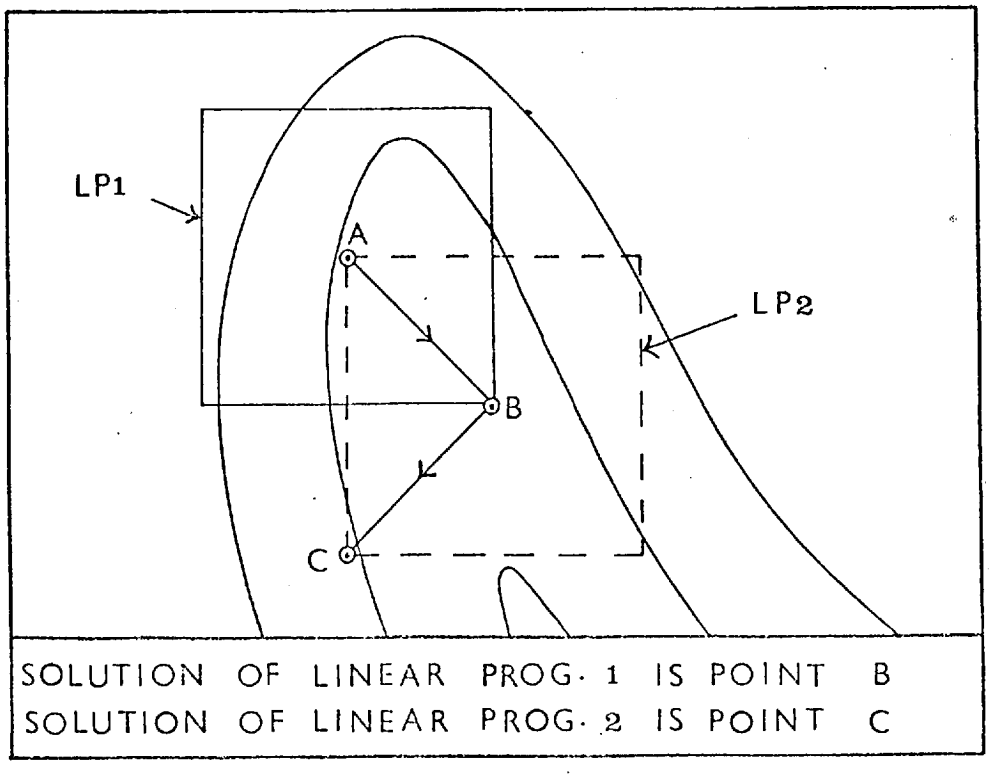

FIG. 4-5 Inefficient Zig-zag Search.

$$i.e. \quad \text{if,} \quad |x_i^K - x_i^{K-2}| \leqslant \varepsilon$$

(2) The value of the variable at iteration number k is not the same as it was for iteration number k-1.

$$i.e. \quad \text{if,} \quad |x_i^K - x_i^{K-1}| > \varepsilon$$

- where ε is a small tolerance factor.

(Note: This second condition ensures that stationary variables are not classified as oscillating.) If one or more of the independent variables is found to be oscillating then the cubic fitting procedure is used to estimate the position of the minimum, if any, lying between the current and previous points. The procedure is to determine a point $\tilde{\mathbf{x}}$ such that

$$\tilde{\mathbf{x}} = \lambda_m \mathbf{x}_{K-1} + (1-\lambda_m)\, \mathbf{x}_K \qquad (4\text{-}12)$$

- where $0 \leqslant \lambda_m \leqslant 1$ and is chosen to minimise a cubic approximation to the object function, $a(\lambda)$, between the points $\mathbf{x}_K$ and $\mathbf{x}_{K-1}$.

$$a(\lambda) = a_3\lambda^3 + a_2\lambda^2 + a_1\lambda + a_0 \qquad (4\text{-}13)$$

In the early stages of the development of SLA the coefficients a_0, a_1, a_2 and a_3 were determined using the two object function values $f(\mathbf{x}_K)$ and $f(\mathbf{x}_{K-1})$ and also the gradients at $\mathbf{x}_K$ and $\mathbf{x}_{K-1}$. However the calculation of these gradient values, if done numerically, involved for an n-variable problem n+1 object function evaluations in each case. For complex object functions this could involve a not insignificant amount of computing effort. For this reason the cubic fitting procedure was changed to use four function values only to determine the coefficients for the cubic fit. The function evaluations are evenly distributed between $\mathbf{x}_K$ and $\mathbf{x}_{K-1}$ on the line defined by equation 4-12 (*i.e.* at $\lambda = 0, 1/3, 2/3, 1$). If these four function values are denoted by f_1, f_2, f_3 and

f_4 then, by direct substitution into equation 4-13, we have:

$$\left.\begin{aligned} a_0 &= f_1 \\ (^1/27)a_3 + (^1/9)a_2 + (^1/3)a_1 + a_0 &= f_2 \\ (^8/27)a_3 + (^4/9)a_2 + (^2/3)a_1 + a_0 &= f_3 \\ a_3 + a_2 + a_1 + a_0 &= f_4 \end{aligned}\right\} \qquad (4\text{-}14)$$

The solution of equation 4-14 for the coefficients a_0, a_1, a_2 and a_3 is quite straightforward. The minimum of equation 4-13 is found by setting $\frac{\partial a}{\partial \lambda} = 0$ and solving the resulting quadratic equation which yields:

$$\lambda_m = \frac{-a_1}{a_2 \pm \sqrt{a_2^2 - 3a_3a_1}} \qquad (4\text{-}15)$$

Inspection of the second derivative of equation 4-13 shows that λ_m corresponds to a minimum when the square root in equation 4-15 has a positive sign. There are several ways in which this numerical procedure can break down when implemented on a finite arithmetic computer. In the cubic fitting subroutine checks are made for breakdowns such as zero or very small denominators in the expressions used to calculate the coefficients a_0, a_1, a_2 and a_3 and also in the calculation of λ_m. In such cases λ_m is set to a value corresponding to the lowest of the four function values. More complete computational details are given in Chapter 5.

It should be noted that the cubic fitting procedure is not an iterative search for a minimum but a 'once only' fit. The extra effort required to locate a minimum to high accuracy would probably not be justified in terms of increased computational efficiency except possibly for complex unconstrained problems.

Once λ_m has been determined from equation 4-15 or otherwise, the value of the 'fitted point', $\tilde{x}$ is found from equation 4.12. This point

is used as the linearization point for the next iteration. Before this next iteration begins however, the step lengths of the oscillating variables only are reduced to a value of FACRED.$|x_i^K - x_i^{K-1}|$. The value of the reduction factor, FACRED, is supplied by the user and will generally be in the range 0.1 to 0.4. The computer time required to solve a given problem is not sensitive to variations in FACRED. The choice of an optimum value for FACRED is discussed fully in Chapter 6.

4-7 Pattern Moves

Figure 4-6 shows the first few steps made by SLA on the well-known (unconstrained) Rosenbrock valley problem [Ref. 6]. The axis of this long curving valley is marked, and the effect of using cubic fitting is shown. The points marked 4F, 6F and 8F are the result of fitting cubic approximations to the object function between points 3 and 4, 5 and 6, and 7 and 8 respectively. It is clear from figure 4-6 that cubic fitting is able to locate the valley axis quite accurately. It is also clear that a 'pattern move' in a direction defined by two consecutive fitted points would produce a function decrease without the computational expense of setting up and solving a linear program. The procedure adopted by SLA is that whenever there are two consecutive fitted points a pattern move is attempted. The new point is first tested for feasibility with respect to both equality and inequality constraints, if any. An infeasible point is rejected and a new linear program is initiated from the last fitted point. If however the new point is feasible, the object function at that point is tested to see whether it is the best so far located. If it is not, then the procedure is as for an infeasible point. However if it is a best ever point, another pattern move of twice the size of the previous move is attempted. The rule for generating successive pattern moves is:

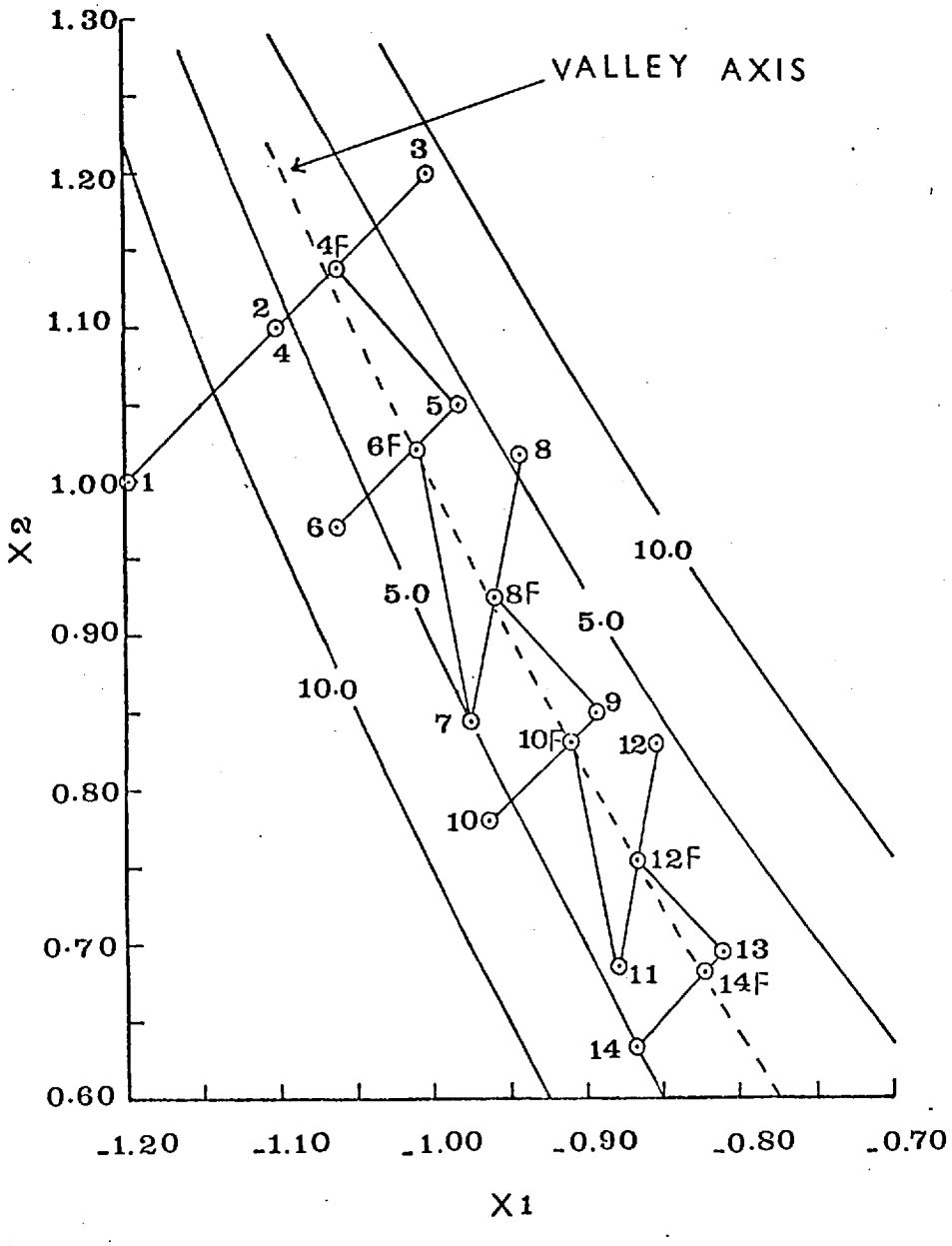

FIG. 4-6 Progress along Rosenbrock Valley using Cubic Fitting.

$$x_p^n = x_p^{n-1} + (x_f^{K+1} - x_f^K).2^{n-1} \qquad (4\text{-}16)$$

- where x_p^n is the n^{th} pattern move point.

x_f^K is the k^{th} fitted point.

(Note: $x_p^0 = x_f^{K+1}$, and so the first pattern move is

$x_p^1 = 2.x_f^{K+1} - x_f^K$)

The pattern moves are essentially unidirectional probes which terminate when a point is found which is either infeasible or not a 'best ever' point. Figure 4-7 shows the effect on the Rosenbrock function of using pattern moves. On unconstrained problems it has been found that the rate of progress is invariably enhanced by pattern moves. The effect on constrained problems is less marked but pattern moves usually provide some useful gains at relatively low computing cost.

For unconstrained problems there would almost certainly be an increase in efficiency by changing the probe to a unidirectional search for a minimum. However the algorithm was designed primarily for constrained problems and a unidirectional search which took account of constraints would undoubtedly make the algorithm more complex and would not necessarily produce significant gains.

The pattern move plays another important role in SLA apart from accelerating the search procedure. If two consecutive fitted points are the same element by element to within the input convergence criteria then a test is made for 'zero length pattern move convergence'. This test, which is described more fully in Chapter 5, checks the latest fitted point for feasibility and then checks it against the previous best ever point. If the fitted point is a 'best ever' point, or very close to it, convergence is declared and the algorithm terminates. This method of detecting convergence prevents a tendency of the algorithm to circle around the optimum point on some problems.

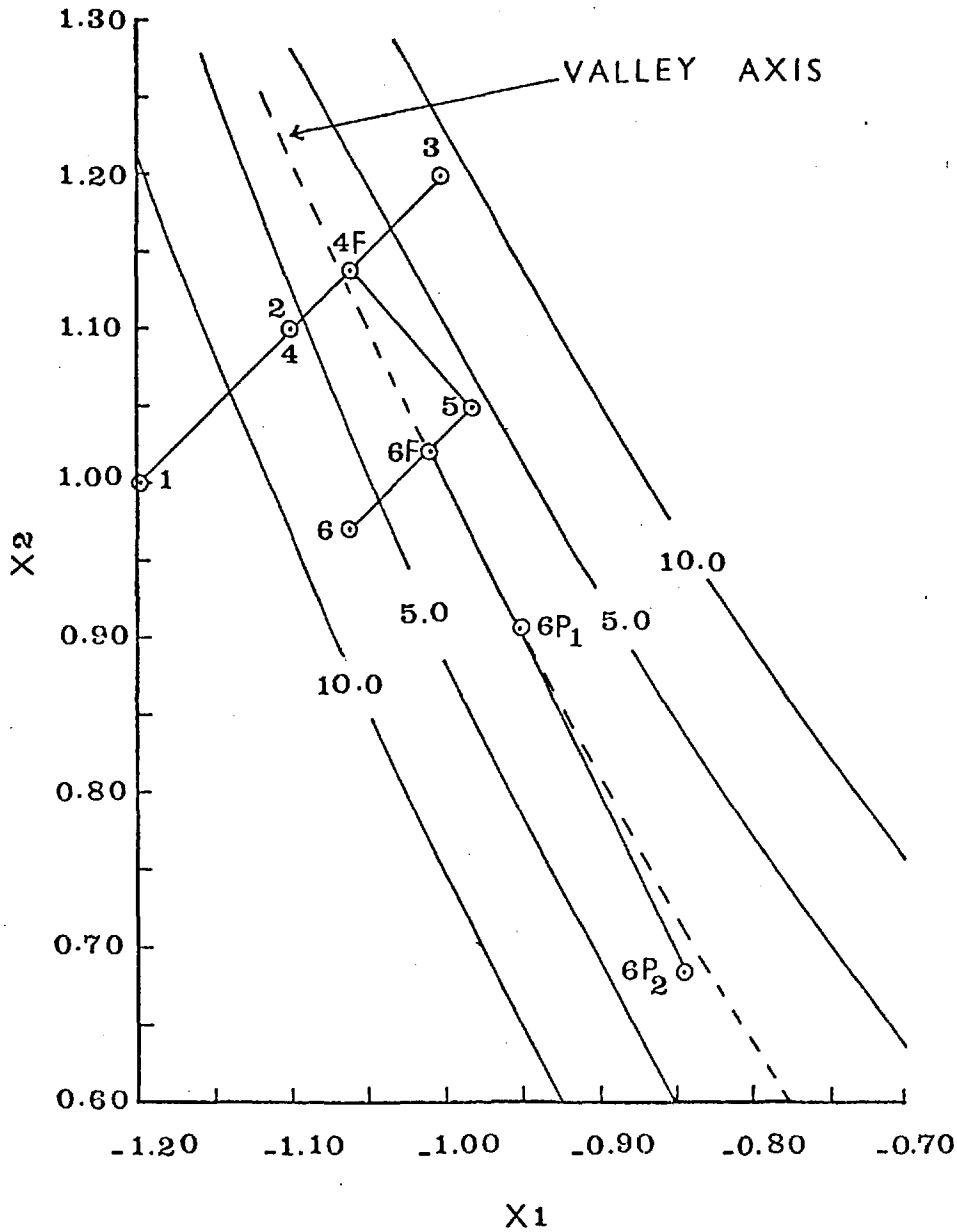

FIG. 4-7 Progress along Rosenbrock Valley using Cubic Fitting and Pattern Move.

CHAPTER 5

IMPLEMENTATION OF THE ALGORITH, SLA

5-1 Optimization Package

One of the objectives of the work described here is to provide a computer program which will be of use to others confronted with non-linear optimization problems. In this chapter a description is given of the program SLA which has been designed as an easy to use optimization package. Great importance has been attached to the ease with which the code may be used since one requiring laborious input is likely to be passed over in favour of another which is easier to prepare even though it may be less efficient. It was originally intended to require the user to provide a main calling routine as well as separate routines to calculate the object function and constraints. An optional routine to supply analytical first derivatives was also to be provided. However in order to reduce to the very minimum the amount of preparation to be done by the user, empty routines have been supplied so that the user is relieved of the chore of punching any cards which are not absolutely specific to his problem. This has also enabled the provision of counters in the user subroutines so that statistics of interest (*e.g.* number of function evaluations) can be conveniently collected. Perhaps most important of all it enables the documentation to be included as comment cards in the program itself. This is certainly the most direct and durable method of documentation. All that is required of the user is that he reads through the comment cards in the first section of the code and follows the instructions found there. A listing of this first section of the code is given in Appendix 2. The most demanding task for the user is probably that of assigning dimensions to the main arrays. However the instructions are explicit and it has only to be done once in the main

routine. All other routines have variable dimensions.

5-1-1 Structure of Optimization Package

Figure 5-1 shows schematically the relation between the various routines used in the SLA optimization package. A line joining two subroutines indicates that the lower routine is called by the higher. The routines shown in the centre section of figure 5-1 require no attention from the user. The other five routines must be completed by the user by the addition of information describing the problem. Details of the required information are given below.

(1) Main routine

Dimension main arrays.

Set upper and lower bounds on each independent variable.

Set: N = number of independent variables.

NCONS = number of inequality constraints.

NEQUS = number of equality constraints.

Set: IDERIV = 1, if analytical derivatives to be supplied, otherwise set to zero.

For I = 1,N set:

TEST(I) = Convergence criterion for X(I).

XSTRT(I) = Initial value for X(I).

STEP(I) = Maximum step length for X(I).

DELX(I) = Perturbation in X(I) for numerical derivative determination. (Only if IDERIV = 0).

Set four print control parameters.

(2) Subroutine UREAL

Provide FORTRAN statements which, given the value of X(I), evaluate the objection function, f.

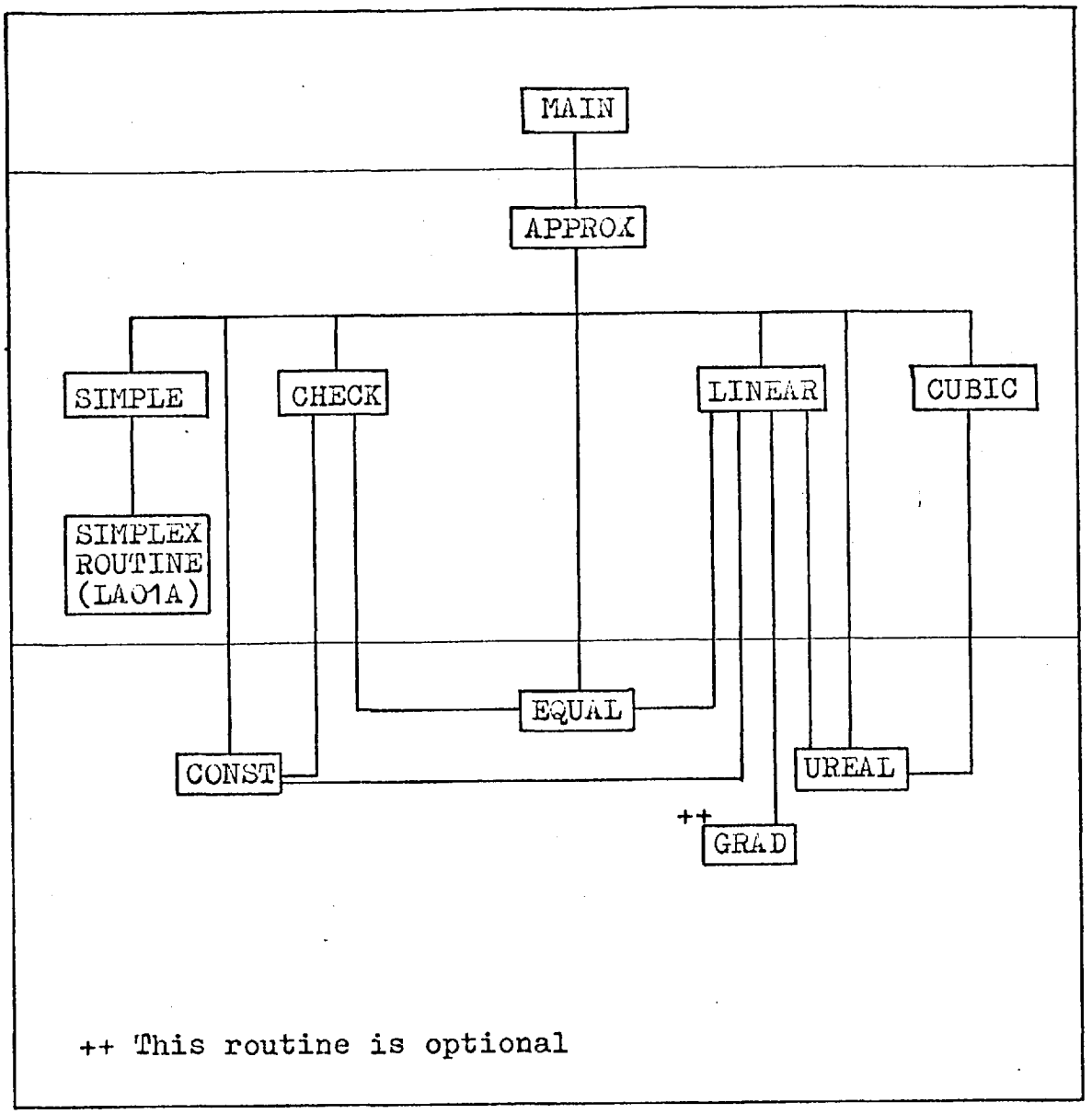

FIG. 5-1 Construction of SLA Optimization Package.

(3) Subroutine CONST

Provide FORTRAN statements which, given the value of X(I), evaluate each inequality constraint. These values are set in an array PHI(I), where I=1,NCONS. If there are no inequality constraints do nothing.

(4) Subroutine EQUAL

Provide FORTRAN statements which, given the value of X(I), evaluate each equality constraint. These values are set in an array PSI(I), where I=1,NEQUS. If there are no equality constraints do nothing.

(5) Subroutine GRAD

If IDERIV has been set to zero do nothing. Otherwise, provide FORTRAN statements which, given the value of X(I), evaluate the first partial derivatives of the object function and set them in order into the array DELX(I), I=1,N. If there are constraints then FORTRAN statements must be provided to set into the location (I,J) of the array CDERIV the first partial derivative of the I^{th} constraint with respect to X(J). Only non-zero elements need be entered. Inequality constraints must be entered before equality constraints.

Appendix 2 gives FORTRAN listings of these five routines and shows exactly how a complete problem is set up for solution by SLA.

5-1-2 Core Requirements for Optimization Package

The core (central memory) requirements for a FORTRAN computer program may be conveniently divided into three categories:

(1) Program instructions.

(2) Array storage requirements.

(3) System routines.

The core requirement for program instructions depends partly upon which compiler is used. Some compilers produce compact machine code requiring relatively little storage whilst other compilers, which may be designed primarily for fast execution, are less economical. The other, usually

dominant factor which determines the storage required for program instructions is the number of statements necessary to encode the problem itself. For example, one problem may require only a single line of FORTRAN to calculate the object function, whilst another may require a large and complex subroutine with many program instructions. Thus it is not possible to predict in advance exactly how much computer core will be required for any given problem. However, as an example, the shielding problem discussed in Chapter 7, which has 25 variables and 50 inequality constraints required approximately 4200 words of central memory for the search routines (*i.e.* APPROX, LINEAR, CUBIC, CHECK, SIMPLE and LAØ1A) plus 3600 locations for the problem specific routines (*i.e.* MAIN, CONST, EQUAL, GRAD and UREAL).

The array storage required by SLA for a particular problem is precisely predictable given the number of variables and constraints. If the number of variables is N, the number of inequalities NCONS and the number of equalities NEQUS then:

$$\text{Core Requirement} = (M+1)(M+5) + M(N+2) + 13N + NCONS \qquad (5\text{-}1)$$

- where $M = NCONS + NEQUS + N$

For the 25 variable shield problem equation 5-1 predicts a core requirement of 8480.

The size and number of system routines (*e.g.* input and output routines) will vary from installation to installation depending upon the operating system in use. For the shield problem the system routines required approximately 3400 central memory words. The total core requirement for the shield problem is thus 4200 + 3600 + 8480 + 3400, which is 19680 words of central memory.

In order to give an indication of the core requirements for other problems figure 5-2 has been constructed. It has been assumed for illustrative purposes that an average problem will require 11000 locations for program instructions and system routines. Also it has been assumed that this average problem has four times as many variables as equality constraints. This latter assumption is made since, given the core available, equation 5-1 is a function of three variables (N, NCONS and NEQUS) which cannot easily be plotted. Thus by setting NEQUS = N/4 in equation 5-1 it is possible to plot lines showing the maximum problem size attainable for a given central memory allocation as a function of inequality constraints and independent variables only. Therefore problems to the left of the line marked CM = 20000 in figure 5-2 have array requirements of 9000 or less. This is 20000 less the notional 11000 locations for program instructions and system routines. Problems to the left of the line marked CM = 40000 require arrays of 29000 or less. It must be emphasized that figure 5-2 can only give an indication of the core requirements. For example a problem may not have any equality constraints at all in which case extra inequalities could be accommodated for a given number of independent variables. Also variations in the core requirements for program instructions and system routines will displace the lines shown in figure 5-2.

The core requirements discussed above, if met, enable the code to run. However before this is possible the code must be first compiled and then loaded into the central memory. The amount of core required for each of these two operations may exceed the core required to execute the code. The core requirements of the compiler and the loader vary depending upon the particular compiler used and the operating system. The shielding code required 23300 locations to compile and 15400 to load on the Imperial College CDC 6400 computer.

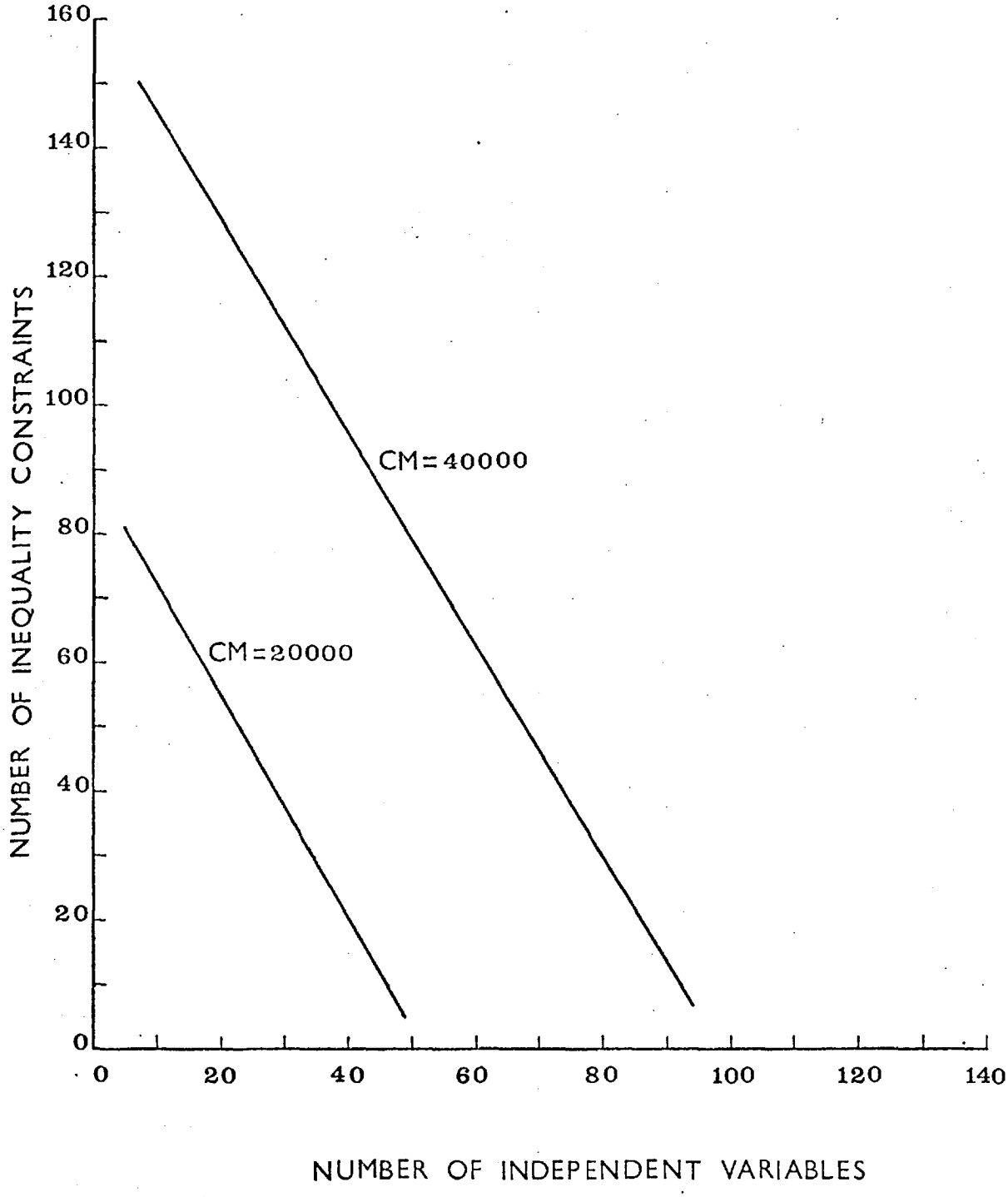

FIG. 5-2 Core Requirements of SLA.

5-2 The Subroutine APPROX

This routine is the largest and most complex part of SLA. It serves four main functions:

(1) To adjust maximum step lengths.

(2) To test for convergence.

(3) To form pattern moves.

(4) To increase step lengths if the linear programming routine finds no feasible solution.

Figure 5-3 shows in the form of a block diagram the principal tasks performed by APPROX. The preliminary housekeeping shown at the top of the diagram involves setting some constants which are used for variable dimensions in other subroutines. Also some data checking is done. For example, a check is made to ensure that the lower bounds on the independent variables are less than the upper bounds. The subroutine LINEAR which sets up the linear program is described in detail in Section 5-3 and will not be discussed here. The test for a failure of the linear program subroutine involves only the inspection of a variable, KO. This is set to 1 in the linear programming routine if it fails, and is otherwise zero. The loop involving a failure of the linear program only has relevance at the start of the calculation when the user may have set the initial step lengths too short to encompass the feasible region. If however linearization at the input starting point produces a problem which has no feasible solution at all this step doubling procedure cannot help and alternative procedures must be employed (see Section 4-3). In almost all the test problems on which SLA has been tried, once one successful linear program has been executed the feasible region is never lost; *i.e.* there are no subsequent linear program failures. Cases where feasibility is lost are discussed in Chapter 6. The new point generated by the linear program is tested for feasibility by first evaluating the inequality con-

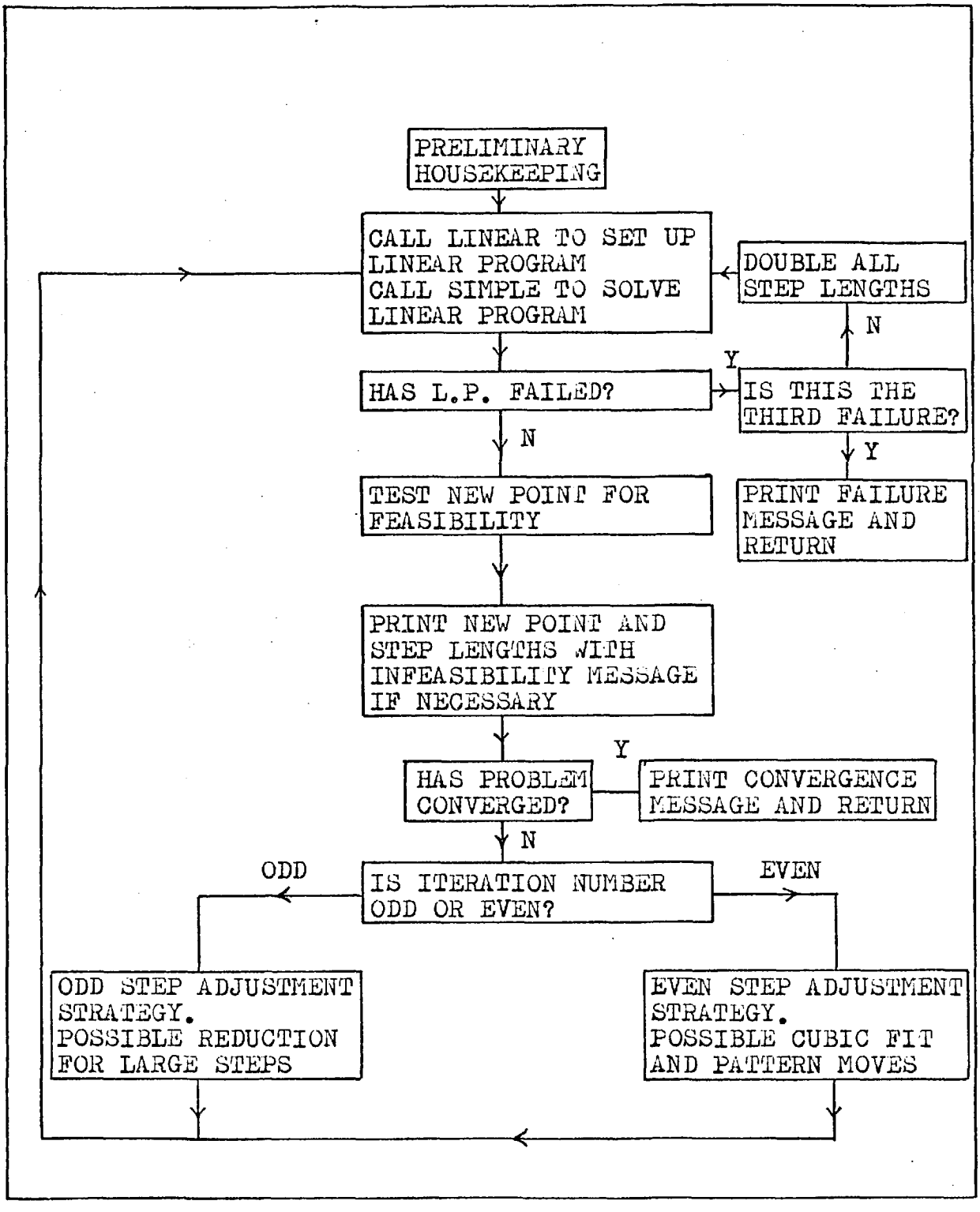

FIG. 5-3 Block Diagram of Subroutine APPROX.

straints at the new point. For strict feasibility the value of the constraints should be greater than or equal to zero. However, small violations (up to 10^{-6}) are tolerated. For most purposes this is more than adequate. If the new point is feasible with respect to the inequalities it is then tested for feasibility with respect to the equality constraints. If however the point is infeasible with respect to the inequalities an informative message is printed and no test is made for feasibility with respect to equality constraints. A point which is feasible with respect to inequalities but not equalities also produces an informative message. When satisfied equality constraints should be exactly equal to zero but, as for the inequality constraints, a tolerance of 10^{-6} is allowed. Therefore any point described as feasible by SLA will cause inequality violations of at most 10^{-6} and the equality constraints, when evaluated, will lie in the range -10^{-6} to $+10^{-6}$. The convergence tests which follow the feasibility tests are fully described in Section 5-6. The bulk of the routine APPROX is concerned with the step adjustment strategies. Odd and even iterations are treated separately. The procedure for odd iterations is brief compared to that for even iterations. Both procedures are described in some detail below.

5-2-1 Even Iteration Step Adjustment Strategy

The functions performed by the even iteration step adjustment strategy are fourfold:

(1) To identify variables which are oscillating, if any.

(2) If any variables are oscillating, to call the subroutine CUBIC and perform a cubic fit, at the same time reducing the step length of oscillating variables.

(3) If a cubic fit is performed and the last even iteration also involved a cubic fit, to attempt a pattern move.

(4) To increase by a constant factor (FACINC) the step length of any variable which has made two full length steps in the same direction.

These four functions are described below with the aid of flow diagrams.

The first stage involves labelling each variable as one of four types.

(1) Oscillating

(2) Stationary

(3) Moving

(4) Moving - last two steps were of full length and in the same direction.

The logic flow for this section is shown in figure 5-4. The array JELLY holds indicators from 1 to 4, which show the type of movement being made by a particular variable. For instance, if JELLY(7) = 1 this shows that X(7) is oscillating. In figure 5-4, X(I) is an n-element array containing the current values of the independent variables, and the array XSTRT(I) holds the values of the independent variables at the last but one iteration; *i.e.* the previous even iteration. The array XINC holds the changes made to the values of the independent variables by the current iteration. This means that the value of the I^{th} independent variable at the previous (odd) iteration was X(I) - XINC(I). The array OSCI(I) holds the oscillation tolerance factors for each variable. The tolerance factors used are:

(Input convergence criterion) × FACRED × 0.1.

Referring to figure 5-4 it is seen that variables which move by less than this oscillation tolerance factor are labelled as stationary (JELLY(I)=2). Further if a variable moves by more than the oscillation tolerance factor and its current value is the same, within the oscillation tolerance factor, as its value at the previous even iteration then it is deemed os-

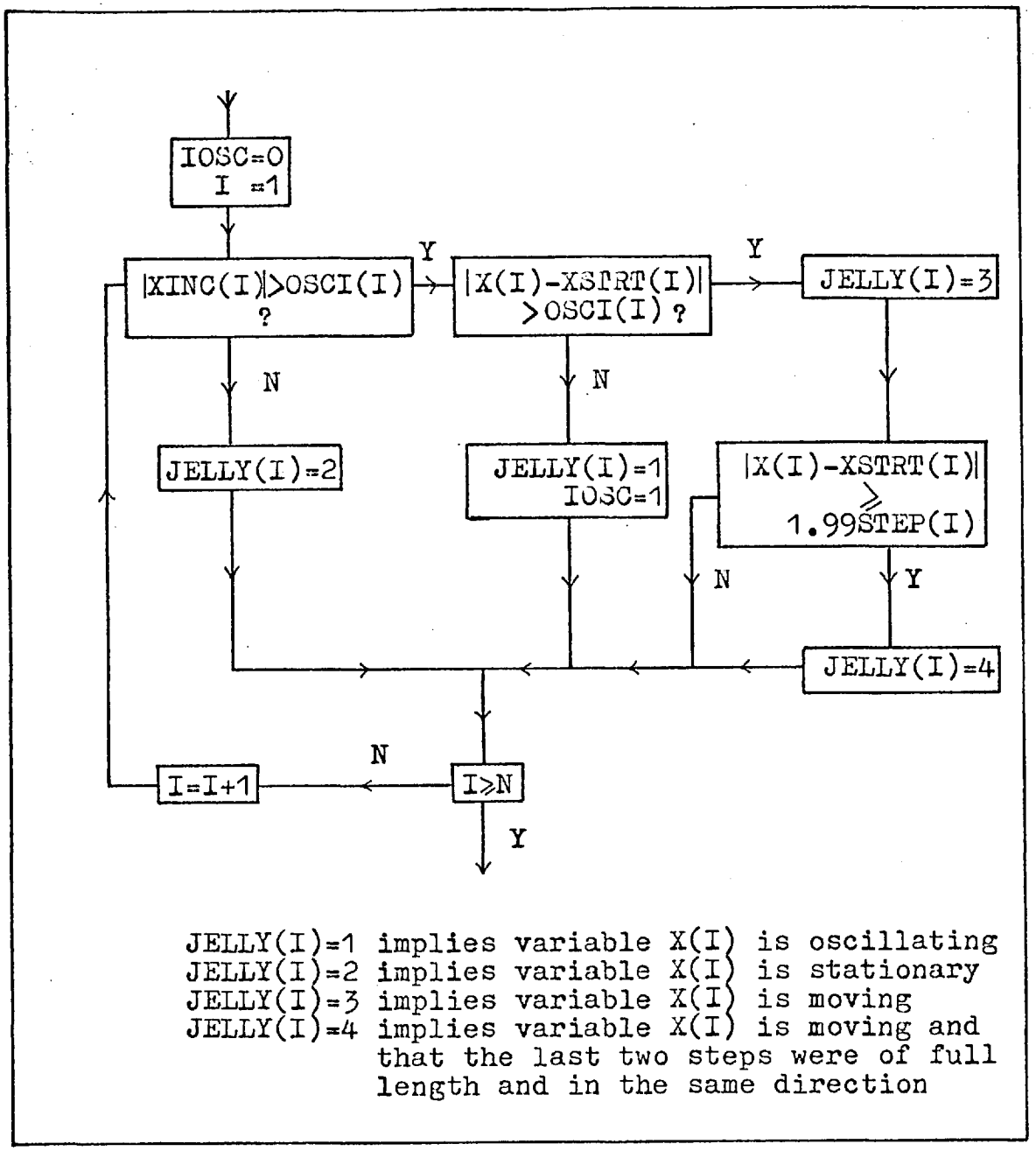

FIG. 5-4 Classification of Variable Movement.

cillating. Variables which are neither stationary nor oscillating are classified as merely 'moving', but those for which the distance travelled between the last even iteration and the current even iteration is greater than 1.99 × STEP(I) are classified as moving strongly in one direction (JELLY(I)=4). The array STEP(I) holds the maximum step length for each of the independent variables. If any of the independent variables is oscillating then the indicator IOSC is set to 1. This indicator is used to determine whether a cubic fit should be made in the next stage of the even iteration step adjustment procedure.

Figure 5-5 shows the next stage. Type 3 variables which move less than five percent of the maximum step length have this step length halved. If there has been no oscillation (IOSC=0) then this is the only step reduction which is made. However if at least one variable is oscillating then a call is made to the subroutine CUBIC which estimates the position of a minimum, if one exists, between the current point and the last point in the search. The subroutine CUBIC requires the value of the object function at the current point. The indicator IUCALC shows whether it has been calculated during earlier feasibility and convergence tests. If the object function has not been evaluated (IUCALC=0) a call is made to the subroutine UREAL to do this calculation. Computational details of the cubic fitting subroutine are to be found in Section 5-4.

The third stage of the even iteration step adjustment strategy involves the formation and testing of pattern moves. As described in Section 4-7 moves of this type are attempted following the determination of the second of two consecutive (even iteration) fitted points. Figure 5-6 shows the computational logic. The difference between the current and last fitted point is assessed on an element by element basis. This means that in order to be considered for a zero length pattern move convergence the change in the value of each independent variable must be

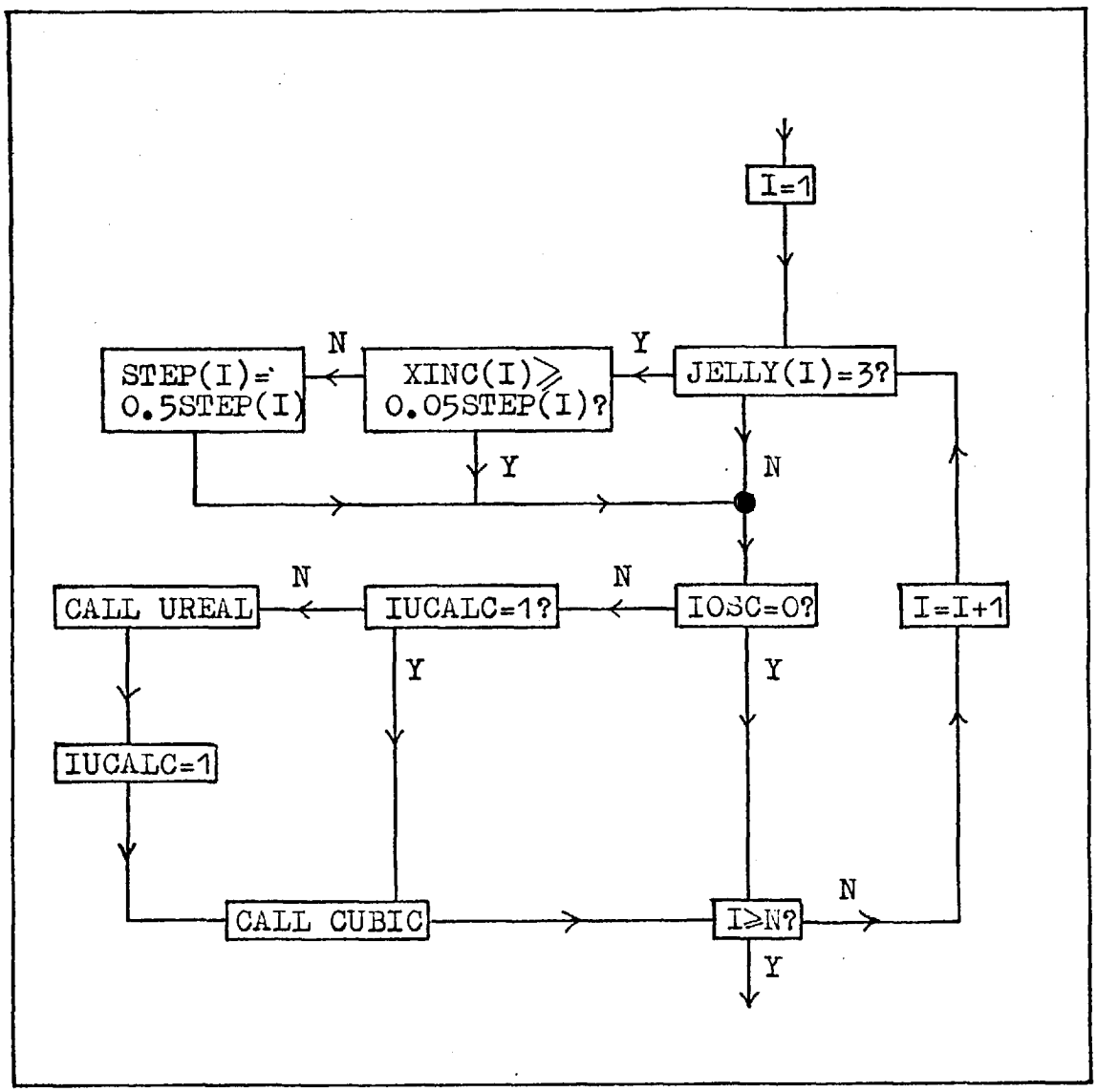

FIG. 5-5 Cubic Fitting Procedure in APPROX.

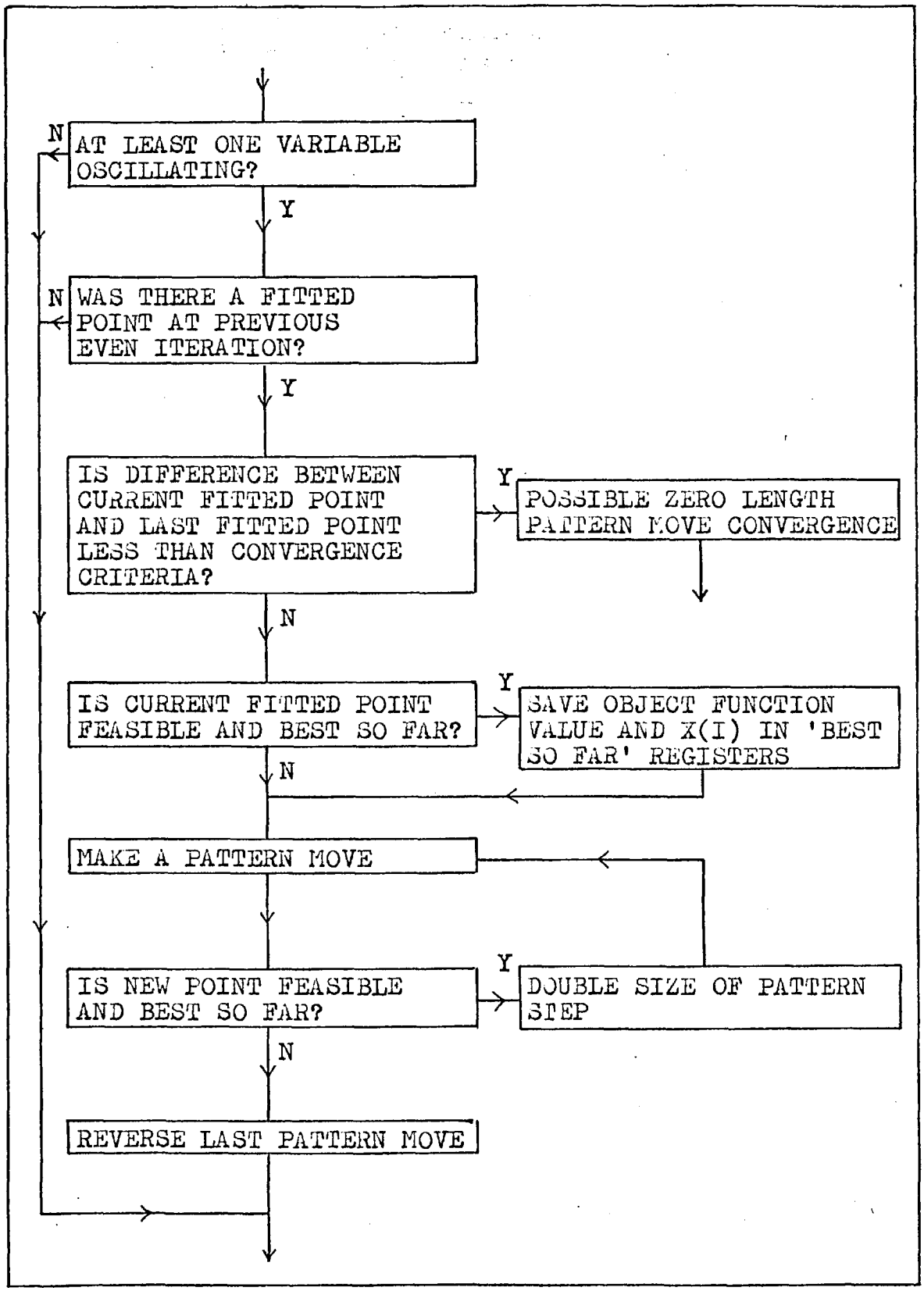

FIG. 5-6 Pattern moves in APPROX.

less than the prespecified convergence criterion for that variable. If at least one movement exceeds this value then a test is made, using the subroutine CHECK, to determine whether the current fitted point is both feasible and the best so far. If it is, it is saved in special locations and a pattern move is formed based upon the current and last fitted point. The rule for generating pattern moves is given in equation 4-16 of Section 4-7. The new point is tested against the joint criteria of feasibility and lowest object function value so far. If it is a successful move another pattern move is attempted. Otherwise the step is reversed and no further moves are attempted.

The fourth and final section of the adjustment strategy deals with type 4 variables only. The step lengths of these variables are increased by a factor FACINC provided that such an increase does not produce a step length which is greater than the maximum range of the variable. This maximum range is set to the difference between the upper and the lower bound for each variable.

5-2-2 Odd Iteration Step Adjustment Strategy

No classification of movement is necessary for the odd iteration step adjustment strategy. There are no changes in step lengths unless movement is less than five percent of the current step length and greater than the oscillation tolerance factor in which case the step length is halved. However there is one additional mechanism for step reduction. At every odd iteration which is an exact multiple of 5 (*i.e.* 5, 15, 25, ...) a check is made to see if any step lengths are more than 200 times the smallest step length. If there are any, they are reduced by the factor FACRED. This is done since large differences in step sizes could cause numerical difficulties in the linear programming routine. There is a test to ensure that this type of step adjustment does not cause the step lengths to be reduced to below the convergence criteria. This prevents

one rapidly convergent variable forcing convergence on all other variables.

5-3 The Subroutine LINEAR

The main task performed by this routine is to set up the equations required by the linear programming subroutine. Specifically, values are assigned to the arrays a_{ij}, b_j and c_i of equations 4-2 in Section 4-3. Figure 5-7 shows the structure of the routine. The control parameter IDERIV is set to zero by the user if numerical derivatives are required. In this case the array DELX must be given values δx_i to be used in formulae of the form:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, x_2 \ldots x_i + \delta x_i \ldots x_N) - f(x_1, x_2 \ldots \delta x_i \ldots x_N)}{\delta x_i} \tag{5-1}$$

These δx_i values are set in the main routine (see Section 5-1-1). If the user wishes to provide analytical first derivatives of both object function and constraints then IDERIV is set to 1 and FORTRAN statements for the evaluation of these derivatives must be added to the subroutine GRAD. The initial section of LINEAR takes the first partial derivatives of the object function and sets them into the array, c. Following this a test is applied, for unconstrained problems only, on the square of the gradient norm:

$$[\text{Gradient norm}]^2 = \sum_{i=1}^{N} \left[\frac{\partial f}{\partial x_i}\right]^2 = \sum_{i=1}^{N} [c_i]^2 \tag{5-2}$$

If the square of the gradient norm is less than 5×10^{-7} then convergence is assumed. This test gives a useful extra convergence mechanism for unconstrained problems. The next stage in LINEAR is the calculation of the displacement vector **s** (See Section 4-5).

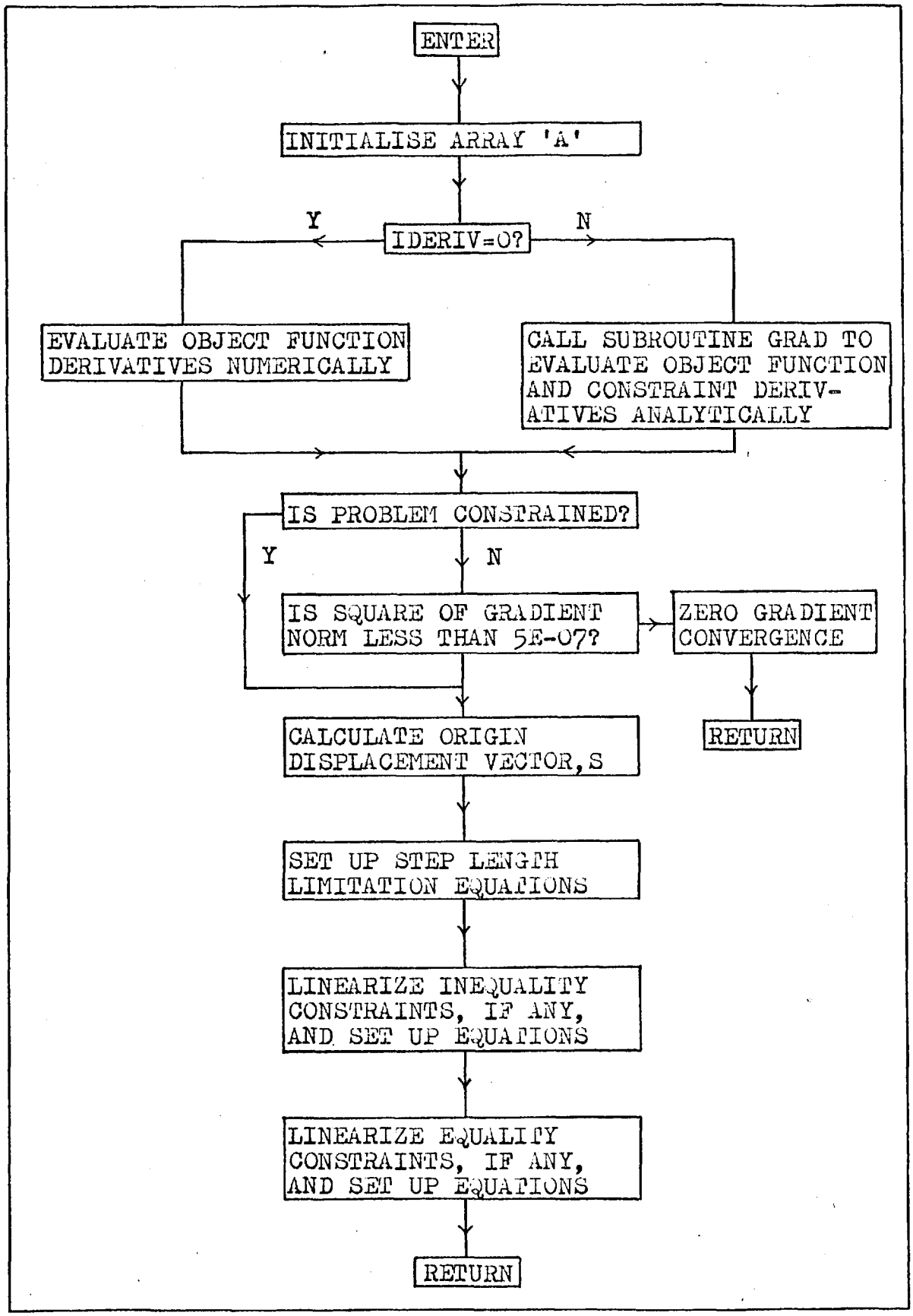

FIG. 5-7 The subroutine LINEAR.

5-3-1 Calculation of the Displacement Vector

As described in Section 4-5 the vector S has elements s_i which give the displacements required to change the origin of the linear program from the point of linearization to the intersection of the lower bounds. The lower bound for a variable may be either the input lower bound constraint or it may be the linearization point (current position) less the current variable step length. The situation is illustrated in figure 5-8. In this diagram the four possible arrangements of upper and lower bound constraints (marked x_U and x_L) with the step length limitations (marked by the two broken lines) are shown. The linearization point is at $\bar{x}_i$ and the appropriate displacement vectors are marked s_i. It is apparent that in general s_i should be:

$$s_i = \text{Min } \{(\bar{x}_i - x_L), \text{STEP}_i\}$$

- where STEP_i is the current step length for variable, x_i.

It should be noted that $\bar{x}_i$ will always lie between x_U and x_L.

5-3-2 Determination of Maximum Step Length

In the absence of upper and lower bound constraints the maximum step length allowed from the displaced origin would be simply twice the current variable step length, STEP_i. Figure 5-8 shows the situations which can arise when upper bound constraints are present. The maximum step length is seen to be the minimum of the distance between the displaced origin and the upper bound, and the distance between the displaced origin and the linearization point plus the current step length. That is:

$$u_i = \text{Min } \{(x_U - \bar{x}_i + s_i), (\text{STEP}_i + s_i)\}$$

Values of u_i are calculated for each variable x_i. In this way step length limitations and upper bound constraints are combined. The pro-

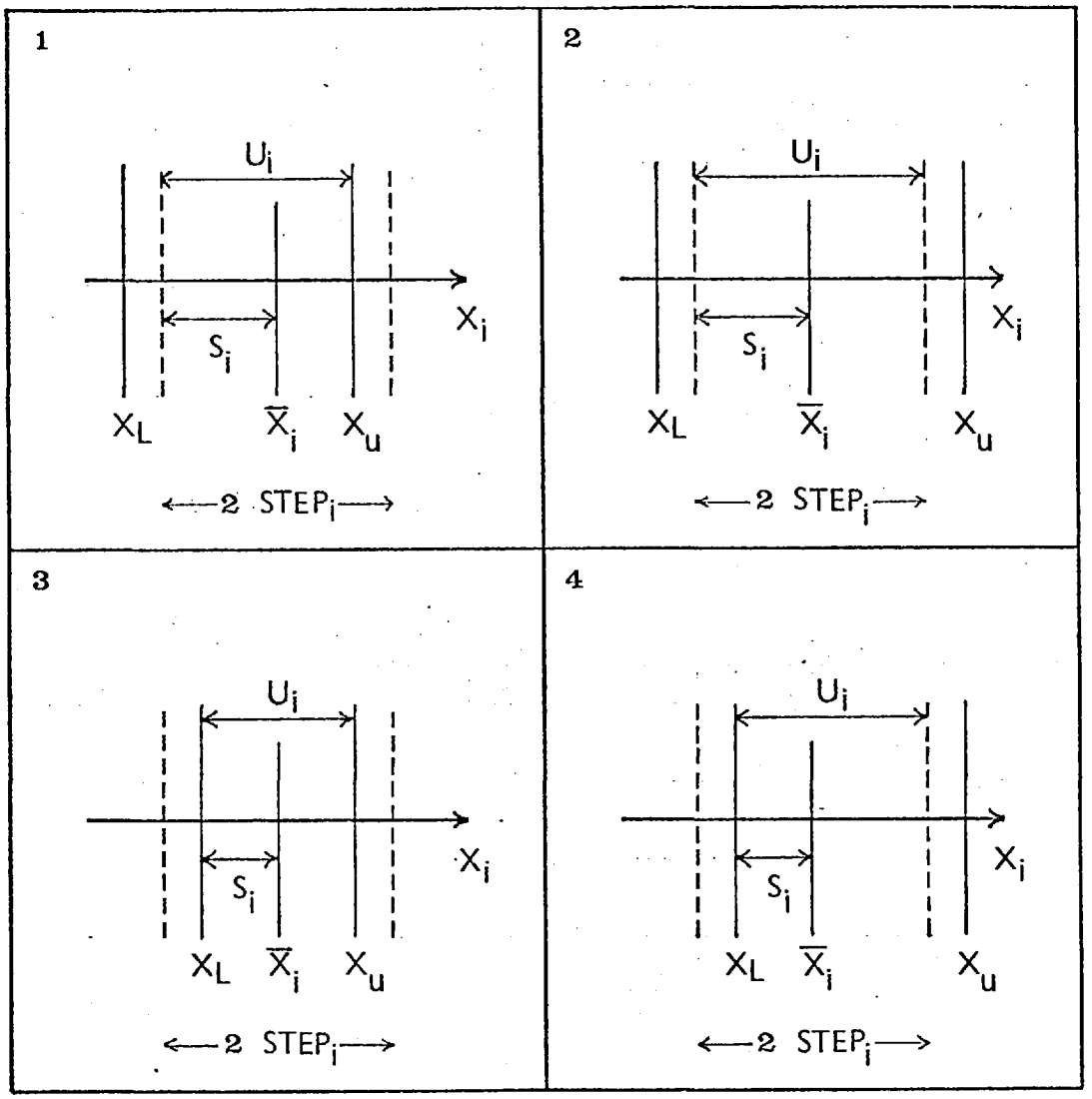

FIG. 5-8 Evaluation of Displacement Vector and Maximum Step Length.

cedure described above chooses the more restrictive of the two constraints. The values of u_i for each of the four cases are marked on figure 5-8.

5-3-3 The Linearization of Constraints

The linearization of the constraints is done either numerically or analytically. If done analytically (IDERIV=1) the partial derivatives are simply transferred from an array which the subroutine GRAD has generated. Numerical derivatives are calculated using the difference formula given in equation 5-1 of Section 5-3. Figure 5-9 shows schematically the way in which the input for the linear programming routine is prepared. The notation used is that of equations 4-2 of Section 4-3 and equations 4-6 to 4-11 of Section 4-5. The example given is a four variable problem having two constraints, one inequality, Φ_1, and one equality, Ψ_1. The first four rows of A set up the step length limitations. The values u_1, u_2, u_3 and u_4 are the upper limits on the step lengths and are obtained as described in the previous section. The next (fifth) row of A sets up the inequality Φ_1, and the last row arranges the equality constraint Ψ_1. The arrangement of A and B is dictated by the linear programming algorithm, LAØ1A, which is used with SLA. This routine requires that the inequalities precede the equalities in row order in A. There are two other considerations to be noted while using LAØ1A.

(1) Inequalities must be in the form:

$$\sum_{j=1}^{N} \frac{\partial \Phi_i}{\partial x_j} \delta x'_j \leq G_i \qquad (i = 1,m)$$

(Note reversed inequality as compared to equation 4-7.)

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\partial \Phi_1}{\partial x_1} & \frac{\partial \Phi_1}{\partial x_2} & \frac{\partial \Phi_1}{\partial x_3} & \frac{\partial \Phi_1}{\partial x_4} \\ \frac{\partial \Psi_1}{\partial x_1} & \frac{\partial \Psi_1}{\partial x_2} & \frac{\partial \Psi_1}{\partial x_3} & \frac{\partial \Psi_1}{\partial x_4} \end{bmatrix} \qquad b = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ G_1 \\ G_2 \end{bmatrix}$$

$$c = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} & \frac{\partial f}{\partial x_4} \end{bmatrix}^T$$

FIGURE 5-9

THE ARRAY A AND VECTORS B AND C AS PREPARED IN LINEAR

(2) Equalities must be in the form:

$$\sum_{j=1}^{N} \frac{\partial \Psi_i}{\partial x_j} \delta x_j' = G_i \qquad (i = m+1,p)$$

where $G_i \geq 0$ $\qquad (i = m+1,p)$

(*i.e.* the r.h.s. must be non-negative).

The first of these requires that both sides of the inequalities in equations 4-7 must be multiplied by -1 in order to reverse the inequality sign. Thus for the i^{th} inequality constraint:

$$G_i = \Phi_i(x_K) - \sum_{j=1}^{N} \frac{\partial \Phi_i}{\partial x_j} s_j \qquad (i = 1,m)$$

- where $\Phi_i(x_K)$ is the value of the inequality at the linearization point x_K. The partial derivatives are also evaluated at x_K and the s_i values are derived as shown in Section 5-3-1. The second requirement, that G_i be greater than or equal to zero for equality constraints is met by multiplying both G_i and the corresponding row of array A by -1 if G_i is less than zero. Therefore:

$$G_i = \pm \left(\sum_{j=1}^{N} \frac{\partial \Psi_i}{\partial x_j} s_j - \Psi_i(x_K) \right) \qquad (i = m+1,p)$$

It should be noted that the use of a different linear programming routine with SLA may require some changes in the format of the arrays A and B. Such changes will only involve minor modifications.

5-4 The Subroutine CUBIC

This subroutine attempt to locate a minimum lying between two points x_K and x_{K-1}, if one exists. The last qualifying phrase is important since when the routine is called it is not known whether there is a

minimum at all. As explained in Section 4-6 the subroutine originally used two function values and two gradients to fit the cubic equation. However in order to economise on the number of object function evaluations the final version of CUBIC uses just four function values only. Only the final version of CUBIC is described here. The most important objective in the design of this routine was that of numerical reliability. To this end four tests were made to determine whether a cubic fit is liable to run into difficulties. The first test takes the four function values f_1, f_2, f_3 and f_4 in two sets, $(f_1 f_2 f_3)$ and $(f_2 f_3 f_4)$ and tests:

(1) $f_1 < f_2$ and $f_3 < f_2$

(2) $f_2 < f_3$ and $f_4 < f_3$

If either of these two tests is positive, there is a maximum in the interval and cubic fitting to locate a minimum is inappropriate. The second test involves assessing the separation between the maximum and minimum values of the fitted cubic. This separation is expressed in terms of the cubic coefficients of equation 4-13 as:

$$\text{Root separation} = \frac{2}{3a_3}\sqrt{a_2^2 - 3a_3a_1}$$

If the turning values of the cubic fit are too close then the fit is rejected, since they represent only a small kink in an otherwise monotonic function. The test actually applied is that the root separation shall always exceed 0.5. Note that the fit is scaled so that the distance between the two end points of the fit is 1.0. The third test is applied to the denominator of the algebraic expression for λ_m given by equation 4-15. If this falls below 10^{-12} the fit is rejected so avoiding the possibility of program failure due to the generation of an 'infinite' result. The final test checks whether the calculated result lies between 0 and 1, *i.e.* within the interval of interest. Figure 5-10 shows a block diagram of the subroutine. The four tests described above are

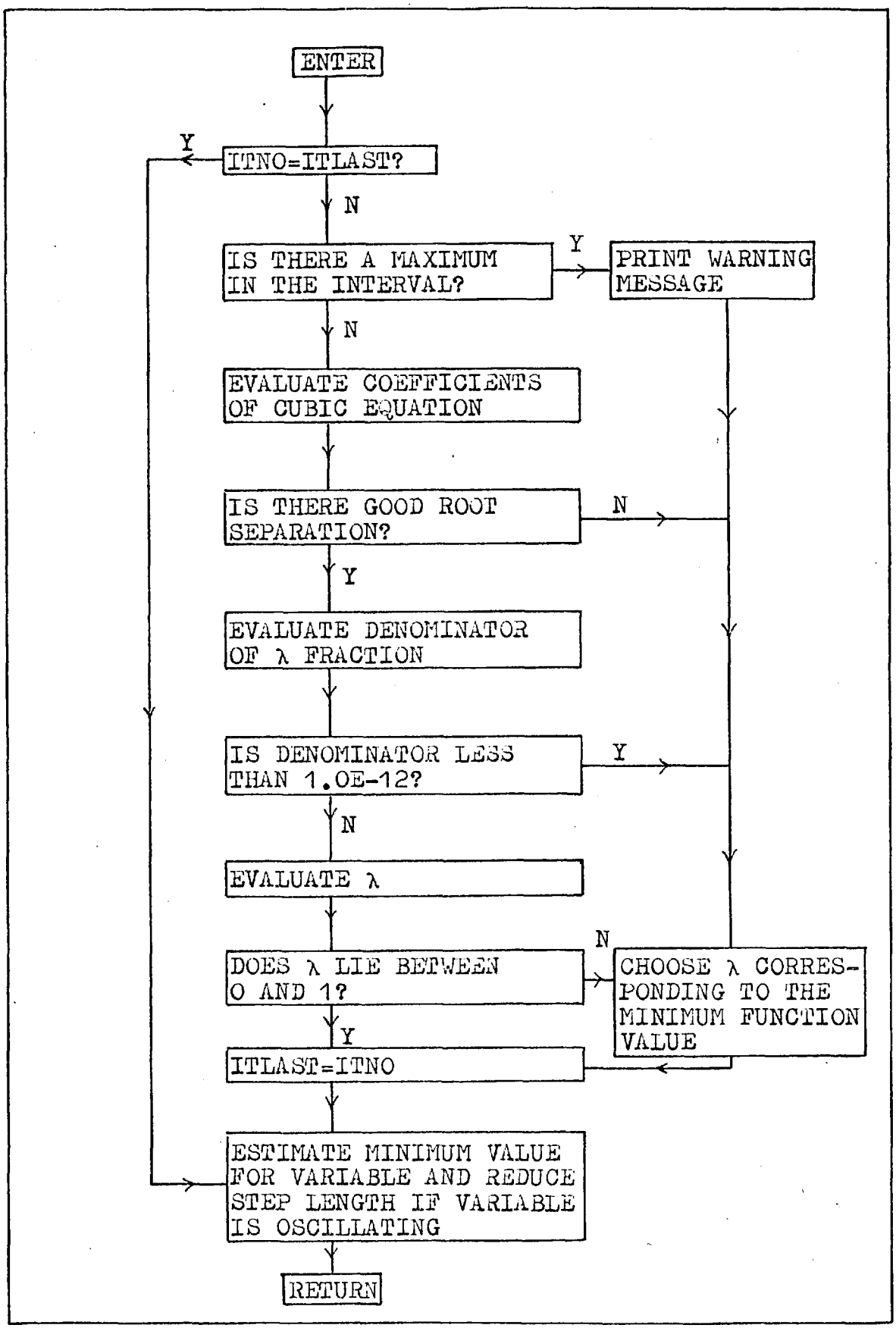

FIG. 5-10 Block Diagram of the Subroutine CUBIC.

shown. If any one of these tests fails the value of λ is chosen to correspond to the minimum of the four function values. This use of the minimum of the four function values is thus the 'back-up' to the cubic fit and is very reliable. The only possibility of failure arises where two or more function values take exactly the same minimum value. Since the 60 bit computer word on the CDC 6400 machine gives 12 significant figures such exact equality between function values is rare.

As shown in figure 5-5 the subroutine CUBIC is called once for each variable during the even iteration step adjustment strategy in cases where there has been at least one oscillating variable. It would obviously be wasteful of computing effort to repeat the same cubic fit for each variable in turn. Therefore once a value of λ has been determined for a given even iteration a variable ITLAST is set to the value of the current iteration number, ITNO. In subsequent calls to CUBIC on the same iteration ITLAST is equal to ITNO and λ is not re-evaluated.

Tests on a variety of problems have shown that root separation and small denominator failures are rare events. Maxima have been found in only one problem which involved one quadratic and one linear equality constraint on a quadratic object function. This problem (Number 12 in Appendix 1) was successfully converged. By far the most common reason for rejecting the cubic fit is that of a λ value greater than 1 or less than 0. Almost invariably this is because the object function is monotonic between the two end points. However a well-behaved cubic fit with a well-defined minimum is found on most problems most of the time.

5-5 The Subroutines SIMPLE and CHECK

The subroutine SIMPLE is called only by the routine APPROX and allows all the input data for the simplex routine to be printed. Normally this information is not required and the indicator IDATA is used to suppress the output. The output can be very useful for tracing errors in, for example, user-supplied analytical derivatives. However in normal circumstances SIMPLE is a transparent routine serving only to call the linear programming subroutine.

The subroutine CHECK takes a point X(I) and tests whether it is feasible and whether it has the lowest object function value so far. If the point satisfies both these conditions the value of X(I) is transferred to an array WORK19(I) and the corresponding value of the object function is stored as the variable ULAST. An indicator, IN2, is set to 1 to show that a new, feasible, best ever point has been found. The indicator is otherwise 0. Figure 5-11 shows the structure of CHECK. The elements of X(I) are only checked for boundary violations when X(I) has been obtained by a pattern move. The implementation of this check is controlled by the indicator IN1.

5-6 Modes of Convergence

There are four ways in which the code SLA detects convergence.

(1) The changes in the elements of X(I) between any two consecutive iterations are less than the convergence criteria.

(2) Two consecutive fitted points are the same, element by element, to within the convergence criteria.

(3) There is no change in the best (*i.e.* lowest) feasible function value in ten consecutive iterations.

(4) The magnitude of the local gradient is close to zero (unconstrained problems only).

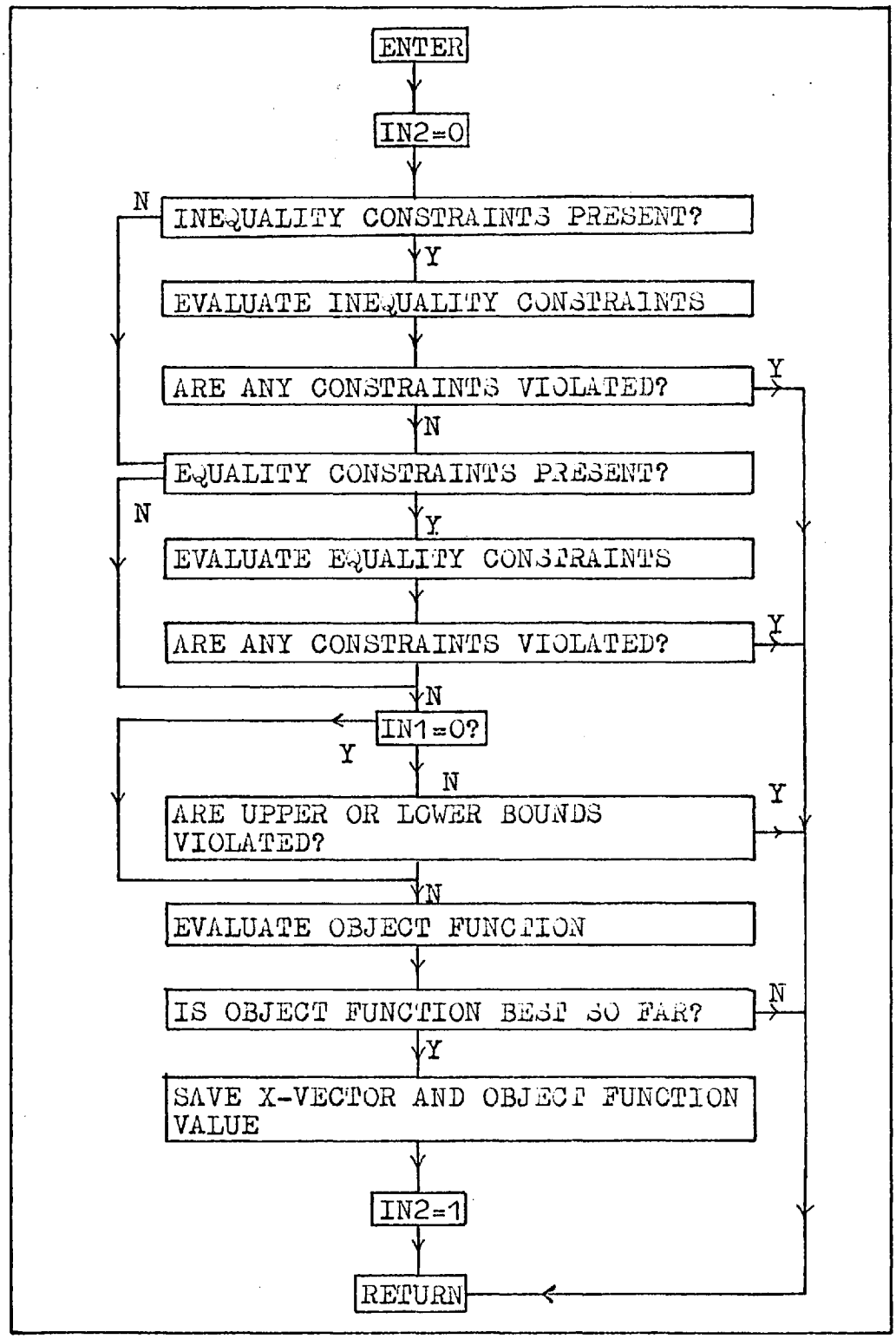

FIG. 5-11 Block Diagram of the Subroutine CHECK.

The convergence criteria used in the first two tests above are the convergence criteria supplied by the user (See Section 5-1-1) reduced by the step reduction factor, FACRED. This tightening of the convergence requirements is done since the cubic fitting procedure only estimates the position of a minimum. It is not an iterative search. This estimate may be in error by as much as the difference between the current and last iteration values of X(I). If convergence is achieved by gradual step length reduction, the tighter convergence criteria ensure that at least one cubic fit, and consequent step reduction, is required between points which are separated by no more than the user-supplied convergence criteria. In practical terms this means that the degree of convergence requested by the user is actually achieved.

The four modes of convergence are each discussed below.

Convergence Mode (1)

Convergence of this type is potentially the most efficient since, if the number of active constraints is at least equal to the number of independent variables, then convergence may be achieved without any step reduction. Under these circumstances the search routine effectively uses the linear programming routine to solve a set of non-linear simultaneous equations. Convergence is achieved when the same point is generated on two successive iterations. When the number of active constraints is less than the number of independent variables convergence is less rapid. The rate of convergence in such cases is principally dependent upon how far the step lengths are required to be reduced.

Convergence Mode (2)

Mode 2 convergence is of one of two types. In both these cases two consecutive fitted points are, element by element, the same to within the convergence criteria. However the most recent fitted point may or may not be the best so far (*i.e.* both feasible and having the lowest object

function value to date). If it is the best so far a message is printed:

'ZERO LENGTH PATTERN MOVE CONVERGENCE'

If the point is not the best so far a further test is made to see whether it is, element by element, within the convergence criteria of the best point so far located. If it is then a modified message is printed:

'BEST POINT SO FAR USED FOR ZERO LENGTH PATTERN MOVE CONVERGENCE'

Any points not satisfying either of these tests produce a message:

'PATTERN MOVE CONVERGENCE FAILURE IT.NO.'

- followed by the appropriate iteration number.

These warnings are not frequent occurrences and do not indicate that the program is in difficulty. Almost invariably zero length pattern move convergence follows within a few iterations of such a message. The zero length pattern move convergence mechanism has proved very successful in preventing the search circling around the optimum point.

Convergence Mode (3)

Convergence tests of this type are made at iteration numbers 5, 15, 25, 35 ... *etc.*. The test compares the function value, ULAST, which is the current best so far located, with the function value which was the current best so far located at the last test. This value is stored in a register UREF. If this has not changed by more than 1 in 10^6 then convergence is declared. Convergence can not be declared at iteration number 5 since UREF is given an initial value of 10^{50}. Before the convergence test is made a counter is checked to ensure that there have been at least two feasible points generated since the last test. This test is important since it is possible that, while SLA is following a non-linear constraint and making useful progress towards the optimum, a whole series of infeasible points are found. The points are infeasible because of the linear approximations being made. As the optimum is

approached step lengths are reduced, the approximation improves and feasible points are found. However since no infeasible point can be selected as a best point so far, a sequence of ten such infeasible points is sufficient to cause false convergence. The test is applied to ensure that at least two feasible points have been located since the last test was made. The choice of two feasible points is of course arbitrary but has proved sufficient to prevent false convergence.

Convergence Mode (4)

As stated earlier this mode of convergence only operates for unconstrained problems. It takes the form of a test on the square of the gradient norm. The test involves very little extra computing effort since the components of the gradient of the object function are required by the linear programming routine at each iteration. The test employed is:

$$\sum_{i=1}^{N} \left(\frac{\partial f}{\partial x_i}\right)^2 \leq 10^{-7}$$

If the test is satisfied convergence is declared. The value of 10^{-7} is arbitrary but may be altered by the user. It is true that a flat optimum or a saddle point could cause premature convergence, but since SLA has been designed for constrained problems no attempt has been made to implement more sophisticated tests.

CHAPTER 6

NUMERICAL EXPERIENCE WITH SLA

The results presented in this chapter demonstrate to what extent the algorithm measures up to the criteria discussed in Chapter 3. Computer resource requirements are assessed both in terms of central processor time used and, for the smaller problems, in terms of the number of effective function evaluations required for problem solution. The execution time of Colville's standard timing program [Ref. 16] was found to be 43 seconds on the CDC 6400 computer and this is used as a normalization factor for comparisons with published data. An indication of the core requirements for the algorithm is given for some of the larger problems. The results for the smaller problems have been obtained using explicit formulae for the first derivatives of both the constraints and object function. There is little, if any, deterioration in the algorithm performance using numerical derivatives. The results for the larger test problems have all been obtained using numerical approximations to the derivatives since this is by far the easiest method of problem preparation and hence the most likely to be used in practice. The use of analytical derivatives might in some cases reduce running times by a few seconds but usually at the expense of several hours preparation time.

The presentation of the results has been divided into two parts. The first of these shows the results obtained with the code N1OC. The problems in this section are all small. None has more than four variables. In some cases solution times required for various increment and reduction factor combinations are given. The sensitivity of the algorithm to the choice of initial step length is discussed. Some comparisons are made between results obtained with analytical and numerical derivatives.

The second set of results are for larger problems. These are taken from various sources and execution times are compared with published data wherever possible. For one problem (number 2) comparisons are made between SLA and an earlier version of the code which utilized the original Griffith and Stewart method of splitting the variables (See Section 4-4). The reductions in computational effort achieved by the use of the displaced origin technique are shown to be substantial.

6-1 Results for Small Test Problems

The small test problems have been divided into four groups:

Unconstrained problems.

Linearly constrained problems.

Non-linearly constrained problems.

Problems involving equality constraints.

Results and comparisons with published data are given for each problem in turn and, in Section 6-4, general conclusions concerning the performance of SLA on these small problems are drawn. Appendix 1 gives the formulation and solution for each of these problems.

6-1-1 Unconstrained Problems

Three well-known test problems, namely the functions of Rosenbrock, Powell and Wood are used in this section. It should be noted that results for unconstrained problems are given only in order to indicate that SLA is capable of solving unconstrained problems. SLA is not regarded as being a competitive algorithm for this type of problem.

The results obtained with the first test problem, Rosenbrock's function, are given in table 6-1. The two-variable function describes a long curving valley with a global minimum at the point (1.0,1.0). There are no local minima. The locus of the valley axis is given by $x_2 = x_1^2$ and is thus parabolic and symmetric about the x_2-axis. Figure

TABLE 6-1

RESULTS OBTAINED ON ROSENBROCK'S FUNCTION

Increment Factor		Reduction Factor 0.2	0.3	0.4
1.7	Mode	1	1	4
	Time	9.11	1.55	0.72
	f	0.12×10^{-1}	0.68×10^{-5}	0.82×10^{-7}
	EFE	4146	797	416
1.9	Mode	4	2	4
	Time	6.69	1.34	0.79
	f	0.14×10^{-6}	0.29×10^{-4}	0.16×10^{-6}
	EFE	3149	728	456
2.1	Mode	4	1	4
	Time	1.11	1.65	1.54
	f	0.38×10^{-6}	0.46×10^{-4}	0.19×10^{-8}
	EFE	626	899	829

Notes:
1. Modes of convergence are discussed in Section 5-6.
2. Times are C.P. seconds on CDC 6400.
3. f is optimum function value found.
4. EFE - effective function evaluations.

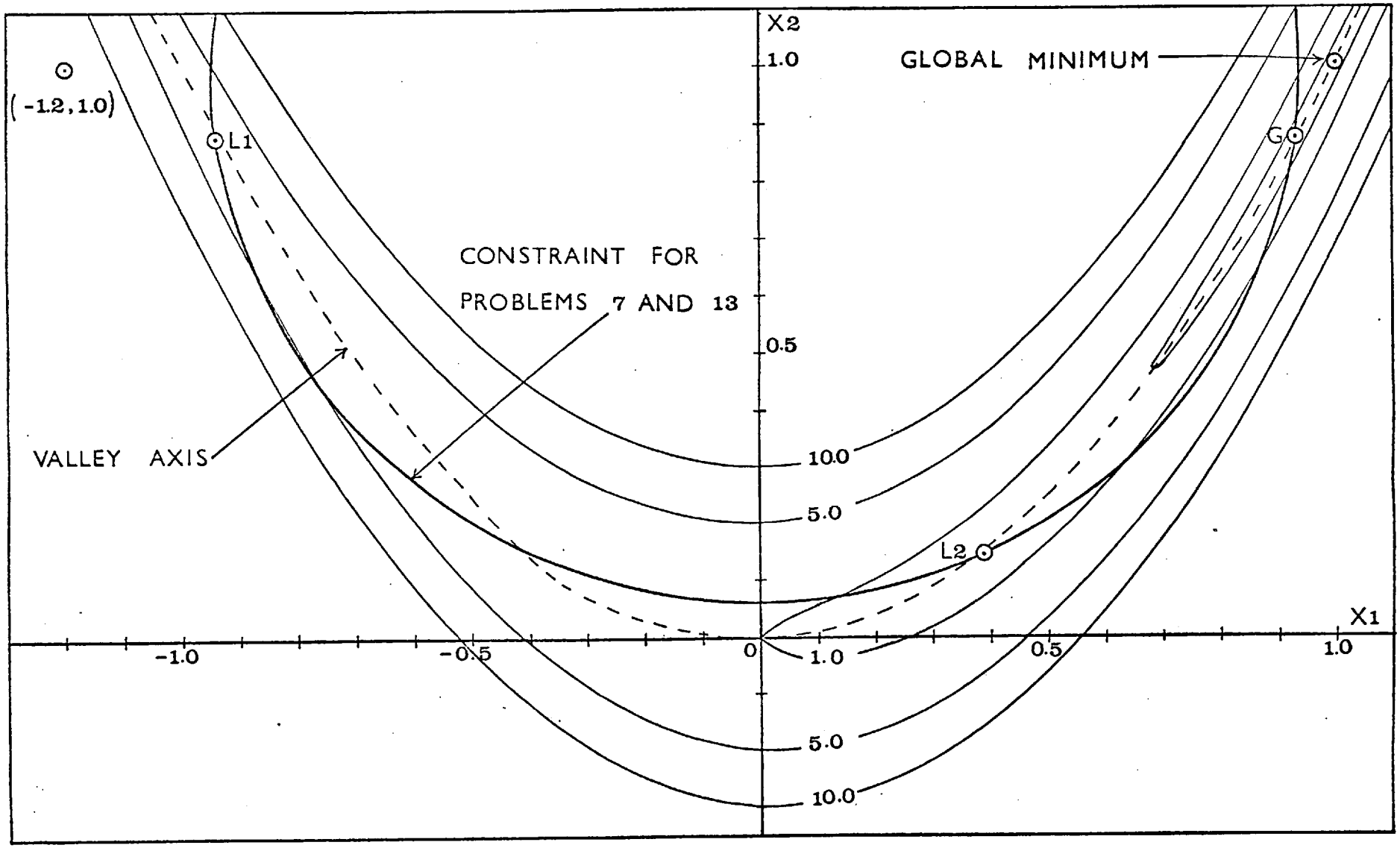

FIG. 6-1 Contours of Rosenbrock's Function.

6-1 shows the contours of this function. Rosenbrock's function is quite a severe test of the step increment and reduction strategies used in SLA. In the early stages progress must be made in the x_2-direction only. The step length of x_1 must thus be reduced rapidly and that of x_2 increased. As the origin is approached the x_1 step length must be expanded and the x_2 step reduced. Finally, as the optimum is approached the x_1 step must be again reduced and the x_2 step expanded. The results in table 6-1 show the mode of convergence, execution time, object function value and number of effective function evaluations required by SLA for nine different increment and reduction factor combinations. It is apparent that a reduction factor of 0.2 is too severe for this problem, causing premature convergence. Inspection of the detailed output reveals that step lengths in these cases become very small indeed and progress is correspondingly slow. The most common mode of convergence is zero gradient, (mode (4)). That is the square of the gradient norm is less than 5×10^{-7}. SLA does not have the quadratic convergence property of the most efficient unconstrained algorithms and so a compromise between high accuracy and excessive running time has to be made. However all but one of the results are of the order 10^{-5} or less which would probably be adequate for most purposes. In terms of function evaluations SLA does not compare well with other algorithms, the average value, excluding those for FACRED=0.2, being 688. Nelder and Meade [Ref. 5] quote 150 function evaluations for their simplex search procedure whilst Powell [Ref. 8] gives a figure of 151 for his algorithm. Dixon [Ref. 20] quotes 223 function calls for his hybrid simplex/quadratic hill climbing algorithm, ACSIM. No figures for execution time or core requirements are given by these authors.

The second test problem, Powell's function, is more complex being a quadratic with four variables. The solution vector for this problem is

(0,0,0,0) giving an object function value of zero. It is apparent from table 6-2 that SLA approaches this value quite closely. Unlike Rosenbrock's function a reduction factor of 0.2 does not cause a large increase in the computational effort required for solution. The average number of function evaluations and the average central processor time for the nine calculations shown in table 6-2 are 652 and 1.54 seconds respectively. Typically the algorithms of Powell and Nelder and Meade would require 200 to 250 function evaluations to obtain an object function value below 10^{-8}.

The third test problem is a four variable quadratic function. The results given in table 6-3 show that the minimum is accurately located by SLA in all cases. This is primarily because the cubic fitting procedure works much better on a quadratic function like Wood's function than on a quartic like Powell's function. Dixon [Ref. 20] quotes function evaluation requirements for his own algorithm and that of Powell as 395 and 522 respectively. He observes that the Nelder and Meade algorithms 'frequently failed to converge on this function'. The average function evaluation requirement for SLA for the nine calculations shown in figure 6-3 is 425. The average time requirement is 1.04 seconds (CDC 6400). For short initial step lengths SLA frequencly terminated on a saddle point with a function value of about 8.0. The inability of SLA to detect a saddle point is clearly a disadvantage but one shared by other algorithms using first derivative information only.

The results obtained with SLA on the three test functions show that unconstrained problems may be solved but not with any great economy in terms of function evaluations when compared to other algorithms specially designed for unconstrained minimization.

TABLE 6-2

RESULTS OBTAINED ON POWELL'S FUNCTION

Increment Factor		Reduction Factor 0.2	0.3	0.4
1.7	Mode	1	2	2
	Time	2.20	1.80	2.06
	f	0.11×10^{-4}	0.63×10^{-4}	0.21×10^{-4}
	EFE	919	804	902
1.9	Mode	2	2	2
	Time	1.57	1.03	0.72
	f	0.40×10^{-5}	0.68×10^{-6}	0.77×10^{-6}
	EFE	628	408	306
2.1	Mode	4	2	2
	Time	1.53	0.29	2.65
	f	0.15×10^{-5}	0.49×10^{-1}	0.16×10^{-4}
	EFE	624	123	1152

Notes:
1. Modes of convergence are discussed in Section 5-6.
2. Times are C.P. seconds on CDC 6400.
3. f is optimum function value found.
4. EFE - effective function evaluations.

TABLE 6-3

RESULTS OBTAINED ON WOOD'S FUNCTION

Increment Factor		Reduction Factor 0.2	0.3	0.4
1.7	Mode	4	4	4
	Time	0.64	0.84	1.23
	f	0.8×10^{-9}	0.3×10^{-9}	0.3×10^{-8}
	EFE	249	335	503
1.9	Mode	4	4	4
	Time	0.33	0.80	2.00
	f	0.6×10^{-9}	0.1×10^{-8}	0.3×10^{-8}
	EFE	137	335	797
2.1	Mode	4	4	4
	Time	0.54	0.71	2.24
	f	0.4×10^{-9}	0.5×10^{-9}	0.5×10^{-9}
	EFE	221	307	937

Notes:
1. Modes of convergence are discussed in Section 5-6.
2. Times are C.P. seconds on CDC 6400.
3. f is optimum function value found.
4. EFE - effective function evaluations.

6-1-2 Linearly Constrained Problems

Three linearly constrained test problems have been used. The first of these, Rosenbrock D, is Rosenbrock's function, as discussed in the previous section, with two constraints of $x_1 \leq 0$ and $x_2 \leq 0$. These constraints are of course simple bounds on the two variables and are treated as such by SLA. The solution to the problem lies at the origin where both constraints are active. As noted in Section 5-6 an important characteristic of SLA is that where the number of active constraints is equal to the number of independent variables convergence is rapid. The procedure is essentially the repeated solution of a set of simultaneous linearized equations. Each solution gives a better approximation to the point of intersection of the active constraints. In general two or three iterations are sufficient to locate the point of intersection to high accuracy (*i.e.* more than six significant figures). Under these circumstances the step reduction strategy plays little or no part in the convergence procedure. The factor governing the rate of convergence for a given starting point is the initial maximum step length. A very short initial step length can make many steps necessary to reach the region of the constraint intersection. The increment factor will influence the rate of approach to the solution point. Table 6-4 shows the results obtained for the Rosenbrock D problem with three different initial step lengths representing a variation of two orders of magnitude. The computation time in all cases is less than one tenth of a second. The smallest initial step length requires marginally more computing effort to attain a solution. Dixon [Ref. 20] quotes the requirement of his own algorithm, ACSIM, as 77 e.f.e. (effective function evaluations) and that of the SUMT penalty function technique as 142 e.f.e.. SLA requires between 23 and 29 e.f.e.

TABLE 6-4

RAPIDLY CONVERGENT SMALL PROBLEM RESULTS

Problem	Initial Step Length (All variables)	C.P. Time (CDC 6400)	EFE
Rosenbrock D	0.5	0.07	29
	5.0	0.05	23
	50.0	0.05	23
Post Office Box B	0.1	0.25	52
	1.0	0.11	24
	10.0	0.04	8
Sefton's Problem	0.001	0.26	51
	0.01	0.19	33
	0.1	0.12	17
Cattle Feed Problem	0.2	0.18	24
	2.0	0.11	13
	20.0	0.15	13
Rosenbrock Ridge	0.05	0.15	23
	0.50	0.09	15
	1.00	0.09	19

Note: All problems use an increment factor of 2.0 and a reduction factor of 0.2.

The second test problem, the Post Office Box A problem, has an object function which is a simple product of the three independent variables. There are upper and lower bounds on all three variables of 42.0 and 0.0, plus one linear constraint involving all three variables. As far as SLA is concerned this problem is more difficult than the previous one in that at the solution point only one constraint is active. Thus convergence is achieved by step reduction following oscillation and cubic fitting. The problem is therefore a good test of the sensitivity of the solution time to variations in increment and reduction factors. Table 6-5 gives execution times and e.f.e. requirements for four different increment and reduction factors and three different initial step lengths. The initial step lengths are chosen to span two orders of magnitude and are 0.1, 1.0 and 10.0. It is difficult to identify definite trends in these figures. This is not unexpected since whilst any single mode of convergence may show a smooth variation with increment and reduction factors the aggregate response of all the convergence mechanisms is less well-defined. Since the convergence of SLA on this problem is governed mainly by how rapidly the step lengths can be reduced, a large reduction factor (*i.e.* small in absolute size) may be expected to give the most rapid convergence, particularly for a large initial step length. This effect shows up in the column averages shown in tables 6-6 and 6-7 where a reduction factor of 0.1 gives the minimum average central processor time and e.f.e. requirement. There is no significant trend in execution time with increment factor. The solution to this problem is 3456.00 and this figure was achieved exactly in 44 out of the 48 calculations shown in table 6-5. The remaining four results differed by at most 2 in the fifth significant figure. All four of these calculations had a reduction factor of 0.4. This point will be returned to later.

TABLE 6-5

RESULTS OBTAINED ON POST OFFICE BOX A PROBLEM

Increment Factor	Initial Step Length	Reduction Factor							
		0.1		0.2		0.3		0.4	
		Time	EFE	Time	EFE	Time	EFE	Time	EFE
1.7	10.0	0.58	167	0.59	205	0.62	206	0.64	201
	1.0	0.50	134	0.70	194	0.71	194	0.50	134
	0.1	1.11	315	0.81	219	0.80	236	1.01	300
1.9	10.0	0.52	166	0.60	205	0.42	146	0.63	206
	1.0	0.44	128	0.50	136	0.43	134	0.45	134
	0.1	0.58	176	0.62	182	0.55	153	0.64	181
2.1	10.0	0.47	149	0.60	205	0.44	146	0.64	206
	1.0	0.35	89	0.51	154	0.90	256	0.45	136
	0.1	0.67	171	0.90	244	1.12	304	1.40	363
2.3	10.0	0.43	131	0.56	205	0.55	206	0.61	206
	1.0	0.44	137	0.32	83	0.32	83	0.64	195
	0.1	0.70	183	0.60	186	0.55	143	0.63	184

TABLE 6-6

POST OFFICE BOX A PROBLEM - AVERAGE NUMBER OF FUNCTION EVALUATIONS

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	205	206	212	212	209
1.9	157	174	144	174	162
2.1	136	201	235	235	202
2.3	150	158	144	195	162
Column Averages	162	185	184	204	

TABLE 6-7

POST OFFICE BOX A PROBLEM - AVERAGE EXECUTION TIMES

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	0.73	0.70	0.71	0.72	0.72
1.9	0.51	0.57	0.47	0.57	0.53
2.1	0.50	0.67	0.82	0.83	0.71
2.3	0.52	0.50	0.47	0.63	0.53
Column Averages	0.57	0.61	0.62	0.69	

Note: Times are C.P. seconds on CDC 6400.

The Post Office Box B problem is the last linearly constrained example used. It has the same object function as the A problem but the upper bounds on the first two variables are reduced to 20 and 11 respectively. This change in the variable bounds produces a solution point of (20,11,15) where three constraints are active. SLA is thus able to converge rapidly on this problem as it did on the Rosenbrock D problem. Table 6-4 shows results obtained with three different initial step lengths of 0.1, 1.0 and 10.0. As in the Rosenbrock D problem the longest step length gives the most rapid solution requiring only 8 e.f.e..

SLA has thus demonstrated its ability to solve small linearly constrained problems. The results obtained have all been tightly converged with the object function accurate to six significant figures in almost all cases. Direct comparison of the computer resource requirements of SLA with those of other computer codes is, for reasons outlined in Chapter 3, not easy. However table 6-8 compares the results obtained with SLA with results given by Dixon [Ref. 20] for several other algorithms in terms of effective function evaluations. The SLA results quoted for the two rapidly convergent problems (*i.e.* Rosenbrock D and Post Office Box B) are average values for the three initial step lengths. The results given for the Post Office Box A problem are the average values for a reduction factor of 0.2 and an increment factor of 2.1. This anticipates conclusions drawn later about the best values to use for these two parameters. From these results it can be seen that in all cases SLA requires fewer function calls than the two penalty function techniques. The modified simplex technique of Box is similarly less efficient than SLA. The hybrid technique of Dixon (ACSIM) is more efficient than SLA for the Post Office Box A problem but not for the other two. The two projection techniques of Goldfarb and Davies are however very efficient each requiring only 16 function evaluations for each of the Post Office Box problems. However

TABLE 6-8

FUNCTION EVALUATION REQUIREMENTS FOR THE SOLUTION OF LINEARLY CONSTRAINED PROBLEMS

Test Problem	Solution Technique							
	ACSIM (Dixon)	Box Complex	DFP with Carroll	Goldfarb	Davies	SUMT	SLA	
							EFE	CP Time (CDC 6400)
Rosenbrock D	77	-	-	-	-	142	25	0.06
Post Office Box A	64	310	204	16	16	256	201	0.67
Post Office Box B	131	-	328	16	16	292	28	0.13

table 6-4 shows that where the number of active constraints is equal to the number of independent variables SLA is competitive with these two techniques.

6-1-3 Non-linearly Constrained Problems

Rosenbrock's function with a circular inequality constraint was used as the first of three test problems involving non-linear inequality constraints. This problem, although it has only two independent variables, is a severe test of SLA since both the object function and the inequality constraint are highly non-linear. There is only one active constraint at the optimum point, and so far SLA convergence must be achieved by gradual step reduction as in the Post Office Box A problem. Table 6-9 gives the results of 48 calculations performed on the Rosenbrock C test problem. The increment and reduction factors range from 17 to 2.3 and 0.1 to 0.4 respectively and the initial step lengths span two orders of magnitude being 0.25, 0.025 and 0.0025. The results are given in terms of both execution time and number of function evaluations required. Six calculations are marked with a single asterisk to show they are of insufficient accuracy. This premature convergence occurs for all four calculations having a reduction factor of 0.4 and which use an initial step length of 0.25. The deviation from the true minimum ranges from 5% for the smallest increment factor to 20% for the largest increment factor. This tendency to premature convergence has already been noted in connection with the Post Office Box A problem for reduction factors of 0.4. The other two cases of premature convergence both have an increment factor of 1.7 and an initial step length of 0.025. The largest deviation from the true minimum is $\sim$3%. One other calculation, which is marked with two asterisks in table 6-9 shows a minor deviation of approximately 0.5% from the true minimum. Convergence in this case is caused by the best point remaining unchanged for ten consecutive iterations (*i.e.*

TABLE 6-9

RESULTS OBTAINED ON ROSENBROCK C PROBLEM

Increment Factor	Initial Step Length	Reduction Factor							
		0.1		0.2		0.3		0.4	
		Time	EFE	Time	EFE	Time	EFE	Time	EFE
1.7	0.25	0.82	321	0.81	342	0.85	365	0.28	115*
	0.025	0.33	127	0.24	89*	0.24	89*	0.36	130
	0.0025	1.11	414	0.67	262	0.33	123	0.35	135
1.9	0.25	0.90	382	0.63	277	0.59	254	0.41	171*
	0.025	0.30	123	0.27	106	0.29	115	0.23	93
	0.0025	0.80	304	0.68	263	0.31	117	0.36	134
2.1	0.25	0.69	276	0.60	258	0.86	362	0.70	283*
	0.025	0.27	105	0.21	81	0.32	126**	0.27	107
	0.0025	0.35	134	0.31	117	0.34	125	0.36	131
2.3	0.25	1.04	440	0.51	218	0.72	314	0.29	123*
	0.025	0.51	211	0.24	98	0.27	99	0.23	92
	0.0025	0.56	216	0.47	181	0.28	102	0.32	123

a mode (3) convergence). The convergence is premature since an early point in the calculation falls fortuitiously close to the true optimum. There is always a possibility that this can occur but it appears to happen so rarely that no modification to the convergence strategy has been made. However apart from the seven calculations mentioned above all 41 other calculations located the true minimum (a local minimum in fact) to six figure accuracy and satisfied the constraint to within 10^{-6}. Tables 6-10 and 6-11 present the data of table 6-9 averaged over the three initial step lengths. For this problem there is a decrease in average computing effort as the reduction factor increases in size. This is shown by the two sets of column averages. However this must of course be balanced against the possibility of premature convergence with a reduction factor of 0.4. The best increment factor on average is 2.1 but the overall variation is relatively small.

The second test problem used was the Post Office Box C problem. This has an ellipsoidal constraint. As for the previous problem a set of 48 calculations was performed. The initial step lengths used were 1.5, 0.15 and 0.015. Table 6-12 gives the results in detail and tables 6-13 and 6-14 give these same results averaged over the three initial step lengths. The column averages show little variation in computational effort with reduction factor. The row averages show that a small increment factor is preferable for this problem. All 48 calculations produced the same answer of -22.6274 with one exception. This calculation yielded -22.6271 and used a reduction factor of 0.4. The active constraint was in all cases satisfied to within 10^{-6}.

The third test problem was introduced by Dixon as Sefton's problem [Ref. 20]. Although it has only two variables it is awkward to solve because of its inherent poor scaling. However SLA deals with it very efficiently since there are two active constraints at the minimum. Table 6-4

TABLE 6-10

ROSENBROCK C PROBLEM - AVERAGE NUMBER OF FUNCTION EVALUATIONS

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	287	231*	192*	127*	209
1.9	270	215	162	133*	195
2.1	172	152	204	174*	176
2.3	289	166	172	113*	185
Column Averages	255	191	183	137	

TABLE 6-11

ROSENBROCK C PROBLEM - AVERAGE EXECUTION TIMES

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	0.75	0.58*	0.47*	0.33*	0.53
1.9	0.67	0.52	0.40	0.33*	0.48
2.1	0.44	0.37	0.50	0.44*	0.44
2.3	0.70	0.41	0.42	0.28*	0.45
Column Averages	0.64	0.47	0.45	0.35	

Note: Times are C.P. seconds on CDC 6400.

TABLE 6-12

RESULTS OBTAINED ON POST OFFICE BOX C PROBLEM

Increment Factor	Initial Step Length	Reduction Factor							
		0.1		0.2		0.3		0.4	
		Time	EFE	Time	EFE	Time	EFE	Time	EFE
1.7	1.5	1.11	267	0.91	203	0.95	227	0.81	204
	0.15	0.78	172	1.05	251	0.94	222	0.77	188
	0.015	1.05	241	0.84	196	0.91	222	0.91	246
1.9	1.5	0.97	231	1.02	232	1.00	253	0.90	251
	0.15	1.08	247	0.99	275	0.76	209	0.90	245
	0.015	0.77	196	0.77	212	0.81	198	0.92	270
2.1	1.5	1.02	240	0.92	226	1.12	273	0.77	203
	0.15	0.70	217	0.89	259	1.22	307	0.89	242
	0.015	1.02	293	1.32	307	1.07	263	1.27	316
2.3	1.5	0.94	247	0.96	262	0.82	241	0.62	194
	0.15	0.87	234	1.14	303	1.04	264	0.90	287
	0.015	1.21	319	0.89	213	1.16	272	1.04	264

TABLE 6-13

POST OFFICE BOX C PROBLEM - AVERAGE NUMBER OF FUNCTION EVALUATIONS

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	227	217	224	213	220
1.9	225	240	220	255	235
2.1	250	264	281	254	262
2.3	267	259	259	248	258
Column Averages	242	245	246	243	

TABLE 6-14

POST OFFICE BOX C PROBLEM - AVERAGE EXECUTION TIMES

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	0.98	0.94	0.93	0.83	0.92
1.9	0.94	0.93	0.85	0.91	0.91
2.1	0.91	1.05	1.14	0.98	1.02
2.3	1.00	1.00	1.00	0.85	0.96
Column Averages	0.96	0.98	0.98	0.89	

Note: Times are C.P. seconds on CDC 6400.

shows the results obtained with three different initial step lengths 0.1, 0.01, 0.001. The execution time ranges from 0.12 to 0.26 seconds. Function evaluation requirements are also small ranging from 17 to 51.

From these three problems it may be concluded that SLA can solve problems involving highly non-linear constraints. It does so with moderate efficiency when optimum points are lightly constrained (*i.e.* the number of independent variables exceeds the number of active constraints) and with high efficiency when the optimum point is fully constrained. Except in a few cases the solution obtained by SLA is to high accuracy. The results in table 6-9 show that the choice of initial step length can affect the execution time and function evaluation requirements for the Rosenbrock C problem by a factor of three or more in some cases. However table 6-12 shows the solution times for the Post Office Box C problem to be much less sensitive to the initial step length chosen. This difference in sensitivity between the two problems is probably caused by the differing complexities of the object functions. The Post Office Box function is monotonic and is represented quite well by a linear approximation. The Rosenbrock function however is not so well represented by linear approximations and the choice of initial step length becomes more important. This problem of initial step length choice will be returned to later.

Table 6-15 gives a comparison of the performance of SLA in terms of effective function evaluations with several other algorithms. The comparative figures are again taken from Dixon [Ref. 20]. It can be seen that as for the linearly constrained problems SLA is more efficient than the penalty function technique of Fiacco and McCormick (SUMT) or the combination of Corroll's penalty function with the unconstrained DFP minimization method. The hybrid algorithm of Dixon performs well on the first problem but attains relatively poor accuracy (-22.5958 instead of

TABLE 6-15

FUNCTION EVALUATION REQUIREMENTS FOR THE SOLUTION OF NON-LINEARLY CONSTRAINED PROBLEMS

Test Problem	Solution Technique						
	ACSIM (Dixon)	DFP with Carroll	Goldfarb	Davies	SUMT	SLA	
						EFE	CP Time (CDC 6400)
Rosenbrock C	103	-	-	-	200	152	0.37
Post Office Box C	180	376	384	136	296	264	1.05
Sefton's Problem	104	-	-	-	228	33	0.19

-22.6274) on the Post Office Box C problem. The projection technique of Goldfarb does not perform well.

6-1-4 Problems Involving Equality Constraints

The first problem used to test the ability of SLA to deal with equality constraints was the Cattle Feed problem. This has four variables, one linear equality constraint and two non-linear inequality constraints. In addition, lower bounds of zero are imposed on all four variables. There are four active constraints at the solution point and so SLA converges rapidly. Table 6-4 shows the results of three calculations with initial step lengths of 0.2, 2.0 and 20.0. Even with the shortest step length only 24 function evaluations are required whilst for the two other step lengths the requirement is reduced to 13. This may be compared with figures quoted by Dixon [Ref. 20] of 410 function calls for the SUMT algorithm and 141 function evaluations for his algorithm ACSIM. The solution obtained by SLA of 29.8888 is accurate to six significant figures.

The second test problem used has been specially devised to test SLA and is referred to as the Rosenbrock Ridge problem. As the name suggests the object function is again that of Rosenbrock's function. The problem is to maximize the object function (*i.e.* minimize minus the object function) subject to the equality constraint $x_2 = x_1^2$. This equality constraint is the equation defining the locus of the valley axis. This has the effect of binding the search for a maximum value to the floor of a steeply sided curved valley and is a good test of the ability of the algorithm to follow a non-linear equality constraint. An exponential inequality is added which crosses the valley axis at (-1.0, 1.0) and this is the optimum point at which the object function value is 4.0. The (infeasible) starting point used is (0.5, 0.5). It was found that SLA converged quickly and accurately provided the initial step length was less

than about 1.5. For larger step lengths feasibility was lost. Inspection of figure 6-1 which shows the contours of Rosenbrock's function reveals that with such large step lengths the linear approximation is hopelessly inadequate. The result of using large step lengths was that one or two moves were successfully completed leaving the search at a point where the linearized problem had no feasible solution. The linear programming routine produced an error message and SLA terminated. However table 6-4 shows that within the range of initial step lengths 0.05 to 1.00 SLA converged rapidly. The important conclusion to be drawn from this problem is that SLA can follow non-linear equality constraints successfully unless the step lengths are large. The definition of large obviously depends on the problem in hand and for a complex problem may not be apparent at the outset. However a loss of feasibility during the search is a positive indication of too large a step length which is quite straightforward to correct.

The third test problem used is due to D.A. Paviani [Ref. 21] and is thus referred to as the Paviani problem. The object function is quadratic in three variables, which are all constrained to positive values. In addition there is one spherical equality constraint and one linear equality constraint. Paviani gives two starting points of (2,2,2) and (10,10, 10). Unfortunately when the problem is linearized at either of these two points there is no feasible solution. Thus SLA is unable to start from either of these points. The problem was however put to good use. First, three starting points were selected from which a feasible solution was available. Six problems were then run, three with analytical derivatives and three with numerical derivatives. The results are given in table 6-16. That all six problems converged rapidly and accurately shows that, given a starting point at which a feasible solution to the linearized problem exists, problems with highly non-linear equality constraints may

TABLE 6-16

PAVIANI PROBLEM - RESULTS FROM THREE STARTING POINTS

Starting Point	Derivatives					
	Analytical			Numerical		
	Time	EFE	f	Time	EFE	f
(1.0, 1.0, 4.8)	0.63	107	961.715	0.70	133	961.715
(4.8, 1.2, 0.0)	0.75	125	961.715	0.71	155	961.715
(0.0, 1.8, 4.5)	0.95	154	961.715	0.92	191	961.715

Notes: 1. δx_i used for numerical derivatives was 10^{-7}

2. Initial step length was 0.5

3. Time is in C.P. seconds on CDC 6400

be solved. Also, the calculation time is virtually unaffected by the use of numerical derivatives as is the final accuracy of the solution. Although no systematic surveys of the other small test problems have been done using numerical derivatives numerous spot check calculations have confirmed that their use does not impair the efficiency of SLA. A second series of calculations was performed on the Paviani problem to determine the sensitivity of the solution time to increment and reduction factors as well as initial step lengths. The starting point (4.8, 1.2, 0.0) was used. Table 6-17 shows the results of 48 calculations. The three step lengths used were 0.05, 0.5 and 1.0. It was found that a step length of 5.0 was too great and, as in the Rosenbrock Ridge problem caused a loss of feasibility during the search. All the calculations shown in table 6-17 achieved the true minimum of 961.715. Tables 6-18 and 6-19 show these same results averaged over the three step lengths. The column averages indicate that a reduction factor of 0.1 to 0.2 is most suitable for this problem.

A fourth test problem referred to as Rosenbrock CC was used to test the reliability of SLA for equality constrained problems. The problem is the same as the Rosenbrock C problem discussed in Section 6-1-3 but with the circular inequality transformed to an equality constraint. The problem has two local minima marked L1 and L2 on figure 6-1 and a global minimum marked G. The local minimum L1 is the solution obtained for the Rosenbrock C problem from the starting point (-1.2, 1.0). Three calculations were performed starting at (-1.2, 1.0), (-0.5, 0.0) and (1.1, 0.6). From these three points the three minima L1, L2 and G were all located in less than 90 effective function evaluations. Table 6-20 gives the results in full. Since the average computation time for the Rosenbrock C problem was 1.05 seconds (FACRED=0.2, FACINC=2.1) as compared to the Rosenbrock CC result (FACRED=0.2, FACINC=2.0) of 0.21

TABLE 6-17

RESULTS OBTAINED ON PAVIANI PROBLEM

Increment Factor	Initial Step Length	Reduction Factor							
		0.1		0.2		0.3		0.4	
		Time	EFE	Time	EFE	Time	EFE	Time	EFE
1.7	1.0	0.90	170	0.56	118	0.91	186	1.78	348
	0.5	0.44	92	0.80	258	0.78	151	1.37	278
	0.05	0.92	163	1.10	198	0.85	160	1.58	288
1.9	1.0	0.72	138	0.44	97	0.75	149	1.00	198
	0.5	0.82	163	0.77	147	0.79	161	1.00	194
	0.05	1.08	187	0.88	157	1.17	215	1.13	208
2.1	1.0	0.71	140	0.55	110	0.55	121	0.91	183
	0.5	0.83	137	0.64	113	0.86	143	2.17	371
	0.05	1.53	238	0.88	139	1.22	210	1.26	211
2.3	1.0	0.75	143	0.61	126	1.15	228	0.98	198
	0.5	0.75	131	0.74	133	1.19	224	1.18	218
	0.05	1.01	179	1.32	215	1.23	235	1.45	255

TABLE 6-18

PAVIANI PROBLEM - AVERAGE NUMBER OF FUNCTION EVALUATIONS

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	142	191	166	305	201
1.9	163	134	175	200	168
2.1	172	121	158	255	177
2.3	151	158	229	224	191
Column Averages	157	151	182	246	

TABLE 6-19

PAVIANI PROBLEM - AVERAGE EXECUTION TIMES

Increment Factor	Reduction Factor				Row Averages
	0.1	0.2	0.3	0.4	
1.7	0.75	0.82	0.85	1.57	1.00
1.9	0.87	0.70	0.90	1.04	0.88
2.1	1.02	0.69	0.87	1.45	1.01
2.3	0.84	0.89	1.19	0.88	0.95
Column Averages	0.87	0.78	0.95	1.24	

Note: Times are C.P. seconds on CDC 6400.

TABLE 6-20

ROSENBROCK CC - SOLUTIONS FROM THREE STARTING POINTS

Starting Point	Optimum Found	Solution Vector	Solution Time	EFE
(-1.2, 1.0)	3.77029	(-0.94147, 0.88322)	0.21	66
(-0.5, 0.0)	0.400480	(0.39413, 0.13706)	0.32	88
(1.1, 0.6)	0.336724×10^{-2}	(0.94198, 0.88742)	0.29	77

Notes:
1. Initial step length used = 0.25
2. Increment factor = 2.0
3. Reduction factor = 0.2
4. Time is in C.P. seconds on CDC 6400

seconds it seems that, other things being equal, SLA performs more efficiently on equality constrained problems.

6-2 Conclusions from the Results of Small Test Problems

The most important conclusion to be drawn from the results obtained with these small test problems is that SLA is capable of solving problems with highly non-linear constraints, both inequality and equality. This gives confidence in its potential to solve larger and more complex problems. From the results of the Rosenbrock C calculations it is clear that a reduction factor of 0.4 is too large particularly when coupled with a large initial step length. Two cases of premature convergence with the same problem indicate that an increment factor of 1.7 is in some cases too small. This leaves a range of acceptable reduction factors of 0.1 to 0.3 and a range of acceptable increment factors from 1.9 to 2.3. The best combination to use depends on the particular problem and initial step lengths used.

A combination of 0.2 for reduction factor and 2.0 for increment factor has been used for the large problems described in the next section. This choice is made to standardize the results so obtained and does not imply that better results could not be obtained with different values. For consistency the results shown in table 6-8 for the Post Office Box A problem and in table 6-15 for the Rosenbrock C problem are average values for reduction factors of 0.2 and increment factors of 2.1.

The choice of a suitable initial step length for any given problem depends upon the complexity of the object function and the relative scaling of the independent variables. It is difficult to give a prescription for a 'best value' since in some cases the user will know little about the sensitivity of the object function to changes in the independent variables or the degree of non-linearity inherent in the constraints. However the

difficulties associated with step length choice are not confined to SLA. Most techniques involving uni-directional searches require an initial step length to begin the search. Simplex direct search techniques require an initial simplex which must be scaled. However this problem turns out to be not as formidable as might be thought since in general a range of initial step lengths spanning two orders of magnitude are acceptable to SLA. For most problems a step length of between 10 and 50% of the total change expected in any variable may be used. An estimate of this nature will usually be sufficiently accurate. For problems involving non-linear equality constraints the results obtained from the Rosenbrock Ridge and Paviani problems suggest it is better to choose an initial step length which is too small rather than one which is too large.

All the problems described in the previous section have had a convergence tolerance of 0.0001 on the elements of the solution vector. However convergence in most cases was obtained by zero length pattern moves (*i.e.* mode (2)) or by unchanging best points (*i.e.* mode (3)). This does not necessarily imply that the elements of the solution vector were not converged to this tolerance. For most problems they were. It does however mean that savings in computational effort caused by slackening the convergence criteria will not be great unless the change is considerable, say to 0.01.

Notwithstanding the difficulties of initial step length choice the algorithm is able to obtain tightly converged solutions to a variety of non-linear problems. It has proved reliable and competitive with other techniques in terms of effective function evaluations. For tightly constrained problems where the number of active constraints is equal to the number of independent variables convergence is rapid.

The next section describes the results obtained with SLA when applied to larger and more complex problems.

6-3 Results for Large Test Problems

Ten problems have been selected to test the capability of SLA to solve larger and more complex problems than those described in Section 6-1. The problems originate from various sources but all may be found in Himmelblau's book [Ref. 22]. Appendix 1 contains the detailed specification of these problems together with the standard starting points and solutions. Table 6-21 shows the general characteristics of the problems. The number of independent variables ranges between 3 and 24, and the number of constraints, including bounds, from 13 to 48. The presentation of the problems varies considerably. Some, like problems 2 and 4, are straightforward analytical functions which present few problems for codes requiring analytical first or even second derivatives. Others, like problems 6 and 7, have object functions and constraints which can be manipulated analytically only with extreme difficulty, if indeed at all. In all cases numerical derivatives have been used since this is the easiest method of problem preparation. As described in Section 5-3 a forward difference formula is used to estimate first derivatives of both object function and constraints. A perturbation of 10^{-7} has been used throughout. Values of 0.2 and 2.0 have been used for reduction and increment factors and the convergence criteria have been set to 0.001 in all calculations.

The initial step lengths chosen vary from problem to problem. In most cases the same value has been used for all the independent variables. Obviously the author has prior knowledge of the solution for each problem and this enables a suitable step length to be chosen. However in most practical problem-solving situations the user will be familiar with the relative sensitivities of the independent variables, and will have some idea of how much he expects them to change. This insight into the problem structure should allow step lengths to be chosen which are not grossly

TABLE 6-21

CHARACTERISTICS OF LARGE TEST PROBLEMS

Problem Number	Inequalities		Equalities		Number of Bounds	Number of Variables	Number of Active Constraints
	Linear	Non-linear	Linear	Non-linear			
1		3			10	5	5
2		5			15	15	11
3	10				5	5	4
4		6			10	5	5
5		12			2	9	7
6		14			6	3	2
7	4	34			10	5	6
8			3		10	10	3
9			8		32	16	13
10		6	2	12	24	24	40

under- or over-sized. In general step lengths have been chosen which are between 10 and 50% of the total change expected. It has already been shown on the smaller test problems that variations of two orders of magnitude often cause little change in total computation time required for solution. No attempt has been made to choose the best initial step lengths. Without doubt the computation time for many of the problems could be reduced by a different choice of step lengths. By not adopting this approach the author hopes that the results obtained may be regarded as typical of what a non-specialist user with some knowledge of his own particular problem, could obtain.

All the problems used were compiled and executed in less than 25000 words of central memory. The results obtained with SLA are compared with those given by Himmelblau [Ref. 22] for seven other algorithms. The comparison, given in table 6-22, is in terms of standardized central processor time. This means that the actual execution time is divided by the time to execute Colville's standard timing program [Ref. 16]. Himmelblau quotes the execution time for the timing program as 22.0 seconds on his machine (CDC 6600). As previously noted the timing program required 43 seconds on the Imperial College CDC 6400 machine. Each of the ten test problems is discussed below.

Problem 1

This problem was used by Box [Ref. 13] to introduce his Complex algorithm. Evaluation of the object function involves substitution into a cascade of equations to evaluate seven constants. These constants are then substituted into the object function formula along with the five independent variables. The evaluation of the three non-linear constraints also involves the determination of the same seven constants. The problem is easily solved by SLA since at the solution point there are as many active constraints as independent variables. Himmelblau gives only two

comparative times (table 6-22). SLA is more than ten times as fast as GRG and almost 500 times as fast as the Flexible Tolerance method of Paviani [Ref. 14]. Himmelblau does not indicate why the other five codes were not applied to this problem. The initial step length used for all five variables was 10.0. The maximum located by SLA was slightly greater than that given by Himmelblau.

Problem 2

This problem is more complex than the previous one having fifteen variables and five non-linear inequality constraints. There are 11 active constraints at the solution point. Table 6-22 shows that for both starting points (one feasible, one infeasible) the SUMT penalty function technique is faster by a factor of about three. The code GRG is also three times faster from the feasible starting point. The SLA calculations are probably converged too tightly since, for the feasible starting point calculation, the final solution of -32.3548 had been reached by the 36^{th} iteration but convergence was not achieved until the 49^{th}. The converged object function value of -32.386 given by Himmelblau was in fact passed by the 25^{th} iteration. The calculation beginning at the infeasible starting point produced an object function value of -32.3487 in 70 iterations. The converged object function value of Himmelblau was passed by iteration number 43. Table 6-22 shows that SLA performs significantly better than NLP which is a similar technique based on successive linear approximations.

In Chapter 4 it was noted that the early work on SLA was done using the Griffith and Stewart split variable technique. It is interesting to compare the time per iteration obtained for problem 2 using the original formulation with that obtained using the displaced origin technique. In fact three cases will be compared:

(1) Split variables, bounds treated as inequality constraints.

(2) Split variables, bounds set by step length limits.

(3) Displaced origin, bounds set by step length limits (*i.e.* the current SLA method).

The standardized times per iteration for these three methods are 0.118, 0.074 and 0.015 respectively. Thus the iterations are speeded up, on average, by a factor of about 1.6 by using the step length limitations to impose the 15 bounds of problem 2. A further factor of five in speed is gained by using the displaced origin technique thus enabling problem 2 to be treated as a 15 variable problem instead of a 30 variable problem. The overall gain is about a factor of eight in speed which demonstrates the considerable advantage gained by the use of the displaced origin method. As well as a reduction in execution time there is also a reduction in computer core requirements. The size of the matrix a_{ij} of equations 4-2 in Section 4-3 being 1500, 1050 and 300 elements for the three cases considered above. For large problems reductions in total core requirements are substantial.

Problem 3

This is a relatively easy problem for SLA since all ten inequalities are linear. Convergence takes 16 iterations and at the solution point there are four active inequality constraints. Table 6-22 shows that SLA, GRG and NLP are require about 0.07 seconds for convergence. The SUMT algorithm is a little slower and the Flexible Tolerance algorithm considerably slower. Like problem 2, which is in fact the dual of problem 3, the object function and constraints are simple analytical functions which are easily differentiated for use in codes which require analytical derivatives. The initial step length used for all variables was 0.2.

Problem 4

This is a five variable problem having six non-linear inequality constraints. Two starting points are used, one feasible and one infeasible. At the solution point there are five active constraints. For both starting points SLA converges more quickly than any of the six codes used by Himmelblau. The initial step length used for all variables was 0.2. Since this problem has as many active constraints as variables, convergence does not depend on step reduction following cubic fitting. A larger step length, which would encompass the solution point from the start of the calculation, would have produced a more rapid solution.

Problem 5

This nine variable problem produced some interesting results. From the starting point given by Himmelblau (*i.e.* $x_i = 1.0, \quad i = 1,9$) SLA converged to a local maximum of 0.67498 after 59 iterations. This is inferior to the result given by Himmelblau of 0.8660. The initial suspicion was that an error had been made in the coding of the 13 non-linear inequality constraints. To test this SLA was given a starting point close to Himmelblau's solution. The code converged in 17 iterations to the exact solution given by Himmelblau. Thus the problem appeared to have been correctly coded and to possess two maximum values. A further calculation was performed starting from $x_i = 0.0, \quad i = 1,9$. This converged to the maximum value given by Himmelblau, but for an entirely different solution vector. Thus there are at least three constrained maxima for problem 5. All three may be local maxima with the global maximum not yet located. Alternatively there may be two (or more?) co-equal global maxima with one or more local maximum values. The execution time given in table 6-22 is for the zero (*i.e.* non-standard) starting vector. The time in parentheses is the time required to reach the local maximum value from the standard starting vector. Problem 5 is clearly a diffi-

TABLE 6-22

STANDARDIZED TIMES REQUIRED TO SOLVE LARGE TEST PROBLEMS

Problem Number	GRG	Flexible Tolerance	NLP	SUMT	POPII	Rosenbrock	GGS	SLA
1	0.102	3.277	N.A.	N.A.	N.A.	N.A.	N.A.	0.007
2	0.245 (N.A.)	N.S. (N.S.)	4.155 (3.823)	0.254 (0.305)	N.S. (N.S.)	N.S. (N.S.)	N.S. (N.S.)	0.724 (1.008)
3	0.070	0.345	0.074	0.127	N.S.	N.S.	N.S.	0.073
4	0.162 (0.296)	0.121 (0.632)	0.105 (0.029)	0.048 (0.135)	N.S. N.S.	0.078 (0.136)	0.105 (N.S.)	0.030 (0.017)
5	N.S.	2.709	N.S.	N.A.	N.S.	N.S.	N.S.	0.266 (0.593)*
6	0.174	0.192	0.073	N.S.	0.078	0.002	N.A.	0.062
7	N.A.	4.695	0.222	N.S.	N.S.	N.S.	N.A.	0.195
8	0.066	1.268	0.244	0.057	-	-	0.052	0.219
9	0.133	6.455	2.518	0.363	-	-	0.594	0.735
10	0.226	23.227	N.S.	-	-	-	N.S.	0.502

Notes: 1. N.S. = No solution found. N.A. = Problem not attempted. - = Algorithm cannot be applied.

2. Figures in parentheses are for alternative infeasible starting point.

3. Standardization factor for SLA (CDC 6400) = 43.

4. * , see Section 6-3.

cult problem since Himmelblau reports that five of the six algorithms applied to this problem failed. Only the Flexible Tolerance code succeeded in solving the problem, and that required considerably more time than SLA. The initial step length used for all variables was 1.0.

Problem 6

This problem has only three variables but there are 14 non-linear inequality constraints. An added complication is that the object function and constraints are defined by a FORTRAN subroutine. This makes the evaluation of analytical derivatives very difficult and time consuming. Surprisingly, at the solution point only one of the non-linear constraints is active. The upper bound on x_2 is also active at the solution. The codes SLA, POP II and NLP recorded very similar times and are faster than GRG and Flexible Tolerance by a factor of two. The direct search method of Rosenbrock performs extremely well on this problem being about 30 times as fast as the three linear approximation codes. In view of this code's poor performance on other problems it is likely that an optimal point was located more by chance than by an efficient search procedure. It would have been inappropriate to use the same initial step length for all three variables in this problem since the total movement required for the second variable is several hundred times the movement of the other two. Accordingly, step lengths of 2.0 were assigned to the first and third variables, and a step length of 400.0 was chosen for the second.

Problem 7

This problem has a large number of inequality constraints. There are upper and lower bounds on each of the five independent variables as well as 34 non-linear inequalities and four linear inequalities. As in problem 1 the independent variables are used to calculate a whole series of intermediate quantities by successive substitutions in a cascade of

34 equations. The intermediate quantities are used in conjunction with the independent variables to define both object function and constraints. Since the functional form of the object function and some of the constraints is so complex, techniques requiring analytical derivatives are at a distinct disadvantage. For instance, Himmelblau states that the penalty function code SUMT, '... almost certainly failed because of undiscovered errors in one or more of the 315 second partial derivatives.' The code SLA performs well on this problem converging in eight iterations. The rapid solution is again due to the number of active constraints being at least equal to the number of independent variables. The time taken to converge is shown in table 6-22 to be a little less than the NLP algorithm and considerably less than the Flexible Tolerance method. The step length used for all five variables was 20.0. This is sufficient to encompass the solution values of all variables except the first. Convergence is thus delayed whilst the first variable moves in steps of 20 from its initial value of 900 to the solution value of 705.17. The step length is increased twice during the calculation by the step increment factor (2 in this case) so progress is not too slow. A second calculation was performed on this problem, this time setting the initial step length of the first variable to 200.0, the others remaining at 20.0. This reduced the number of iterations required for convergence to five and approximately halved the execution time shown in table 6-22.

Problem 8

This is a ten variable problem with three linear equality constraints. Its most awkward feature is that the object function contains terms involving the natural logarithms of the independent variables. If any of the variables is zero an 'infinite' value is generated by the computer and the program fails. There is no difficulty when using SLA since by setting the lower bound for each variable to a very small value (10^{-8} was used)

it is possible to ensure that the search never obtains values closer to zero. The code NLP could only solve the problem by using a variable transformation of the form:

$$\overline{x}_i = ln(x_i)$$

This avoids the possibility of the program generating an infinite value but it also makes the equality constraints non-linear and hence much more difficult to deal with. Table 6-22 shows SLA to be less competitive on this problem. As before Flexible Tolerance is slow to converge. This problem is more difficult than some of the others for SLA since only three constraints are active at the solution. This means that convergence is by oscillation, cubic fitting and step reduction. The step length used was 0.2 for all variables.

Problem 9

This problem has 16 independent variables and is thus larger than any so far described. Analytical derivatives are easily obtained since the object function has a simple symmetric form and the eight equality constraints are linear. Thirty-one iterations were required for solution by SLA. At the optimum point there were five active lower bounds making, with the equality constraints, thirteen active constraints in total. All the equality constraints were satisfied to 10^{-11} or better. The code GRG performs well on this problem being faster by a factor of three than the next best code, SUMT. The codes SLA and GGS required about the same amount of time whilst NLP and Flexible Tolerance were considerably slower. An initial step length of 5.0 was used for all variables.

Problem 10

This is the largest and most complex problem described in this section. It has 24 variables, 12 non-linear equality constraints, 2 linear equality constraints, 6 non-linear inequality constraints and 24 lower bounds. The object function is linear. The principal difficulty with this problem as

far as SLA is concerned lies in the six non-linear inequality constraints. It is seen from the problem specification in Appendix 1 that the constants, e_i, in the inequalities are all positive. Since all the independent variables are constrained to be greater than or equal to zero the six inequalities can only be satisfied if the variables x_4, x_5, x_6, x_{16}, x_{17} and x_{18} are exactly zero. If they are slightly less than zero then the non-negativity condition is violated. If they are slightly greater than zero then one or more of the six inequalities will be violated. In fact the problem is badly posed and is in reality only an 18 variable problem. An attempt was made to solve the problem as posed, with initial values of 0.04 for all variables. This failed when the linear programming routine could find no feasible solution. A second attempt at solution was made this time allowing a tolerance of 10^{-7} on the inequality constraints, thus expanding the feasible region. Again the linear programming routine could find no feasible solution. A third attempt at solution was made this time setting the six critical variables (x_4, x_5, x_6, x_{16}, x_{17} and x_{18}) to zero. The first iteration was successfully completed but in the second iteration once again the linear programming routine could find no feasible region. A detailed investigation showed that the cause lay in the small rounding errors incurred during the linearization of the inequality constraints. The very small errors so introduced caused the feasible region for the six key variables to disappear. Consequently a tolerance of 10^{-7} was reintroduced into the inequalities and a fourth calculation performed. This calculation was successful. Setting six variables to zero enabled an initial feasible solution to be found and adding a tolerance of 10^{-7} to the inequalities prevented subsequent loss of feasibility. Convergence required only six iterations. The reason for such rapid convergence is the 40 active constraints at the solution point. The convergence time shown in table 6-22

is, strictly speaking, not directly comparable with the results of the GRG and Flexible Tolerance algorithms, since the calculation did not begin at the standard starting point. However the code SLA is clearly greatly superior to the Flexible Tolerance code for this type of problem. It is also worth noting that the solution found by SLA was nearly 10% lower than that located by Flexible Tolerance. This lower object function value was not achieved by constraint violation since the inequalities were satisfied to 10^{-7} or better and the equalities to 10^{-8} or better.

It was noted earlier that problem 10 had been poorly posed. By removing the six variables which were necessarily zero the problem was reformulated as an 18 variable problem having 11 equality constraints and 18 lower bounds. This modified problem was solved by SLA again in six iterations. However the execution time was less than half that of the original problem. The solutions obtained by SLA for the 24 and 18 variable problems were the same to at least six decimal places. The halving of the execution time by reformulating the original problem emphasizes the advantages gained by careful problem preparation.

6-4 Discussion of the Results of Large Test Problems

The correct and accurate solutions obtained by SLA for the ten test problems have established the reliability of the algorithm. The only difficulties experienced were with problem 10 and, as explained in the preceding section, these were caused primarily by poor problem formulation. Nine of the ten test problems, when linearized from the given starting point, had feasible regions. This is an indication that the requirement of SLA for a feasible region should not be a significant drawback. Table 6-22 shows that only SLA succeeded in solving all the test problems.

The standardized times in table 6-22 show that in four out of the ten problems SLA required the least computer time. Only GRG, Flexible

Tolerance and NLP have reliability comparable to SLA and, of these three, both NLP and Flexible Tolerance are consistently slower. GRG is the fastest algorithm for four of the ten problems. However GRG requires analytical derivatives and so problem preparation in some cases would be both long and tedious. GRG failed to obtain a solution for problem 5. It should be noted that since no effort was made to find the optimum initial step lengths the results in table 6-22 are not necessarily the best which could be obtained with SLA.

In most cases less calculation time is required to evaluate derivatives analytically, since straightforward substitution in formulae is computationally simpler than a series of subroutine calls to evaluate the object function and constraints numerically. Thus, had analytical derivatives been used in SLA the times shown in table 6-22 would probably have been less.

It is difficult to assess the relative accuracies of the results obtained by the algorithms shown in table 6-22, since the author does not have access to the detailed results. However as far as SLA is concerned all the inequalities are met to within 10^{-6}. The upper and lower bounds on the variables are never violated. The equality constraints are certainly met to 10^{-6} and are usually 10^{-8} or better. Moreover the optimal object function produced by SLA is in all cases as good as that quoted by Himmelblau and in several cases better.

An important feature of the code SLA is its ability to utilize numerical derivatives. This of course greatly simplifies the preparation required to solve most problems. More importantly, for problems involving complex object functions and/or constraints it may be impossible to express the derivatives analytically. In such cases only direct search techniques, such as those due to Rosenbrock and Box, and techniques capable of using derivatives generated numerically may be applied. Preparation

times for all ten problems, excluding card punching, were typically about half an hour. Half of the problems ran correctly at the first attempt. The rest required one or two additional runs, mainly to correct punching errors in the data.

CHAPTER 7

APPLICATION OF SLA TO RADIATION SHIELD OPTIMIZATION

In the early days of the exploitation of nuclear power the radiation shielding provided for operating personnel was very conservatively designed. This was necessary since calculational techniques for radiation transport predictions were relatively unsophisticated and the basic data required for such computations were either unavailable or of limited accuracy. The shield designer today is equipped with an array of powerful computational techniques backed up by extensive libraries of nuclear data. He has access to large, powerful computers and, what is more important, has over twenty years of accumulated design experience. It is now no longer ncessary to design radiation shielding with enormous safety factors. In fact, since the cost of a shield is a significant proportion of the total cost of a nuclear installation, there is a great financial incentive to produce not merely an adequate design but an optimal design.

The prime requirement of any radiation shield is that it attenuates radiation to such an extent that workers are not exposed to radiation doses above internationally prescribed limits. This has been achieved by restricting access to high radiation areas and also by imposing area dose rate limitations on the shield design such that prolonged occupancy of some areas is possible without incurring an excessive radiation dose. This use of area dose rate limitations which permit high occupancy factors for all workers will usually require the provision of shielding which is essentially superfluous. In most situations workers will spend their working days moving between several work-stations, some of which will be far removed from the radiation source. Therefore the assumption that a worker spends the whole of his working day close to the radiation source is both conservative and costly. In order to reduce these costs it is

necessary to examine the movements of individual workers and tailor the shield system to them. The limitations on the shield design thus become the integrated doses of the individual workers and not the area dose rates.

In the work described here it is shown possible to optimize a radiation shield taking into account the integrated doses acquired by individual workers whilst following a particular work schedule. Results are given for a 25 element shield system within and around which 25 workers spend various parts of their working day at any one of 25 reference dose points. The effects on the optimal shield design of imposing area dose rate constraints of varying magnitudes in addition to the integrated dose limitations on workers are examined.

7-1 Shield Optimization

There are many parameters which may be varied in order to achieve an optimal radiation shield design. However, because of the computational complexities involved, it is not usually feasible to optimize a shield by changing all the available parameters simultaneously. The optimization of a total radiation shield is thus usually achieved by solving a series of simpler sub-optimization problems. The main design parameters which are at the disposal of the shield engineer are:

(1) The number of shields and their geometrical form.

(2) The material composition of the shields.

(3) The ordering of material layers within a shield.

By changing some or all of these parameters the designer seeks, in qualitative terms, to produce a 'best' or 'most desirable' shield.

In an ideal case the designer would decide on all the attributes of the radiation shield of interest to him (*e.g.* its cost, weight, integrity) and would then draw up a series of 'utility functions', one for each attri-

bute. These utility functions would express, in the form of a continuous curve, the variation of the designer's preference for each attribute. They would all be to a common scale. It would then be possible to optimize the shield design by maximizing the sum of the utilities of the individual attributes. The qualitative behaviour of these utility functions is clear. A low cost shield has a high utility whilst a high cost shield has low utility. A light shield is preferable to a heavy one, and so on. However the precise shape of the utility function is not unique. Each designer will have his own relative scale of values. The most difficult problems arise when attempts are made to set a common scale to several utility functions. It is exceedingly difficult to decide on the relative values or trade-offs to be made between cost, weight, size, dose rate, reliability, ease of construction and so forth.

In practice a much simpler approach is adopted. The most important or 'prime' attribute is chosen to be the object function and other attributes of concern are included as constraints. For example, in the case of a mobile reactor the prime attribute might be the total shield weight. Secondary attributes would be the total cost and, say, the average dose acquired by each radiation worker. Thus the problem would be formulated as a minimum weight problem subject to maximum cost and dose constraints. Prime attributes which could be used as object functions for constrained optimization include the cost, the weight or the volume of the shield. Secondary attributes which might be used as constraints are:

Dose rates at points within the shield structure.

Dimensions of shield elements.

Material compatibility.

Rate of heat production within the shield.

Material densities.

Thus, there are many ways of formulating a 'shield optimization problem'.

However, whichever formulation is used the problem of calculating the attenuation properties of the radiation shield for a specified source remains. For problems of practical importance the effort involved in calculating the radiation attenuation is considerable. Consequently, it is not possible to use a shield model which requires a complete recalculation of attenuation properties to be performed each time a perturbation is made in one of the design variables. This would require vast amounts of computing effort for all but the very simplest problems. An alternative approach is to use a model in which changes in shield attenuation due to variations in the design variables are expressed as simple functions. These functions are fitted empirically to either experimental results or a detailed reference calculation and will therefore have a limited range of validity. However, by using such functions, changes in the shield attenuation properties due to perturbations from a reference design are easily calculated. The use of such a simplified shield model makes the use of mathematical programming techniques computationally feasible. Should the result of an optimization calculation involve unacceptable extrapolations from the reference design it may be necessary to repeat experiment or calculation and establish new local approximations. The optimization calculation may then be repeated within the region of valid extrapolation.

7-2 The Approach Adopted

A shield model based on perturbations from a reference design has been used. The shield is modelled by splitting it up into N sections each of area A_i, $i = 1,N$. The average dose rate over the surface of each section is known as the tolerance, T_i. These tolerances are assumed to vary exponentially with the thickness of the shield section:

$$T_i = R_i e^{-K_i t_i} \qquad (7\text{-}1)$$

- where R_i is the reference tolerance.

  t_i is the shield thickness perturbation.

  K_i is a pseudo-attenuation coefficient suitably chosen to allow for obliquity effects.

The dose rate, D_r, at any point is expressed as a sum of contributions from each element:

$$D_r = \sum_{i=1}^{N} C_{ri} T_i \tag{7-2}$$

- where C_{ri} is the fraction of the i^{th} tolerance reaching the r^{th} close point.

These dose rates are then used to calculate the integrated doses acquired by workers within the shield structure. The integrated dose of the m^{th} man is defined as:

$$I_m = \sum_{r=1}^{Q} F_{rm} D_r \tag{7-3}$$

- where Q is the total number of dose points.

  F_{rm} is the time spent by man m at dose point r.

Substituting (7-2) into (7-3) we have:

$$I_m = \sum_{r=1}^{Q} F_{rm} \sum_{i=1}^{N} C_{ri} T_i \tag{7-4}$$

The objective function used is that of shield weight and is defined as:

$$U = \sum_{i=1}^{N} A_i t_i \rho_i \tag{7-5}$$

- where ρ_i is the density of the i^{th} shield section.

Substituting for t_i in equation 7-5 we have:

$$U = \sum_{i=1}^{N} \frac{A_i \rho_i}{K_i} \left\{ ln \frac{R_i}{T_i} \right\} \tag{7-6}$$

Two sets of inequality constraints are applied to the object function defined in equation 7-6:

(1) Constraints limiting the maximum integrated dose acquired by each worker.

$$I_m \leqslant \text{Maximum Dose} \tag{7-7}$$

(2) Constraints limiting the maximum dose rate at each dose point.

$$D_r \leqslant \text{Maximum Dose Rate} \tag{7-8}$$

Both these sets of constraints are linear. Note that the independent variables of the model are the tolerances, T_i, not the shield thicknesses, t_i.

This model was originally developed by Vickers Limited (Barrow-in-Furness) [Ref. 23] in order to optimize reactor shielding for nuclear powered submarines. The optimum set of shield thicknesses was calculated using the Lagrange method of undetermined multipliers. However this method of solution enabled the integrated dose of only one man at a time to be considered.

By using SLA to minimize the object function given in equation 7-6 subject to the inequalities shown in equations 7-7 and 7-8, minimum weight configurations have been obtained in which integrated dose limitations on all workers are satisfied. Results are presented in the following section of calculations involving a 25 element shield.

7-3 Shield Optimization Results Obtained with SLA

A set of data was prepared for a hypothetical shield system having 25 shield elements. A work schedule or 'Time Allocation Matrix' was drawn up for 25 men working at 25 dose points. This is shown in table 7-1 where the percentage of time spent by each of the 25 men at each of the 25 dose points is shown. Also a Contribution Matrix was prepared which showed the fraction of the tolerance at each shield section which is contributed to the total dose at each dose point. This is shown in table 7-2. These two matrices are the F_{rm} and C_{ri} shown in equations 7-3 and 7-2 respectively. The figures in table 7-1 are percentages and must be multiplied by the number of hours in a working week and divided by 100 to obtain the actual time worked in hours. Other data used in the calculation are shown in table 7-3.

A series of calculations was performed in which a limit of 140 mR was set on the maximum integrated dose to be absorbed by any man in a 40 hour week. The dose rate limit at each dose point was set at 0.5 mR/hr and increased by 1 mR/hr for each successive calculation. The resultant shield weight reduction is shown in arbitrary units in figure 7-1. The decrease in shield weight due to the relaxation of the area dose rate constraints is initially large but as the dose rate limit approaches 3.5 mR/hr weight reductions become progressively smaller. By this stage the design is entirely determined by the maximum integrated dose limitations. This change from a design limited by area dose rate constraints to one limited by the integrated doses acquired by the workers is illustrated in figure 7-2. When the area dose rate limit is set at 0.5 mR/hr there are 12 limiting dose points. That is, there are 12 dose points which have a dose rate of exactly 0.5 mR/hr. When the area dose rate limits are relaxed to 5.5 mR/hr no dose points are limiting but five of the men acquire an integrated dose of 140 mR and thus limit

TABLE 7-1 TIME ALLOCATION MATRIX - PERCENTAGE OF TIME SPENT BY 25 MEN AT EACH OF 25 DOSE POINTS

MEN	DOSE POINTS																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	3	0	7	0	15	0	5	4	0	30	0	0	2	0	0	6	0	8	0	10	0	10	0	0
2	2	2	5	0	10	0	7	0	0	8	0	18	0	0	20	0	0	10	8	0	5	0	0	5	0
3	0	0	0	0	0	0	0	0	8	0	7	0	23	0	0	23	0	0	0	14	0	5	0	5	15
4	10	0	15	10	7	0	0	15	0	0	0	0	0	15	0	0	20	0	0	0	0	8	0	0	0
5	7	0	3	0	0	15	10	0	5	0	10	0	0	8	0	0	0	0	30	0	0	2	0	6	0
6	0	7	0	2	5	8	0	0	0	2	0	0	0	0	5	0	0	5	0	0	18	10	20	8	10
7	8	0	10	0	0	0	15	20	0	0	0	10	0	0	0	7	5	0	0	15	0	0	0	0	0
8	0	18	0	0	10	0	0	0	7	15	0	0	15	8	0	0	0	19	0	0	0	0	0	7	0
9	5	0	0	15	0	6	0	0	0	0	14	0	8	0	20	0	0	0	20	0	12	0	0	0	0
10	0	13	0	0	0	0	13	0	14	0	0	18	20	11	9	2	0	0	0	0	0	0	0	0	0
11	0	0	18	0	0	14	0	20	0	0	13	0	0	11	0	0	13	0	0	0	0	9	0	2	0
12	7	0	0	3	0	0	15	0	10	6	0	0	2	0	0	0	0	0	0	5	10	8	0	0	30
13	8	7	23	0	0	0	0	0	0	0	0	0	0	0	0	15	5	5	14	0	0	0	0	0	23
14	10	0	0	0	15	0	10	7	0	15	0	15	0	20	0	0	0	0	0	0	0	0	8	0	0
15	0	0	18	0	0	0	0	0	0	0	16	0	7	0	0	10	0	0	15	0	6	19	0	7	0
16	0	0	0	13	0	20	0	0	18	0	0	0	0	14	0	0	11	13	0	9	0	0	2	0	0
17	7	30	0	0	3	0	6	10	0	8	15	0	10	0	0	5	0	0	0	0	0	0	0	0	2
18	0	0	15	5	0	0	0	0	0	0	0	8	0	0	23	0	0	7	0	0	5	0	23	14	0
19	0	12	0	0	18	0	15	0	5	0	0	0	22	0	0	16	0	0	12	0	0	0	0	0	0
20	0	0	13	0	0	20	0	0	0	18	0	0	0	0	0	0	13	11	0	14	0	9	0	0	2
21	10	0	0	15	0	0	0	10	0	0	8	7	0	15	15	0	0	0	0	0	20	0	0	0	0
22	18	10	0	0	0	7	0	0	16	0	0	0	15	0	0	8	0	0	19	0	0	0	7	22	0
23	0	0	12	0	18	0	15	0	0	12	0	0	0	0	0	0	0	16	0	0	0	5	0	0	0
24	0	7	0	7	0	7	0	20	0	0	9	0	0	20	0	0	25	0	0	5	0	0	0	0	0
25	12	0	18	0	0	12	0	16	0	0	0	5	0	0	0	0	0	0	15	0	0	0	0	0	22

TABLE 7-2 CONTRIBUTION MATRIX - FRACTION OF TOLERANCE CONTRIBUTED TO EACH DOSE POINT

SHIELD	DOSE POINTS																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	.020	0	0	.041	.010	0	.100	0	0	0	.500	0	0	0	.250	0	0	0	.010	0	0	.010	0	.060	0
2	.020	0	0	.085	0	0	.120	0	0	0	.200	0	0	.100	0	0	.005	0	0	0	.330	0	0	0	0
3	0	.010	0	0	.200	0	0	0	.015	0	0	0	.070	0	0	.250	0	0	0	0	0	0	.080	0	0
4	0	0	.050	0	0	.500	0	0	0	.100	0	0	0	.002	0	0	0	.005	.010	0	0	.001	0	0	0
5	.100	0	0	0	0	0	0	.800	0	0	.008	0	0	0	0	0	0	0	0	.050	0	0	0	0	.200
6	0	.010	.006	0	.025	0	0	0	.300	0	0	0	0	0	0	0	.700	0	0	0	0	.020	0	0	0
7	.005	0	0	.020	0	.010	0	0	0	0	0	.750	0	0	.070	0	0	.080	0	0	0	0	0	.300	0
8	.020	.040	.010	0	.100	0	0	0	0	0	0	0	.500	.250	.010	0	0	0	0	.010	.060	0	0	0	0
9	0	.010	0	0	0	0	0	.200	0	0	.010	0	0	0	0	.070	0	0	.250	0	0	0	.080	0	0
10	0	0	.050	.200	0	0	0	0	0	0	0	0	0	0	0	0	0	.800	0	0	0	.100	0	0	.008
11	0	0	0	0	0	0	.005	.020	.010	.750	0	0	0	0	0	.300	0	0	0	.070	.080	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	.330	0	.005	0	0	.100	.200	.120	0	.085	0	.020	0	0
13	.010	0	.200	0	.015	0	0	0	.070	0	.250	0	.080	0	0	0	0	0	0	0	0	0	0	0	0
14	0	.100	0	0	0	0	.800	0	0	0	0	0	0	0	.008	0	0	0	0	.050	0	.200	0	0	0
15	.175	0	0	.120	0	0	0	.120	0	0	0	.500	0	.025	0	.060	0	0	.050	0	0	0	.050	0	0
16	0	.080	.080	0	0	.080	0	0	0	.080	0	0	0	0	.100	0	.100	0	.120	0	0	.180	0	.300	0
17	.130	0	0	.200	0	0	.200	0	.150	0	.070	0	.200	0	0	0	.025	0	0	.025	0	0	.210	0	0
18	0	.005	0	0	.035	0	.015	0	0	.120	0	0	0	0	.025	0	0	.300	0	0	.800	0	0	0	0
19	0	0	.175	0	0	.350	0	0	0	0	0	.020	.020	.020	.020	.020	0	0	.100	0	0	0	.175	0	0
20	0	.100	0	0	.200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	.050	0	0	0	0	.450	0	0	.120	0	0	.050	0	0	.050	0	0	.050	0	0	.150	0	0
22	.065	0	0	0	.220	0	0	0	.065	0	0	0	.075	0	0	.170	0	0	0	0	.075	0	0	0	0
23	0	0	0	.500	0	0	0	.020	0	0	0	.020	0	0	.140	0	0	.020	0	0	0	0	0	.120	0
24	.140	.015	.015	0	.090	.100	.160	.010	.008	.050	.010	.010	.010	.010	.015	.015	.200	.015	.015	.170	.010	.020	.030	.100	.100
25	.080	0	0	0	.160	0	0	0	.090	0	.600	0	0	.070	0	0	0	0	0	0	0	.050	0	0	0

TABLE 7-3

DATA USED IN 25 ELEMENT SHIELD CALCULATION

Area of each shield element.	= 1 m^2
Integrated dose limit for each man.	= 140 mR
Number of working hours per week.	= 40
Pseudo-attenuation coefficient for each shield element.	= 0.5 cm^{-1}
Reference tolerance at each shield element surface.	= 1.0 mR/hr
Convergence criterion on tolerances.	= 0.05 mR/hr

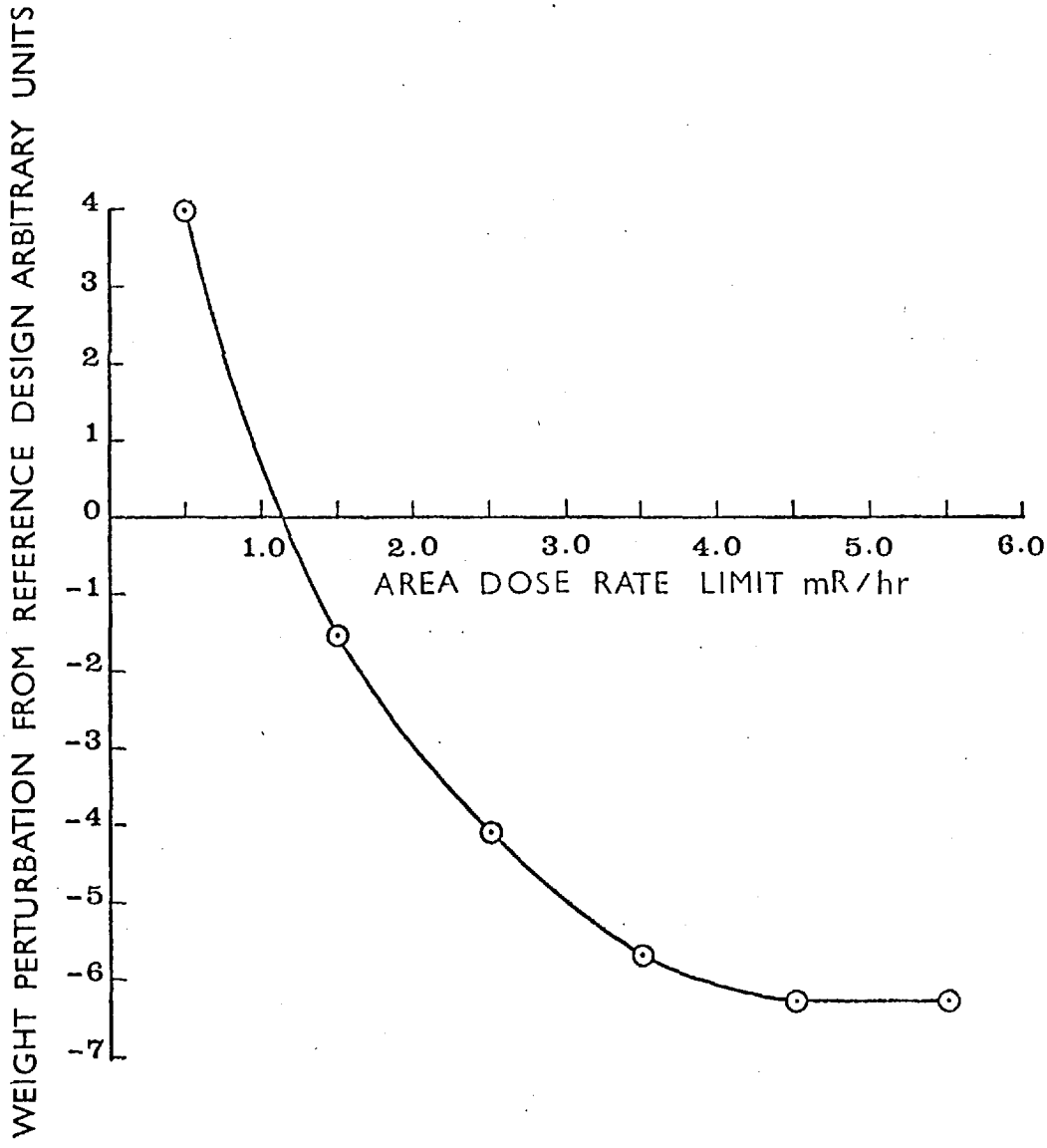

FIG. 7-1 Shield Weight Perturbation as a Function of Area Dose Rate Limit.

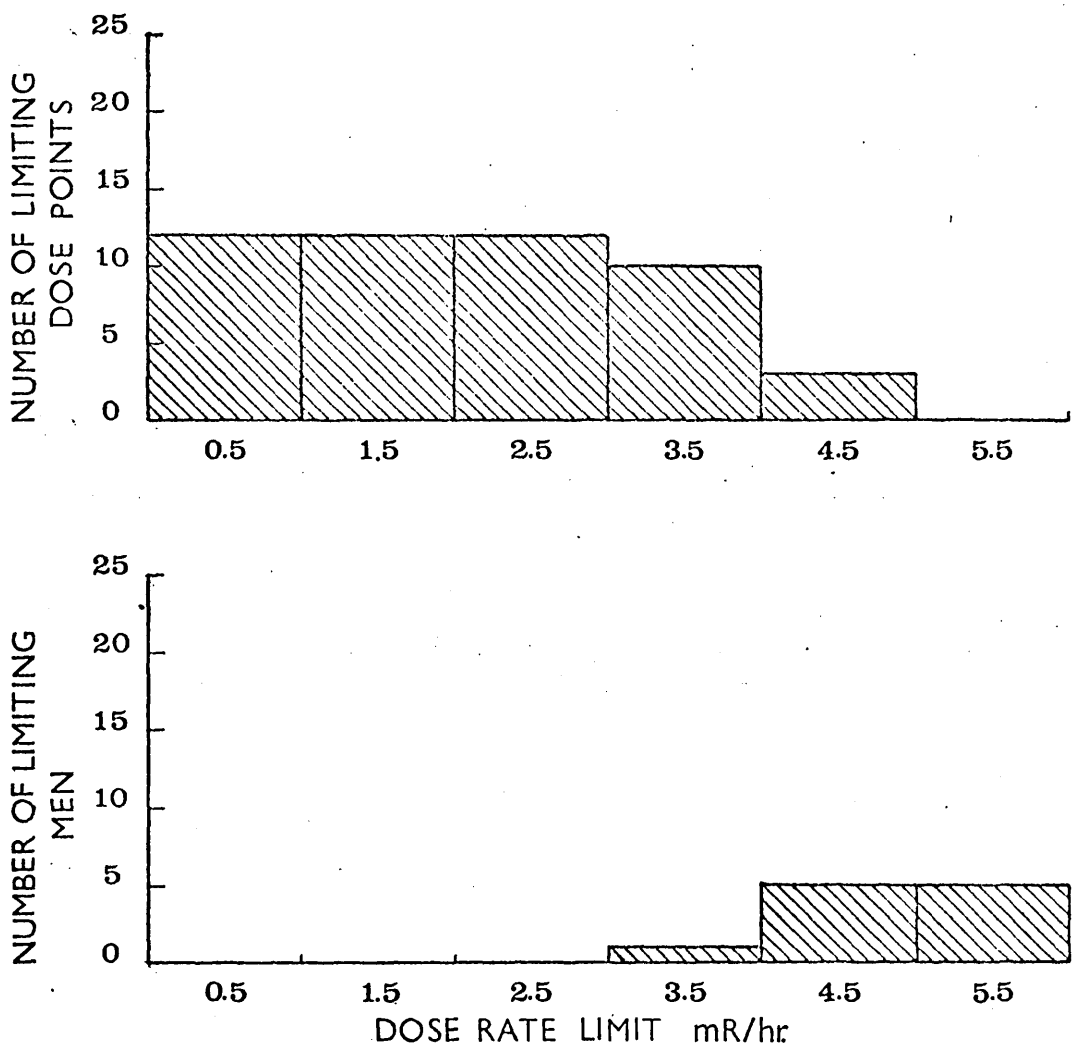

FIG. 7-2 Variation of Limiting Constraints with Area Dose Rate Limits.

the shield design. Figures 7-3 and 7-4 show the shield thickness perturbations for each shield element, for each of the six area dose rate limits used. When the area dose rate limit is set to 0.5 mR/hr all the perturbations are positive. This shows that only by increasing the total weight of the shield can such low area dose rates be met. As these limits are raised the shield sections can be reduced in thickness. This shows up as increasingly negative shield thickness perturbations.

Figure 7-5 shows how, as the area dose rate limit is increased, the average integrated dose per man increases. The curve is linear until the area dose rate limit of 3.5 mR/hr is reached and then an asymptotic value is rapidly attained. These results reflect the interaction between area dose rate limits, integrated dose limits and the work schedule. A reduction in shield weight invariably increases the dose rates observed. The maximum possible shield weight reduction is therefore achieved when the dose rate at every dose point is at the specified maximum. Thus, an optimization procedure which attempts to minimize shield weight will cause all dose rates to rise towards the maximum permitted levels. If, as in the calculations described here, the same dose rate limit is applied to all dose points it is unlikely that shield thicknesses may be manipulated to achieve a uniform dose rate distribution. The geometrical arrangement of both shield sections and source will usually preclude this. Although the shield model used here does not involve geometrical calculations directly the geometry of the shield is implicit in the contribution matrix, C_{ri}. When constraints on integrated doses for each worker are introduced it is even less likely that all area dose rate limits will be reached. Thus the weight reduction will fall further from the theoretical maximum attainable. This is apparent in figure 7-5 where an envelope is drawn showing the maximum value which the average integrated dose per man can achieve. The envelope is constructed by noting that, given an area

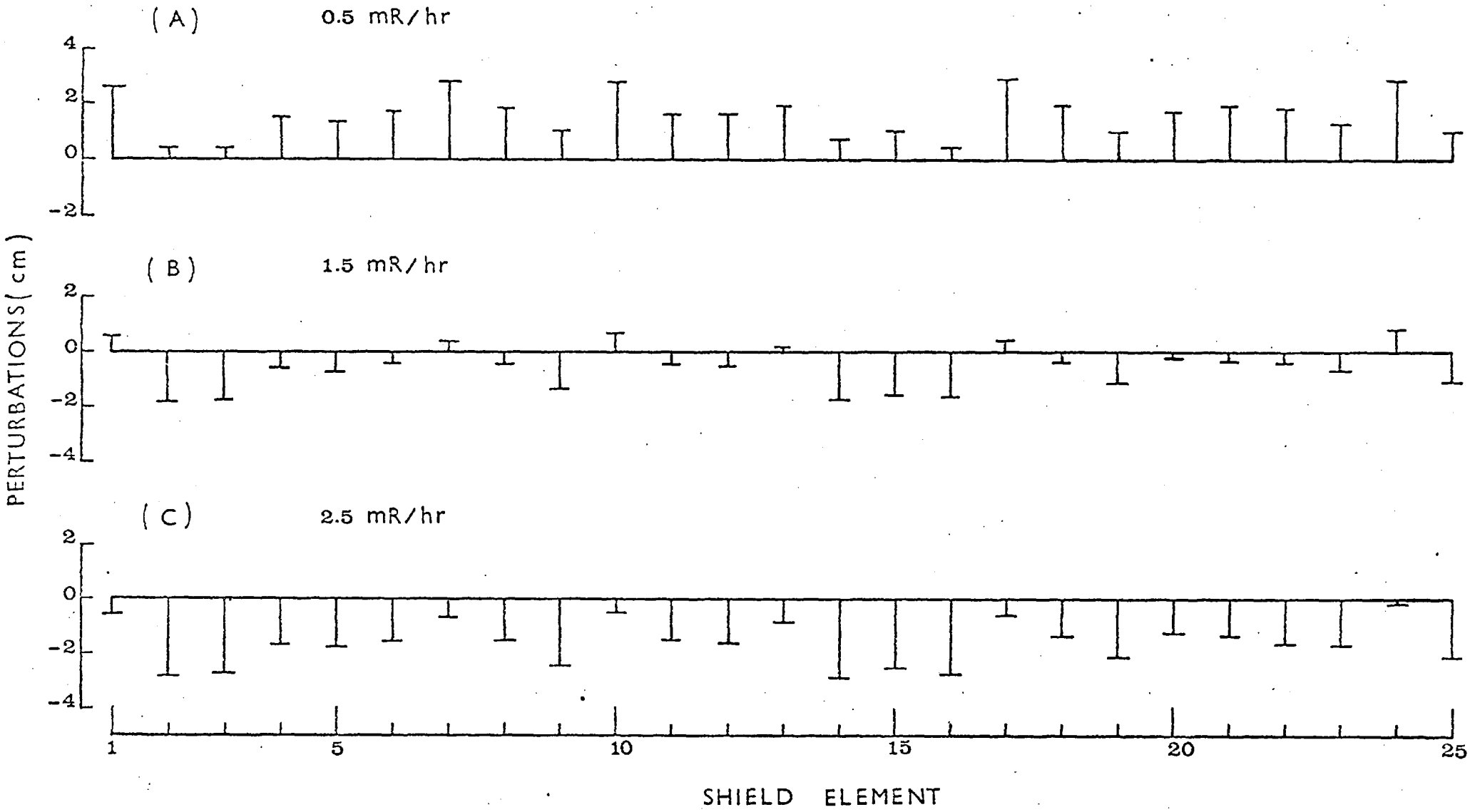

FIG. 7-3 Shield Element Thickness Perturbations (1).

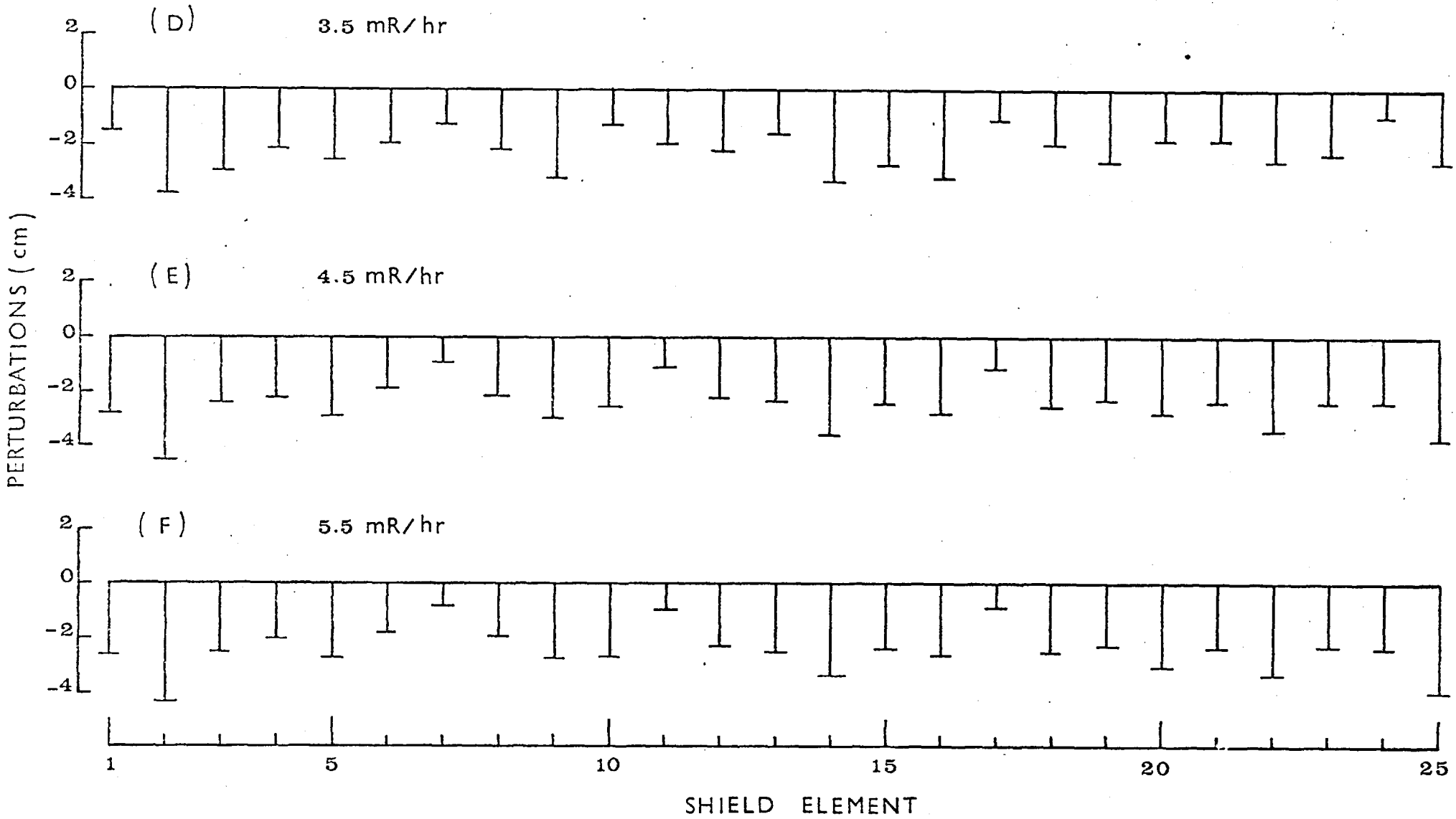

FIG. 7-4 Shield Element Thickness Perturbations (2).

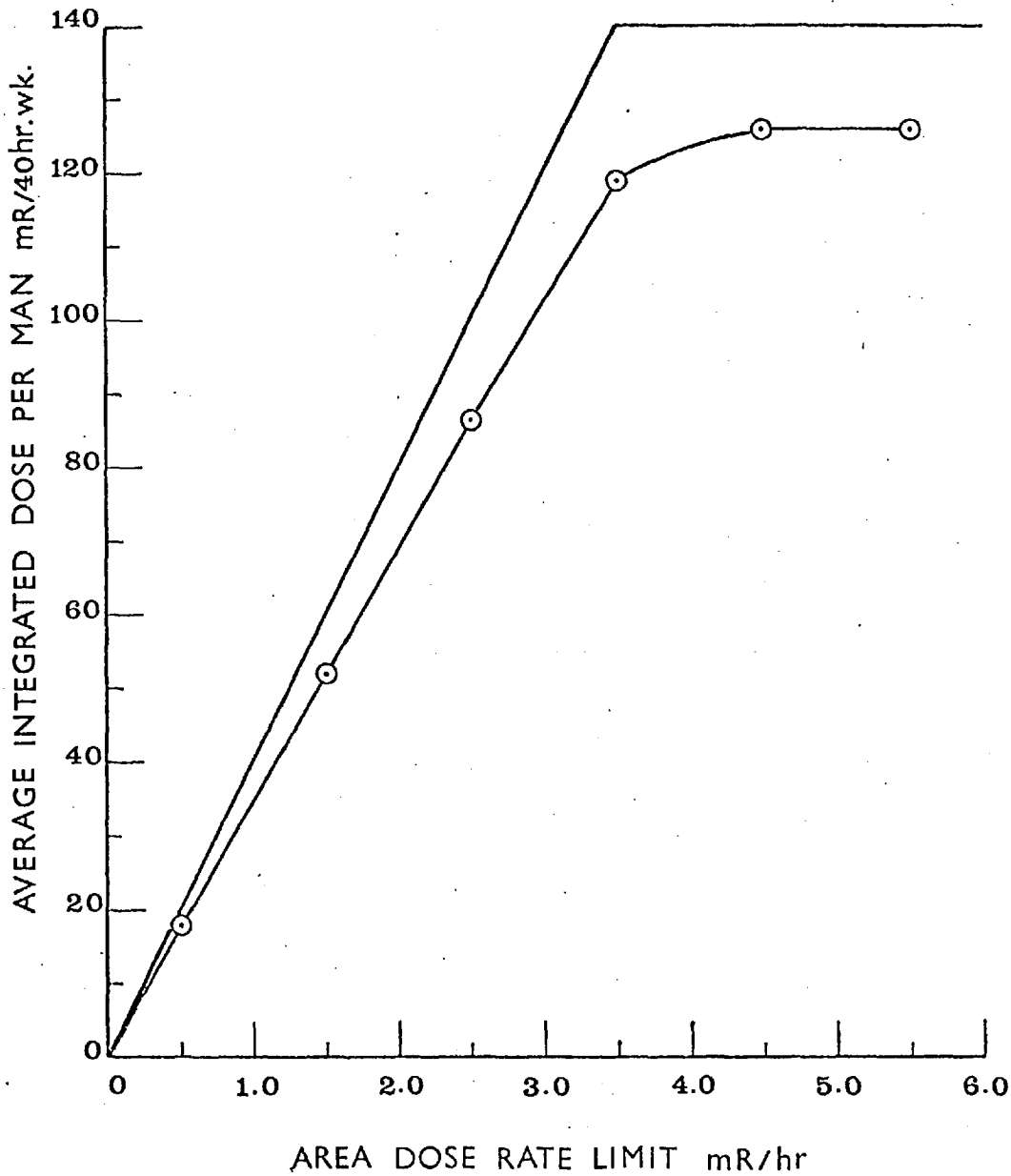

FIG. 7-5 Variation of Average Integrated Dose per Man with Area Dose Rate Limit.

dose rate limit, the maximum integrated dose which can be absorbed per week is equal to the total number of hours worked per week times the maximum hourly dose rate. Thus, for example, with an area dose rate limit of 2.5 mR/hr the maximum integrated dose attainable in a 40 hour week is 100 mR. The envelope has a sharp cut-off at 140 mR since this is the specified maximum value for all calculations.

For a given area dose rate limitation there is no way of improving the situation (*i.e.* reducing shield weight further) without either changing the geometrical arrangement of the shields or modifying the work schedule. For the model used here this corresponds to changing the elements of the arrays C_{ri} and F_{rm} respectively. The easier course of action involves changing the work schedule. This is an operational problem involving such questions as, 'Can all men do all jobs?', 'Must some jobs be done at the same time?', 'Are extra workers available?'. In some ways the problem of optimal scheduling of work is more difficult than that of optimizing the shield itself. The two problems interact in that by optimizing the shield system for a particular work schedule a new radiation field profile is produced for which a new optimal work schedule may then be devised. This new work scheme may then be used as the basis for further shield optimization.

The value of the shield optimization analysis is that it will immediately identify the limiting constraints. This enables operational decisions about work scheduling, redeployment and job automation to be made with full knowledge of the workers and/or dose rates which limit the design as a whole. Also with the aid of curves like that shown in figure 7-1 it enables area dose rate limits to be 'costed' directly in terms of shield weight.

The calculations described here each required approximately 300 seconds of central processor time on the CDC 6400 machine. They are therefore not

to be classed as small problems. However an easy way of reducing the number of constraints and hence the calculation time is to replace the area dose rate constraints by simple bounds on the tolerances. This limits the average dose rate at the surface of the shield and indirectly, the area dose rates. In this way the number of inequality constraints may be halved and a significant reduction in computing time achieved. Alternatively, larger problems could be attempted.

CHAPTER 8

CONCLUSIONS

The mode of development of an algorithm to solve the general non-linear programming problem has been described. The advantages gained by using an interactive test program for the development and evaluation of a computer code to implement the algorithm have been stressed. The provision of a range of test problems and the ability to obtain detailed diagnostic output has enabled numerous strategies to be tested and evaluated. This mode of development has produced a computer program which is both reliable and numerically robust.

The algorithm is conceptually simple and is therefore easily understood by the non-specialist. This basic simplicity has enabled a modular code to be developed. For example it would be an easy matter to change the cubic fitting procedure without modifying, say, the search routines in APPROX. A few small changes will allow the use of an alternative linear programming subroutine. This is an important advantage since although the subroutine LAØ1A has proved both reliable and efficient, there are circumstances where it may be necessary or desirable to use alternative linear programming routines.

Stress has been laid on the importance of a program which involves the user in as little preparatory work as possible. The proven ability of SLA to work with numerical derivatives is a strong argument in its favour. In a situation where the real costs of computing are falling, whilst the costs of human labour are rising, minimal problem preparation will become an increasingly important factor in the choice of solution technique. The ten large test problems described in Chapter 6 were all prepared and successfully run in the space of four working days. The

major part of this time was in fact spent punching the required computer cards.

The use of the displaced origin technique gives SLA a substantial computational advantage over the original formulation for successive linearization given by Griffith and Stewart. It is no longer necessary to double the dimensionality of a problem and this produces considerable savings in both storage requirements and calculation time. Cubic fitting and pattern moves provide an efficient method of convergence acceleration. The resulting algorithm has been shown to be superior to penalty function techniques and at least as good as the best projection methods currently available. Moreover it has the significant advantage over projection techniques of not requiring analytical derivatives. Since there are no doubts about the ability of the revised simplex technique to solve large linear programs the method may be used with confidence for large-scale non-linear optimization work.

Although the code SLA is a fully operational problem solving procedure there are two changes which, given more time, the author would have wished to investigate. The first is concerned with initial feasibility. The step doubling procedure currently employed when no feasible region is found is only of use where a feasible region exists but is not encompassed by the maximum step length constraints. A steepest descent phase using an object function of the form given for $T(x)$ in equation 2-17 would enable the search to progress even from a point where the linearized problem had no feasible solution. This would relieve the user of the chore of choosing a new starting point or of setting up an unconstrained problem as described in Section 4-3.

The second change involves reducing the number of inequality constraint equations supplied to the linear programming routine. It should be possible in the subroutine LINEAR to check each inequality constraint

in turn to ascertain whether all points in the region defined by the maximum step length limitations were feasible with respect to the constraint. If this were the case then the constraint could be dropped since only points within the region defined by the step length limitations are admissible as feasible points. If they are all feasible with respect to a particular constraint then that constraint is redundant. It would be necessary to investigate whether the time required to make such tests was in fact justified by the time savings due to a reduced linear program size. In large problems with many constraints the savings could be substantial.

In Chapter 7 it has been shown possible to optimize, in a minimum weight sense, a radiation shield taking into account via a work schedule the doses absorbed by individual workers within the shield structure. This has been made computationally possible by adopting a greatly simplified non-geometric model of the shield. The computer resource requirements are practicable for small to medium sized problems, *i.e.* less than 100 shield sections. The analysis presented makes it possible to locate the men and/or the dose points which limit the design as a whole. This information provides a good basis for the rearrangement of operational procedures. The shield model used here has been employed successfully in the optimization of marine reactor shielding, but unfortunately much of this work is of a classified nature and not reported in the open literature. The optimal shield analysis work described here could be usefully followed up by an investigation of the operational problems associated with the rearrangement of work schedules. This would involve precise specification of all the available options such as using extra workers, interchanging jobs between workers, automating high dose jobs. Given specifications such as these, efforts could be made to model the interaction of the shield optimization and work schedule problems. Such

an overall optimization procedure would undoubtedly require substantial computer resources for development and use. A careful appraisal of the likely benefits should be made before attempting such a development.

There is still much to be done in the field of non-linear programming. It seems unlikely that any one approach to the solution of such problems will prove to be the best for every type of problem. A much more likely outcome is that a range of efficient methods, each particularly suited to a sub-set of problems, will emerge. It is hoped that the algorithm described here will play a part in such an evolutionary process.

REFERENCES

1. HESTENES, M.R. and STEIFEL, E., "Methods of conjugate gradients for solving linear systems".
   J. Res. N.B.S. **49**, p.409 (1952)

2. FLETCHER, R. and REEVES, C.M. "Function minimization by conjugate gradients".
   Comp. J. **7**, p.149 (1964).

3. DAVIDON, W.C., "Variable metric method for minimization".
   A.E.C. Res. and Dev. Report, ANL-5990 (1959).

4. SPENDLEY, W., HEXT, G.R. and HIMSWORTH, F.R., "Sequential application of simplex designs in optimization and evolutionay operation".
   Technometrics **4**, pp.441-61 (1962).

5. NELDER, J.A. and MEAD, R., "A simplex method for function minimization".
   Comp. J. **7**, p.308 (1965).

6. ROSENBROCK, H.H., "An automatic method for finding the greatest and least value of a function".
   Comp. J. **3**, p.175 (1960).

7. SWANN, W.H., "Report on the development of a new direct search method of optimization".
   I.C.I. Ltd. Res. note 64/3 (1964).

8. POWELL, M.J.D., "An efficient method of finding the minimum of a function of several variables without calculating derivatives".
   Comp. J. **7**, p.155 (1964).

9. ROSEN, J.B., "The gradient projection method for non-linear programming".
   J. Soc. Ind. Appl. Math. **8**, p.181 (1960).

10. GOLDFARB, D. and LAPIDUS, L., "Conjugate gradient method for non-linear programming problems with linear constraints". *Ind. Eng. Chem. Fundamentals* 7, p.142 (1968).

11. ABADIE, J. and CARPENTIER, J., "Generalization of the Wolfe reduced gradient method to the case of non-linear constraints". *In* "Optimization", Ed. R. Fletcher, Academic Press (1969).

12. FIACCO, A.V. and McCORMICK, G.P., "Nonlinear Programming - Sequential Unconstrained Minimization Techniques". Pub. Wiley, New York (1968).

13. BOX, M.J., "A new method of constrained optimization and a comparison with other methods". *Comp. J.* 8, p.42 (1965).

14. PAVIANI, D. and HIMMELBLAU, D.M., "Constrained nonlinear optimization by heuristic programming". *Operations Res.* 17, p.872 (1969).

15. COLVILLE, A.R., "A comparative study on nonlinear programming codes". Paper in *Proc. of Princeton Symposium on Math. Prog.*, Ed. H.W. Kuhn, Princeton University Press (1970).

16. COLVILLE, A.R., "A comparative study on nonlinear programming codes". IBM, N.Y. Sci. Center Report, 320-2949 (1968).

17. GRIFFITH, R.E. and STEWART, R.A., "A nonlinear programming technique for the optimization of continuous processing systems". *Management Sci.* 7, p.379 (1961).

18. GASS, S.I., "Linear Programming". McGraw-Hill (1969).

19. BARNES, G.K., M.S. Thesis University of Texas, Austin, Texas (1967).

20. DIXON, L.C.W., "ACSIM - An accelerated constrained simplex technique". *Comp. Aided Des.* 5, p.22 (1973).

21. PAVIANI, D.A., Ph.D. Thesis
University of Texas, Austin, Texas (1969).

22. HIMMELBLAU, D.M., Applied Nonlinear Programming.
Pub. McGraw-Hill (1972).

23. FADDY, D. Private Communication.

APPENDIX 1 - TEST PROBLEMS USED

Specifications of all the test problems used in Chapter 6 are given here. As well as the form of the object function and constraints, details are given of the standard starting point, solution point and value of the object function at the optimum. The thirteen small test problems incorporated into N1OC are given first and then the ten larger problems are specified.

A1-1 Small Test Problems

(1) Rosenbrock's Function

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

Starting point: (-1.2, 1.0)

Solution point: (1.0, 1.0)

Optimum: $f = 0.0$

(2) Powell's Function

Minimize: $f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$

Starting point: (3, -1, 0, 1)

Solution point: (0, 0, 0, 0)

Optimum: $f = 0.0$

(3) Wood's Function

Minimize: $f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1(x_2 - 1)^2 + 10.1(x_4 - 1)^2 + 19.8(x_2 - 1)(x_4 - 1)$

Starting point: (-3, -1, -3, -1)

Solution point: (1, 1, 1, 1)

Optimum: $f = 0.0$

(4) Rosenbrock D

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

Constraints: $x_1 \leq 0$

$x_2 \leq 0$

Starting point: (-0.5, 0.5)

Solution point: (0, 0)

Optimum: $f = 1.0$

(5) Post Office Box A

Minimize: $f = -x_1x_2x_3$

Constraints: $0 \leq x_i \leq 42 \qquad (i = 1,3)$

$x_1 + 2x_2 + 2x_3 \leq 72$

Starting point: (10, 10, 10)

Solution point: (24, 12, 12)

Optimum: $f = -3456.0$

(6) Post Office Box B

Minimize: $f = -x_1x_2x_3$

Constraints: $0 \leq x_1 \leq 20$

$0 \leq x_2 \leq 11$

$0 \leq x_3 \leq 42$

$x_1 + 2x_2 + 2x_3 \leq 72$

Starting point: (10, 10, 10)

Solution point: (20, 11, 15)

Optimum: $f = -3300.0$

(7) Rosenbrock C

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

Constraint: $x_1^2 + (x_2 - 1)^2 \geq 0.9$

Starting point: (-1.2, 1.0)

Solution point: (-0.94147, 0.88322)

Optimum: f = 3.77029

(8) Post Office Box C

Minimize: $f = -x_1x_2x_3$

Constraints: $x_i \geq 0 \qquad (i = 1,3)$

$x_1^2 + 2x_2^2 + 4x_3^2 \leq 48$

Starting point: (1, 1, 1)

Solution point: (4.0, 2.828427, 2.0)

Optimum: f = -22.627416

(9) Sefton's Problem

Minimize: $f = Ax_1^{0.7}.(1000x_2)^2 + \frac{B}{(1000x_1x_2)}$

$(A = 0.1717.10^{-4}, \quad B = 200.0)$

Constraints: $0.005 \leq x_1 \leq 0.020$

$x_1(1000x_2)^2 \leq 2300$

$x_2(x_1)^{0.8} \leq 0.0223785$

Starting point: (0.0125, 0.0010)

Solution point: (0.02000, 0.33912)

Optimum: f = 29.6161

(10) Cattle Feed Problem

Minimize: $f = 24.55x_1 + 26.75x_2 + 39x_3 + 40.5x_4$

Constraints: $x_i \geq 0 \qquad (i = 1,4)$

$12x_1 + 11.9x_2 + 41.8x_3 + 52.1x_4 - 1.645[(0.53x_1)^2 +$

$+ (0.44x_2)^2 + (4.5x_3)^2 + (0.79x_4)^2]^{0.5} -$

$- 21 \geq 0$

$2.3x_1 + 5.6x_2 + 11.1x_3 + 1.3x_4 - 5 \geq 0$

$x_1 + x_2 + x_3 + x_4 = 1$

Starting point: $(10^{-5}, 10^{-5}, 0.9, 0.1)$

Solution point: (0.63588, 0, 0.31267, 0.05146)

Optimum: $f = 29.8888$

(11) Rosenbrock Ridge

Minimize: $f = -[100(x_2 - x_1^2)^2 + (1 - x_1)^2]$

Constraints: $x_2 \leq e^{-(1+x_1)}$

$x_2 = x_1^2$

Starting point: (0.5, 0.5)

Solution point: (-1.0, 1.0)

Optimum: $f = -4.0$

(12) Paviani Problem

Minimize: $f = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3$

Constraints: $x_i \geq 0 \qquad (i = 1,3)$

$x_1^2 + x_2^2 + x_3^2 = 25$

$8x_1 + 14x_2 + 7x_3 = 56$

Starting points: (2, 2, 2), (10, 10, 10)

Also: (1.0, 1.0, 4.8), (4.8,1.2, 0.0) and

(0.0, 1.8, 4.5)

Solution point: (0.35121, 0.21699, 3.5522)

Optimum: $f = 961.715$

(13) Rosenbrock CC

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

Constraint: $x_1^2 + (x_2 - 1)^2 = 0.9$

Starting points: (-1.2, 1.0), (-0.5, 0.0), (1.1, 0.6)

Solution points: (-0.94147, 0.88322), (0.39413, 0.13706),

(0.94198, 0.88742)

Optima: 3.77029, 0.40048, 0.00336724

A1-2 Large Test Problems

Problem 1

Source: M.J. Box, A New Method of Constrained Optimization and a Comparison with Other Methods, *Computer J.* 8:42 (1965).

No. of variables: 5

No. of constraints: 3 non-linear inequality constraints

10 upper and lower bounds.

Problem 1 is an example of determining parameters in highly non-linear differential equations from experimental data. The object function was the sum of squared residuals between experimental data and numerically integrated solutions of the differential equations.

Object function:

Maximize: $f(\mathbf{x}) = [50y_1 + 9.583y_2 + 20y_3 + 15y_4 - 852{,}960 - 38{,}100(x_2 + 0.01x_3) + k_{31} + k_{32}x_2 + k_{33}x_3 + k_{34}x_4 + k_{35}x_5]x_1 - 24{,}345 + 15x_6$

Calculation of x_6, the y_i's, and x_7, x_8:

$$x_6 = (k_1 + k_2x_2 + k_3x_3 + k_4x_4 + k_5x_5)x_1$$

$$y_1 = k_6 + k_7x_2 + k_8x_3 + k_9x_4 + k_{10}x_5$$

$$y_2 = k_{11} + k_{12}x_2 + k_{13}x_3 + k_{14}x_4 + k_{15}x_5$$

$$y_3 = k_{16} + k_{17}x_2 + k_{18}x_3 + k_{19}x_4 + k_{20}x_5$$

$$y_4 = k_{21} + k_{22}x_2 + k_{23}x_3 + k_{24}x_4 + k_{25}x_5$$

$$x_7 = (y_1 + y_2 + y_3)x_1$$

$$x_8 = (k_{26} + k_{27}x_2 + k_{28}x_3 + k_{29}x_4 + k_{30}x_5)x_1 + x_6 + x_7$$

where:

k_1	= -145,421.402	k_5	= 15,711.36
k_2	= 2,931.1506	k_6	= -161,622.577
k_3	= -40.427932	k_7	= 4,176.15328
k_4	= 5,106.192	k_8	= 2.8260078

$k_9 = 9{,}200.476$ $\quad$ $k_{23} = 16.243649$

$k_{10} = 13{,}160.295$ $\quad$ $k_{24} = -3{,}094.252$

$k_{11} = -21{,}686.9194$ $\quad$ $k_{25} = -5{,}566.2628$

$k_{12} = 123.56928$ $\quad$ $k_{26} = -26{,}237$

$k_{13} = -21.1188894$ $\quad$ $k_{27} = 99$

$k_{14} = 706.834$ $\quad$ $k_{28} = -0.42$

$k_{15} = 2{,}898.573$ $\quad$ $k_{29} = 1{,}300$

$k_{16} = 28{,}298.388$ $\quad$ $k_{30} = 2{,}100$

$k_{17} = 60.81096$ $\quad$ $k_{31} = 925{,}548.252$

$k_{18} = 31.242116$ $\quad$ $k_{32} = -61{,}968.8432$

$k_{19} = 329.574$ $\quad$ $k_{33} = 23.3088196$

$k_{20} = -2{,}882.082$ $\quad$ $k_{34} = -27{,}097.648$

$k_{21} = 74{,}095.3845$ $\quad$ $k_{35} = -50{,}843.766$

$k_{22} = -306.262544$

Constraints:

$$0 \leq x_1 \leq 5$$

$$1.2 \leq x_2 \leq 2.4$$

$$20 \leq x_3 \leq 60$$

$$9 \leq x_4 \leq 9.3$$

$$6.5 \leq x_5 \leq 7$$

$$x_6 \leq 294{,}000$$

$$x_7 \leq 294{,}000$$

$$x_8 \leq 277{,}200$$

Starting point:

$$\mathbf{x}^{(0)} = [2.52 \quad 2 \quad 37.5 \quad 9.25 \quad 6.8]^T$$

$$f[\mathbf{x}^{(0)}] = 2{,}351{,}243.5$$

Results:

$$\mathbf{x} = [4.538 \quad 2.400 \quad 60.000 \quad 9.300 \quad 7.000]^T$$

$$f(\mathbf{x}) = 5,280,254$$

$$x_6 = 75,570$$

$$x_7 = 198,157$$

$$x_8 = 277,200$$

Note: x_6 x_7 and x_8 are not independent variables. They may be calculated from x_1, x_2, x_3, x_4 and x_5.

SLA solution: $\mathbf{x} = [4.5374, 2.4, 60.0, 9.3, 7.0]^T$

$f(\mathbf{x}) = 5,280,340$

Problem 2

Source: Shell Development Co. (cited in Colville, *IBM N.Y. Sci. Center Rept.* 320-2949, June, 1968, p. 22).

No. of variables: 15

No. of constraints: 5 nonlinear inequality constraints

15 bounds on independent variables

This problem is the dual of Problem 3.

Object function:

$$\text{Maximize:} \quad f(\mathbf{x}) = \sum_{i=1}^{10} b_i x_i - \sum_{j=1}^{5} \sum_{i=1}^{5} c_{ij} x_{10+i} x_{10+j} - 2 \sum_{j=1}^{5} d_j x_{10+j}^3$$

Constraints:

$$2 \sum_{i=1}^{5} c_{ij} x_{10+i} + 3 d_j x_{10+j}^2 + e_j - \sum_{i=1}^{10} a_{ij} x_i \geq 0 \qquad j = 1, \ldots, 5$$

$$x_i \geq 0 \qquad i = 1, \ldots, 15$$

Feasible starting point:

$$x_i^{(0)} = 0.0001 \qquad i = 1,\ldots, 15, \quad i \neq 7$$

$$x_7^{(0)} = 60$$

$$f[x^{(0)}] = -2400.01$$

Infeasible starting point:

$$x_i^{(0)} = b_i^{(0)} \qquad i = 1,\ldots, 10$$

$$x_i^{(0)} = 0 \qquad i = 11, \ldots, 14$$

$$x_i^{(0)} = 1 \qquad i = 15$$

$$f[x^{(0)}] = 6829.06$$

Results:

$$\mathbf{x} = [0.0000 \quad 0.0000 \quad 5.1740 \quad 0.0000 \quad 3.0611 \quad 11.8395 \quad 0.0000$$
$$0.0000 \quad 0.1039 \quad 0.0000 \quad 0.3000 \quad 0.3335 \quad 0.4000 \quad 0.4283$$
$$0.2240]^T$$

$$f(\mathbf{x}) = -32.386$$

SLA solution:

$$\mathbf{x} = [0.0, \ 0.0, \ 5.1723, \ 0.0, \ 3.0616, \ 11.837, \ 0.0, \ 0.0$$
$$0.10381, \ 0.0, \ 0.30009, \ 0.33333, \ 0.40019,$$
$$0.42825, \ 0.22413]^T$$

$$f(\mathbf{x}) = -32.3487$$

(Note: The e_j, c_{ij}, d_j, a_{ij}, and b_j are given in the accompanying table.)

Data for Test Problems 2 and 3

j	1	2	3	4	5
e_j	-15	-27	-36	-18	-12
c_{1j}	30	-20	-10	32	-10
c_{2j}	-20	39	-6	-31	32
c_{3j}	-10	-6	10	-6	-10
c_{4j}	32	-31	-6	39	-20
c_{5j}	-10	32	-10	-20	30
d_j	4	8	10	6	2
a_{1j}	-16	2	0	1	0
a_{2j}	0	-2	0	4	2
a_{3j}	-3.5	0	2	0	0
a_{4j}	0	-2	0	-4	-1
a_{5j}	0	-9	-2	1	-2.8
a_{6j}	2	0	-4	0	0
a_{7j}	-1	-1	-1	-1	-1
a_{8j}	-1	-2	-3	-2	-1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
-40	-2	-.25	-4	-4	-1	-40	-60	5	1

Problem 3

Source: Shell Development Co. (cited in Colville, *IBM N.Y.Sci. Center Rept.* 320-2949, June, 1968, p. 21).

No. of variables: 5

No. of constraints: 10 linear inequality constraints
5 bounds on independent variables

Object function:

$$\text{Minimize:} \quad f(x) = \sum_{j=1}^{5} e_j x_j + \sum_{i=1}^{5} \sum_{j=1}^{5} c_{ij} x_i x_j + \sum_{j=1}^{5} d_j x_j^3$$

Constraints:

$$\sum_{j=1}^{5} a_{ij}x_j - b_i \geq 0 \qquad i = 1, \ldots, 10$$

$$x_j \geq 0 \qquad j = 1, \ldots, 5$$

Feasible starting point:

$$x^{(0)} = [0 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

$$f[x^{(0)}] = 20$$

Results:

$$x = [0.3000 \quad 0.3335 \quad 0.4000 \quad 0.4285 \quad 0.224]^T$$

$$f(x) = -32.349$$

SLA solution:

$$x = [0.30000, \quad 0.33347, \quad 0.40000, \quad 0.42831, \quad 0.22396]^T$$

$$f(x) = -32.3487$$

Problem 4

Source: Proctor and Gamble Co. (cited in Colville, *IBM N.Y.Sci. Center Rept.* 320-2949, June, 1968, p. 24).

No. of variables: 5

No. of constraints: 6 nonlinear inequality constraints

10 bounds on independent variables

Note that x_2 and x_4 are not included in the definition of $f(x)$.

Object function:

Minimize: $f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1$

$- 40792.141$

Constraints:

$$0 \leq 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \leq 92$$

$$90 \leq 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \leq 110$$

$$20 \leq 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 25$$

$$78 \leq x_1 \leq 102$$

$$33 \leq x_2 \leq 45$$

$$27 \leq x_3 \leq 45$$

$$27 \leq x_4 \leq 45$$

$$27 \leq x_5 \leq 45$$

Feasible starting point:

$$\mathbf{x}^{(0)} = [78.62 \quad 33.44 \quad 31.07 \quad 44.18 \quad 35.22]^T$$

$$f[\mathbf{x}^{(0)}] = -30367$$

Infeasible starting point:

$$\mathbf{x}^{(0)} = [78 \quad 33 \quad 27 \quad 27 \quad 27]^T$$

$$f[\mathbf{x}^{(0)}] = -32217$$

Results:

$$\mathbf{x} = [78.000 \quad 33.000 \quad 29.995 \quad 45.000 \quad 36.776]^T$$

$$f(\mathbf{x}) = -30665.5$$

SLA solution: exactly as above.

Problem 5

Source: J.D. Pearson, On Variable Metric Methods of Minimization, *Research Analysis Corp. Rept.* RAC-TP-302, McLean, Va., May, 1968.

No. of variables: 9

No. of constraints: 13 nonlinear inequality constraints

1 upper bound

The problem was to maximize the area of a hexagon in which the maximum diameter was unity.

Object function:

$$\text{Maximize: } f(x) = 0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$$

Constraints:

$$\begin{aligned}
1 - x_3^2 - x_4^2 &\geq 0 \\
1 - x_9^2 &\geq 0 \\
1 - x_5^2 - x_6^2 &\geq 0 \\
1 - x_1^2 - (x_2 - x_9)^2 &\geq 0 \\
1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 &\geq 0 \\
1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 &\geq 0 \\
1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 &\geq 0 \\
1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 &\geq 0 \\
1 - x_7^2 - (x_8 - x_9)^2 &\geq 0 \\
x_1x_4 - x_2x_3 &\geq 0 \\
x_3x_9 &\geq 0 \\
-x_5x_9 &\geq 0 \\
x_5x_8 - x_6x_7 &\geq 0 \\
x_9 &\geq 0
\end{aligned}$$

Starting point:

$$x_i^{(0)} = 1 \qquad i = 1, \ldots, 9$$

$$f[x^{(0)}] = 0$$

Results:

	Himmelblau	SLA	
	$[x_i=1 \quad (i=1,9)]$	$[x_i=1 \quad (i=1,9)]$	$[x_i=0.0 \quad (i=1,9)]$
$f(x)$	0.8660	0.674981	0.866025
x_1	0.9971	0.60791	-0.96676
x_2	-0.0758	0.20600	-0.25568
x_3	0.5530	0.53684	-0.26192
x_4	0.8331	0.84368	-0.96509
x_5	0.9981	-0.23585	-0.96675
x_6	-0.0623	0.74271	-0.25571
x_7	0.5642	-0.36894	-0.26195
x_8	0.8256	0.41993	-0.96508
x_9	$2.4 \cdot 10^{-6}$	1.00000	0

Problem 6

Source: A.R. Colville, A Comparative Study on Nonlinear Programming Codes, *IBM N.Y.Sci. Center Rept.* 320-2949, June, 1968, p.31.

No. of variables: 3

No. of constraints: 14 nonlinear inequality constraints

6 bounds on independent variables

Problem 6 is typical of problems in which functions are described by a self-contained computer subroutine.

Object function:

Maximize: $f(\mathbf{x}) = 0.063y_2y_5 - 5.04x_1 - 3.36y_3 - 0.035x_2 - 10x_3$

Constraints:

$$0 \leq x_1 \leq 2000$$

$$0 \leq x_2 \leq 16{,}000$$

$$0 \leq x_3 \leq 120$$

$$0 \leq y_2 \leq 5000$$

$$0 \leq y_3 \leq 2000$$

$$85 \leq y_4 \leq 93$$

$$90 \leq y_5 \leq 95$$

$$3 \leq y_6 \leq 12$$

$$0.01 \leq y_7 \leq 4$$

$$145 \leq y_8 \leq 162$$

Fortran description of the calculation of y_2 to y_8.

```
      Y(2) = 1.6*X(1)
 10   Y(3) = 1.22*Y(2) - X(1)
      Y(6) = (X(2) + Y(3))/X(1)
      Y2CALC = X(1)*(112.+13.167*Y(6)-0.6667*Y(6)**2)/100
      IF(ABS(Y2CALC - Y(2)) - 0.001)30,30,20
 20   Y(2) = Y2CALC
```

```
      GO TO 10
 30   CONTINUE
      Y(4) = 93.
100   Y(5) = 86.35+1.098*Y(6)-0.038*Y(6)**2+0.325*(Y(4)-89.)
      Y(8) = -133.+3.*Y(5)
      Y(7) = 35.82-0.222*Y(8)
      Y4CALC = 98000.*X(3)/(Y(2)*Y(7)+X(3)*1000.)
      IF(ABS(Y4CALC-Y(4))-0.0001)300,300,200
200   Y(4) = Y4CALC
      GO TO 100
300   CONTINUE
```

Feasible starting point:

$$\mathbf{x}^{(0)} = [1745 \quad 12000 \quad 110]^T$$

$$f[\mathbf{x}^{(0)}] = 868.6458$$

Results:

$$\mathbf{x} = [1728.37 \quad 16000 \quad 98.13]^T$$

$$f(\mathbf{x}) = 1162.036$$

SLA solution: exactly as above.

Problem 7

Source: G.K. Barnes, M.S. thesis, The University of Texas, Austin, Tex., 1967. Adapted from C.W. Carroll, Ph.D. dissertation. The Institute of Paper Chemistry, Appleton, Wis., 1959.

No. of variables: 5

No. of constraints: 4 linear inequality constraints
34 nonlinear inequality constraints (as listed - some can be eliminated)
10 bounds on independent variables

The object function in Problem 7 is the net profit of a hypothetical wood-pulp plant. The constraints (or model) include the usual material and energy balances as well as several empirical equations.

Object function:

$$\text{Maximize:} \quad f(x) = 0.0000005843y_{17} - 0.000117y_{14} - 0.1365 - 0.00002358y_{13} - 0.000001502y_{16} - 0.0321y_{12} - 0.004324y_5 - 0.0001\frac{c_{15}}{c_{16}} - 37.48\frac{y_2}{c_{12}}$$

Calculation of y_i's and c_i's:

$$y_1 = x_2 + x_3 + 41.6$$

$$c_1 = 0.024x_4 - 4.62$$

$$y_2 = \frac{12.5}{c_1} + 12.0$$

$$c_2 = 0.0003535x_1^2 + 0.5311x_1 + 0.08705y_2x_1$$

$$c_3 = 0.052x_1 + 78 + 0.002377y_2x_1$$

$$y_3 = \frac{c_2}{c_3}$$

$$y_4 = 19y_3$$

$$c_4 = 0.04782(x_1 - y_3) + \frac{0.1956(x_1 - y_3)^2}{x_2} + 0.6376y_4 + 1.594y_3$$

$$c_5 = 100x_2$$

$$c_6 = x_1 - y_3 - y_4$$

$$c_7 = 0.950 - \frac{c_4}{c_5}$$

$$y_5 = c_6c_7$$

$$y_6 = x_1 - y_5 - y_4 - y_3$$

$$c_8 = (y_5 + y_4)0.995$$

$$y_7 = \frac{c_8}{y_1}$$

$$y_8 = \frac{c_8}{3798}$$

$$c_9 = y_7 - \frac{0.0663y_7}{y_8} - 0.3153$$

$$y_9 = \frac{96.82}{c_9} + 0.321y_1$$

$$y_{10} = 1.29y_5 + 1.258y_4 + 2.29y_3 + 1.71y_6$$

$$y_{11} = 1.71x_1 - 0.452y_4 + 0.580y_3$$

$$c_{10} = \frac{12.3}{752.3}$$

$$c_{11} = (1.75y_2)(0.995x_1)$$

$$c_{12} = 0.995y_{10} + 1998$$

$$y_{12} = c_{10}x_1 + \frac{c_{11}}{c_{12}}$$

$$y_{13} = c_{12} - 1.75y_2$$

$$y_{14} = 3623 + 64.4x_2 + 58.4x_3 + \frac{146,312}{y_9 + x_5}$$

$$c_{13} = 0.995y_{10} + 60.8x_2 + 48x_4 - 0.1121y_{14} - 5095$$

$$y_{15} = \frac{y_{13}}{c_{13}}$$

$$y_{16} = 148,000 - 331,000y_{15} + 40y_{13} - 61y_{15}y_{13}$$

$$c_{14} = 2324y_{10} - 28,740,000y_2$$

$$y_{17} = 14,130,000 - 1328y_{10} - 531y_{11} + \frac{c_{14}}{c_{12}}$$

$$c_{15} = \frac{y_{13}}{y_{15}} - \frac{y_{13}}{0.52}$$

$$c_{16} = 1.104 - 0.72y_{15}$$

$$c_{17} = y_9 + x_5$$

Constraints:

$$y_4 - \frac{0.28}{0.72} y_5 \geq 0$$

$$1.5x_2 - x_3 \geq 0$$

$$21.0 - 3496 \frac{y_2}{c_{12}} \geq 0$$

$$\frac{62,212}{c_{17}} - 110.6 - y_1 \geq 0$$

$$213.1 \leq y_1 \leq 405.23$$
$$17.505 \leq y_2 \leq 1053.6667$$
$$11.275 \leq y_3 \leq 35.03$$
$$214.228 \leq y_4 \leq 665.585$$
$$7.458 \leq y_5 \leq 584.463$$
$$0.961 \leq y_6 \leq 265.916$$
$$1.612 \leq y_7 \leq 7.046$$
$$0.146 \leq y_8 \leq 0.222$$
$$107.99 \leq y_9 \leq 273.366$$
$$922.693 \leq y_{10} \leq 1286.105$$
$$926.832 \leq y_{11} \leq 1444.046$$
$$18.766 \leq y_{12} \leq 537.141$$
$$1072.163 \leq y_{13} \leq 3247.039$$
$$8961.448 \leq y_{14} \leq 26844.086$$
$$0.063 \leq y_{15} \leq 0.386$$
$$71,084.33 \leq y_{16} \leq 140,000$$
$$2,802,713 \leq y_{17} \leq 12,146,108$$
$$704.4148 \leq x_1 \leq 906.3855$$
$$68.6 \leq x_2 \leq 288.88$$
$$0 \leq x_3 \leq 134.75$$
$$193 \leq x_4 \leq 287.0966$$
$$25 \leq x_5 \leq 84.1988$$

Feasible starting point:

$$\mathbf{x}^{(0)} = [900 \quad 80 \quad 115 \quad 267 \quad 27]^T$$

$$f[\mathbf{x}^{(0)}] = 0.939$$

Results:

$$\mathbf{x} = [705.060 \quad 68.600 \quad 102.900 \quad 282.341 \quad 35.627]^T$$

$$f(\mathbf{x}) = 1.905$$

SLA solution:

$$\mathbf{x} = [705.17, \quad 68.600, \quad 102.90, \quad 282.32, \quad 37.584]^T$$

$$f(\mathbf{x}) = 1.90516$$

Problem 8

Source: J. Bracken and G.P. McCormick, "Selected Applications of Nonlinear Programming," John Wiley & Sons, Inc., New York, 1968.

No. of variables: 10

No. of constraints: 3 linear equality constraints

10 bounds on independent variables

Problem 8 is a problem in the chemical equilibrium at constant temperature and pressure.

Object function:

$$\text{Minimize:} \quad f(x) = \sum_{i=1}^{10} x_i \left(c_i + \ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right)$$

where $c_1 = -6.089$ $c_2 = -17.164$ $c_3 = -34.054$ $c_4 = -5.914$

$c_5 = -24.721$ $c_6 = -14.986$ $c_7 = -24.100$ $c_8 = -10.708$

$c_9 = -26.662$ $c_{10} = -22.179$

Constraints:

$$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 = 0$$

$$x_4 + 2x_5 + x_6 + x_7 - 1 = 0$$

$$x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0$$

$$x_i \geq 0 \qquad i = 1, \ldots, 10$$

Nonfeasible starting point:

$$x_i^{(0)} = 0.1 \qquad i = 1, \ldots, 10$$

$$f[x^{(0)}] = -20.961$$

Results:

	NLP	Flexible tolerance	GGS	GRG	SUMT	SLA
f(x)	-47.751	-47.736	-47.656	-47.761	-47.761	-47.761
x_1	0.0350	0.0128	0	0.0406	0.0407	0.0408
x_2	0.1142	0.1433	0.1695	0.1477	0.1477	0.1475
x_3	0.8306	0.8078	0.7536	0.7832	0.7832	0.7831
x_4	0.0012	0.0062	0	0.0014	0.0014	0.0014
x_5	0.4887	0.4790	0.5000	0.4853	0.4853	0.4857
x_6	0.0005	0.0033	0	0.0007	0.0007	0.0007
x_7	0.0209	0.0324	0	0.0274	0.0274	0.0264
x_8	0.0157	0.0281	0	0.0180	0.0180	0.0180
x_9	0.0289	0.0250	0.0464	0.0375	0.0373	0.0376
x_{10}	0.0751	0.0817	0.1536	0.0969	0.0969	0.0972

Problem 9

Source: J.M. Gauthier, IBM France (cited in Colville, *IBM N.Y.Sci. Center Rept.* 320-2949, June, 1968, p. 29).

No. of variables: 16

No. of constraints: 8 linear equality constraints

32 upper and lower bounds on the variables

Object function:

$$\text{Maximize:} \quad f(x) = -\sum_{i=1}^{16} \sum_{j=1}^{16} a_{ij}(x_i^2 + x_i + 1)(x_j^2 + x_j + 1)$$

Constraints:

$$\sum_{j=1}^{16} b_{ij}x_j = c_i \qquad i = 1,\ldots, 8$$

$$0 \leq x_j \leq 5 \qquad j = 1,\ldots, 16$$

Note: The a_{ij}, b_{ij} and c_i are given in the accompanying table.

Data for Problem 9

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a_{1j}	1			1			1	1								1
a_{2j}		1	1				1			1						
a_{3j}			1				1		1	1				1		
a_{4j}				1			1				1				1	
a_{5j}					1	1				1		1				1
a_{6j}						1		1							1	
a_{7j}							1				1		1			
a_{8j}								1		1					1	
a_{9j}									1			1				1
a_{10j}										1				1		
a_{11j}											1		1			
a_{12j}												1		1		
a_{13j}													1	1		
a_{14j}														1		
a_{15j}															1	
a_{16j}																1
b_{1j}	0.22	0.20	0.19	0.25	0.15	0.11	0.12	0.13	1.00							
b_{2j}	−1.46		−1.30	1.82	−1.15		0.80			1.00						
b_{3j}	1.29	−0.89			−1.16	−0.96		−0.49			1.00					
b_{4j}	−1.10	−1.06	0.95	−0.54		−1.78	−0.41					1.00				
b_{5j}				−1.43	1.51	0.59	−0.33	−0.43					1.00			
b_{6j}		−1.72	−0.33		1.62	1.24	0.21	−0.26						1.00		
b_{7j}	1.12			0.31			1.12		−0.36						1.00	
b_{8j}		0.45	0.26	−1.10	0.58		−1.03	0.10								1.00
c_i	2.5	1.1	−3.1	−3.5	1.3	2.1	2.3	−1.5								

Infeasible starting point:

$$x_i^{(0)} = 10 \qquad i = 1,\ldots, 16$$

$$f[x^{(0)}] = -209{,}457$$

Results:

$$x = [0.040 \quad 0.792 \quad 0.203 \quad 0.844 \quad 1.270 \quad 0.935 \quad 1.682 \quad 0.155 \quad 1.568 \quad 0.000 \quad 0.000 \quad 0.000 \quad 0.660 \quad 0.000 \quad 0.674 \quad 0.000]^T$$

$$f(x) = -244.900$$

SLA solution: as above.

Problem 10

Source: D.A. Paviani, Ph.D. dissertation, The University of Texas, Austin, Tex., 1969.

No. of variables: 24

No. of constraints: 12 nonlinear equality constraints
2 linear equality constraints
6 nonlinear inequality constraints
24 bounds on independent variables

This problem represents the minimization of the cost of blending multi-component mixtures.

Object function:

Minimize: $$f(x) = \sum_{i=1}^{24} a_i x_i$$

Note: See accompanying tables for the a_i's, b_i's, c_i's, d_i's, e_i's.

Constraints:

$$\Psi_i(\mathbf{x}) = \frac{x_{(i+12)}}{b_{(i+12)} \sum_{j+13}^{24} \frac{x_j}{b_j}} - \frac{c_i x_i}{40 b_i \sum_{j=1}^{12} \frac{x_j}{b_j}} = 0 \qquad i = 1,.., 12$$

$$\Psi_{13}(\mathbf{x}) = \sum_{i=1}^{24} x_i - 1 = 0$$

$$\Psi_{14}(\mathbf{x}) = \sum_{i=1}^{12} \frac{x_i}{d_i} + h \sum_{i=13}^{24} \frac{x_i}{b_i} - 1.671 = 0$$

where $h = (0.7302)(530)\ \frac{14.7}{40}$

$$\frac{-[x_{(i+3)} + x_{(i+15)}]}{\sum_{j=1}^{24} x_j + e_i} \geq 0 \qquad i = 1,2,3$$

$$\frac{-[x_i + x_{(i+12)}]}{\sum_{j=1}^{24} x_j + e_i} \geq 0 \qquad i = 4,5,6$$

$$x_i \geq 0 \qquad i = 1,\ldots, 24$$

Infeasible starting point:

$$x_i^{(0)} = 0.04 \qquad i = 1,\ldots, 24$$

$$f[x^{(0)}] = 0.14696$$

Results:

	Flexible tolerance	NLP	SUMT	SLA
f(x)	0.05700	0.09670	0.07494	0.051728
x_1^*	7.804E-03	9.537E-07	9.109E-03	0
x_2^*	1.121E-01	0	3.739E-02	0
x_3^*	1.136E-01	4.215E-03	8.961E-02	2.7895E-01
x_4^*	0	1.039E-04	1.137E-02	0
x_5^*	0	0	4.155E-03	0
x_6^*	0	0	4.184E-03	0
x_7^*	6.609E-02	2.072E-01	5.980E-02	0
x_8^*	0	5.979E-01	1.554E-02	0
x_9^*	0	1.298E-01	1.399E-02	0
x_{10}^*	0	3.350E-02	8.780E-03	0
x_{11}^*	1.914E-02	1.711E-02	1.231E-02	0
x_{12}^*	6.009E-03	8.427E-03	1.153E-02	4.1771E-02
x_{13}^*	5.008E-02	4.657E-10	7.570E-02	0
x_{14}^*	1.844E-01	0	7.997E-02	0
x_{15}^*	2.693E-01	0	2.797E-01	6.7785E-01
x_{16}^*	0	0	1.168E-02	0
x_{17}^*	0	0	2.347E-02	0
x_{18}^*	0	0	6.368E-03	0
x_{19}^*	1.704E-01	2.868E-04	2.028E-01	0
x_{20}^*	0	1.193E-03	7.451E-03	0
x_{21}^*	0	8.332E-05	4.547E-03	0
x_{22}^*	0	1.239E-04	1.010E-02	0
x_{23}^*	8.453E-04	2.070E-05	1.220E-03	0
x_{24}^*	1.980E-04	1.829E-05	1.810E-03	1.4215E-03

Data for Test Problem 10

i	a_i	b_i	c_i	d_i	e_i
1	0.0693	44.094	123.7	31.244	0.1
2	0.0577	58.12	31.7	36.12	0.3
3	0.05	58.12	45.7	34.784	0.4
4	0.20	137.4	14.7	92.7	0.3
5	0.26	120.9	84.7	82.7	0.6
6	0.55	170.9	27.7	91.6	0.3
7	0.06	62.501	49.7	56.708	
8	0.10	84.94	7.1	82.7	
9	0.12	133.425	2.1	80.8	
10	0.18	82.507	17.7	64.517	
11	0.10	46.07	0.85	49.4	
12	0.09	60.097	0.64	49.1	
13	0.0693	44.094			
14	0.0577	58.12			
15	0.05	58.12			
16	0.20	137.4			
17	0.26	120.9			
18	0.55	170.9			
19	0.06	62.501			
20	0.10	84.94			
21	0.12	133.425			
22	0.18	82.507			
23	0.10	46.07			
24	0.09	60.097			

APPENDIX 2 - THE SLA OPTIMIZATION PACKAGE

A FORTRAN listing is given of the five routines the user must modify in order to use the SLA package. Comment cards indicate what the user must add to the routine in order to specify his own problem. For convenience the process is broken down into eleven steps. Each step is accompanied by notes which explain what to do. In some cases advice on suitable parameter values to use is given. The listing shows how problem 11 of Appendix 1 should be prepared for solution using analytical derivatives. If numerical derivatives are required step 5 should set IDERIV = 0 and DELX(I) = 1.0E-07 for I-1,2. Steps 10 and 11 are then unncessary. In total the user must add 25 cards to complete the specification of problem 11. These cards are marked with X's in the last six columns.

```
      PROGRAM  SLA(INPUT=514,OUTPUT=514,TAPE5=INPUT,TAPE6=OUTPUT)
C
C
C       PROGRAM. SLA (SUCCESSIVE LINEAR APPROXIMATIONS)
C       MACHINE.  CDC 6000 SERIES
C       AUTHOR .  A.D.ROWE
C       DATE   .  AUTUMN  1974
C
C
C
C           ************************************************
C           * THIS PROGRAM SOLVES THE GENERAL NON-LINEAR   *
C           * PROGRAMMING PROBLEM.                         *
C           *                                              *
C           * MINIMIZE    U(X(I))          (I=1,N)         *
C           * SUBJECT TO  PHI(I).GE.0.0    (I=1,NCONS)     *
C           *             PSI(I).EQ.0.0    (I=1,NEQUS)     *
C           *                                              *
C           * FULL INSTRUCTIONS FOR THE USE OF THIS CODE   *
C           * ARE GIVEN ON THE COMMENT CARDS.THE USER MUST *
C           * INSERT CARDS IN THE DECK TO DESCRIBE HIS     *
C           * PARTICULAR PROBLEM                           *
C           ************************************************
C
C  STEP  1***DIMENSION WORKING ARRAYS.
C
C               NO. OF INDEPENDENT VARIABLES IS   N
C               NO. OF INEQUALITY CONSTRAINTS IS  NCONS
C               NO. OF EQUALITY CONSTRAINTS IS    NEQUS
C
C               LET   M=NCONS+NEQUS+N
C                     ID1=M+1
C                     ID2=NCONS+2*N+1
C
```

```
C               THERE ARE 22 ARRAYS TO BE DIMENSIONED.IF
C               N IS LESS THAN OR EQUAL TO 45 IGNORE (1)
C               BELOW AS THESE ARRAYS ARE ALREADY DIMENSIONED.
C
C           (1)DIMENSION AS (N) THE ARRAYS...
C               WORK3,WORK19,JELLY,DELX,XX,X,XINC,TEST,
C               RMAX,OSCI,XSTRT,STEP.
C
C           (2)DIMENSION AS (ID1) THE ARRAYS...
C               WORKB,WORKD,WORKE,WORKF.
C
C           (3)DIMENSION OTHER ARRAYS AS...
C               WORKA(ID1,ID1),WORKC(ID2),WORK1(M,N),WORK2(M),
C               PHI(NCONS),PSI(NEQUS)
C
C               *NOTE* IF NCONS=0 DIMENSION PHI AS 1.
C               *NOTE* IF NEQUS=0 DIMENSION PSI AS 1.
C               *NOTE* ALL THE ARRAYS SHOULD BE DIMENSIONED
C                      BY PUTTING THEM IN BLANK COMMON.
      COMMON
     1 WORK3(45)    , WORK19(45)   , JELLY(45)    , DELX(45)     ,
     2 XX(45)       , X(45)        , XINC(45)     , TEST(45)     ,
     3 RMAX(45)     , OSCI(45)     , XSTRT(45)    , STEP(45)     ,
     4WORKA(62,62),WORKB(62),WORKC(91),WORKD(62),WORKE(62),WORKF(62),
     5WORK1(61,45),WORK2(62),PHI(50),PSI(20)
      COMMON /RAGS/
     1      FACRED  , FACINC  ,  IDERIV , IPROB   , TINY    , ISTEP   ,
     2      IFUNC   , IEFE    , ICONS   , IECE    , LFAIL   , ICUBOP  ,
     3      IINC    , IDOWN   , IUP     , NBOT    , NTOP    , IPCNT
```

```
      COMMON /OPTI/ KO
      COMMON /WHAT/ ITHR,NNNN
      REAL  LOWER
      COMMON /BOUND/ UPPER(60),LOWER(60)
      DATA FACRED,FACINC,IFUNC,IEFE,ICONS,IECE,LFAIL,IDOWN,IUP,
     1IPCNT,ITHR,NSMAX,ICUROP /0.2,2.0,8*0,99999,5000,1/
C
C  STEP  2***SET UPPER AND LOWER BOUNDS ON ALL VARIABLES
C
C               *NOTE* BOUNDS MUST BE SET FOR ALL VARIABLES IN ARRAYS
C                      UPPER(I),LOWER(I),I=1,N
C               *NOTE* IF THERE ARE NO BOUNDS ON A VARIABLE SET UPPER
C                      AND LOWER JUST SUFFICIENT NOT TO CONSTRAIN IT.
C                      DO NOT USE ENORMOUSLY LARGE OR SMALL VALUES.
C
      DATA  UPPER /2*100.0 /                                          XXXXXX
      DATA  LOWER /2*-100.0 /                                         XXXXXX
C
C  STEP  3***SET PRINCIPAL PARAMETERS (N,NCONS,NEQUS)
C
      N=2                                                             XXXXXX
      NCONS=1                                                         XXXXXX
      NEQUS=1                                                         XXXXXX
C
C  STEP  4***SET STARTING POINT CONVERGENCE CRITERIA AND STEP LENGTH
C
C               *NOTE* SET STARTING VECTOR IN XSTRT(I),I=1,N
C               *ADVICE* A FEASIBLE STARTING POINT IS ALWAYS PREFERABLE
C               *NOTE* SET CONV. CRITERIA (ABSOLUTE) FOR EACH
C                      INDEPENDENT VARIABLE IN TEST(I),I=1,N
C               *ADVICE* VERY TIGHT CONVERGENCE IS COSTLY IN COMPUTER
C                        TIME.TRY 0.0001 ON EACH VARIABLE
C               *NOTE* SET INITIAL STEP LENGTH FOR EACH VARIABLE
C                      IN STEP(I),I=1,N
C               *ADVICE* CHOOSE STEP LENGTH AS 10 TO 50 PER CENT OF
C                        EXPECTED VARIABLE CHANGE
C
```

```
      DO  27  I=1,N                                                    XXXXXX
      STEP(I)=0.5                                                      XXXXXX
      TEST(I)=0.0001                                                   XXXXXX
   27 CONTINUE                                                         XXXXXX
      XSTRT(1)=0.5                                                     XXXXXX
      XSTRT(2)=0.5                                                     XXXXXX
C
C  STEP  5***CHOOSE ANALYTICAL OR NUMERICAL DERIVATIVES
C
C               *NOTE* IF IDERIV SET TO 1 THEN ANALYTICAL DERIVATIVES
C                      FOR BOTH OBJECT FUNCTION AND CONSTRAINTS MUST
C                      BE PROVIDED IN SUBROUTINE GRAD
C               *NOTE* IF IDERIV SET TO 0 THEN NUMERICAL DERIVATIVES
C                      USED.VALUES OF DELTA-X FOR USE IN A DIFFERENCE
C                      FORMULA MUST BE SET IN DELX(I),I=1,N
C               *ADVICE* NUMERICAL DERIVATIVES INVOLVE LESS
C                        PROBLEM PREPARATION
C               *ADVICE* DELTA-X=1.0E-07 IS USUALLY O.K.
C
      IDERIV=1                                                         XXXXXX
C
C  STEP  6***SET PRINT CONTROL PARAMETERS
C
C               *NOTE* SET IPRINT=DESIRED PRINT FREQUENCY
C               *NOTE* SET ISTEP=1 IF STEP LENGTHS TO BE PRINTED
C                      OTHERWISE SET ISTEP=0
C               *NOTE* SET NBOT AND NTOP.THIS CAUSES EVERY ITERATION
C                      FROM NBOT TO NTOP (INCLUSIVE) TO BE PRINTED
C
      IPRINT=5                                                         XXXXXX
      ISTEP=1                                                          XXXXXX
      NBOT=0                                                           XXXXXX
      NTOP=30                                                          XXXXXX
```

```
      CALL UREAL(XSTRT,U)
      WRITE(6,30)  U
   30 FORMAT(//*  U= *,E20.14//)
      IF(NCONS.EQ.0)  GO  TO  40
      CALL CONST(XSTRT,NCONS,PHI)
      WRITE(6,32)  (PHI(I),I=1,NCONS)
   32 FORMAT(//*  INEQUALITY CONSTRAINTS....*/(5E15.7))
   40 IF(NEQUS.EQ.0)  GO  TO  50
      CALL EQUAL(XSTRT,PSI,NEQUS)
      WRITE(6,33) (PSI(I),I=1,NEQUS)
   33 FORMAT(//*  EQUALITY CONSTRAINTS....*/(5E15.7))
   50 CONTINUE
      CALL SECOND(TIM1)
      CALL APPROX(N,RMAX,OSCI,NCONS,NEQUS,XSTRT,DELX,STEP,TEST,M,NSMAX,
     1NSTOP,IPRINT,IDATA,X,U,PHI,PSI,WORK1,WORK2,WORK3,WORK19,WORKA,
     2WORKB,WORKC,WORKD,WORKE,WORKF,JELLY,XX,XINC)
      CALL SECOND(TIM2)
      TIM1=TIM2-TIM1
      WRITE(6,2)  TIM1
    2 FORMAT(//*  CALCULATION TIME IS*,F7.3,* SECONDS*//)
      WRITE(6,3) IFUNC,IEFF,ICONS,IECE
    3 FORMAT(//*  IFUNC =*,I5/*  IEFF =*,I5,/*  ICONS =*,I5/*  IECE =*,
     1I5//)
      WRITE(6,4)  IDOWN,IUP,LFAIL,IPCNT
    4 FORMAT(//* NO. OF CUBIC REDUCTIONS    =*,I4/
     1          * NO. OF STEP INCREMENTS     =*,I4/
     2          * NO. OF ROOT FAILURES       =*,I4/
     3          * NO. OF PATTERN MOVES       =*,I4//)
      IF(NCONS.EQ.0)  GO  TO  41
      CALL CONST(X,NCONS,PHI)
      WRITE(6,32)  (PHI(I),I=1,NCONS)
   41 IF(NEQUS.EQ.0)  STOP
     CALL EQUAL(X,PSI,NEQUS)
      WRITE(6,33) (PSI(I),I=1,NEQUS)
      END
```

```
      SUBROUTINE UREAL(X,U)
C
C             ************************************************
C             * SUBROUTINE UREAL                             *
C             * THIS ROUTINE DEFINES THE OBJECT FUNCTION.    *
C             * IT MUST ALWAYS BE COMPLETED.                 *
C             ************************************************
C
      COMMON /RAGS/
     1      FACRED  , FACINC  ,  IDERIV , IPROB   , TINY    , ISTEP   ,
     2      IFUNC   , IEFE    , ICONS   , IECE    , LFAIL   , ICUBOP  ,
     3      IINC    , IDOWN   , IUP     , NBOT    , NTOP    , IPCNT
      DIMENSION  X(1)
C
C  STEP  7***DEFINE OBJECT FUNCTION
C
C                  *NOTE* SET U=(OBJECT FUNCTION)
C                  *NOTE* MAXIMUM(U) IS EQUIVALENT TO MINIMUM(-U)
C
      U=-(100.0*(X(2)-X(1)*X(1))**2+(1.0-X(1))**2)                    XXXXXX
C
C                  *NOTE* IFUNC COUNTS NO. OF TIMES UREAL IS CALLED
C
      IFUNC=IFUNC+1
      RETURN
      END
```

```
      SUBROUTINE CONST(X,NCONS,PHI)
C
C            ************************************************
C            * SUBROUTINE CONST                             *
C            * THIS ROUTINE DEFINES THE INEQUALITY          *
C            * CONSTRAINTS AND MUST ALWAYS BE COMPLETED     *
C            * UNLESS NCONS=0.                              *
C            ************************************************
C
      COMMON /RAGS/
     1      FACRED  , FACINC  ,  IDERIV , IPROB   , TINY    , ISTEP   ,
     2      IFUNC   , IEFE    , ICONS   , IECE    , LFAIL   , ICUBOP  ,
     3      IINC    , IDOWN   , IUP     , NBOT    , NTOP    , IPCNT
      DIMENSION  X(1)
      DIMENSION  PHI(1)
C
C  STEP  8***DEFINE INEQUALITY CONSTRAINTS, IF ANY
C
C                *NOTE* CONSTRAINTS ARE INPUT IN A FORM SUCH THAT
C                       PHI(I) IS G.E. 0.0 WHEN CONSTRAINT IS SATISFIED
C
      PHI(1)=EXP(-1.0-X(1))-X(2)                                        XXXXXX
C
C                *NOTE* ICONS COUNTS THE NO. OF CONSTRAINT EVALUATIONS
C
      ICONS=ICONS+NCONS
      RETURN
      END
```

```
      SUBROUTINE EQUAL(X,PSI,NEQUS)
C
C            ***************************************************
C            * SUBROUTINE EQUAL                                *
C            * THIS ROUTINE DEFINES THE EQUALITY               *
C            * CONSTRAINTS AND MUST BE COMPLETED UNLESS        *
C            * NEQUS=0.                                        *
C            ***************************************************
C
      COMMON /RAGS/
     1      FACRED  , FACINC  ,  IDERIV , IPROB   , TINY   , ISTEP   ,
     2      IFUNC   , IEFE    , ICONS   , IECE    , LFAIL  , ICUBOP  ,
     3      IINC    , IDOWN   , IUP     , NBOT    , NTOP   , IPCNT
      DIMENSION  X(1)
      DIMENSION  PSI(1)
C
C  STEP  9***DEFINE EQUALITY CONSTRAINTS, IF ANY
C
C                *NOTE* CONSTRAINTS ARE INPUT IN A FORM SUCH THAT
C                       PSI(I)=0.0 WHEN CONSTRAINT IS SATISFIED
C
      PSI(1)=X(2)-X(1)*X(1)                                          XXXXXX
C
C                *NOTE* ICONS COUNTS THE NO. OF CONSTRAINT EVALUATIONS
C
      ICONS=ICONS+NEQUS
      RETURN
      END
```

```
      SUBROUTINE GRAD(DELX,X,ICALL,N,CDERIV,NCONS,NEQUS,IDIM)
C
C          ************************************************
C          * SUBROUTINE GRAD                              *
C          * THIS ROUTINE DEFINES THE FIRST PARTIAL       *
C          * DERIVATIVES OF BOTH OBJECT FUNCTION AND      *
C          * CONSTRAINTS.IT IS ONLY COMPLETED IF IDERIV=1.*
C          ************************************************
C
      COMMON /RAGS/
     1      FACRED  , FACINC  ,  IDERIV , IPROB   , TINY    , ISTEP   ,
     2      IFUNC   , IEFE    , ICONS   , IECE    , LFAIL   , ICUBOP  ,
     3      IINC    , IDOWN   , IUP     , NBOT    , NTOP    , IPCNT
      DIMENSION  CDERIV(IDIM,N),DELX(1)
      DIMENSION  X(1)
C
C  STEP 10***DEFINE PARTIAL DERIVATIVES OF OBJECT FUNCTION
C
C               *NOTE* SET THE FIRST PARTIAL DERIVATIVE OF THE
C                      OBJECT FN. W.R.T. X(J) INTO DELX(J),J=1,N
C
      DELX(1)=-(400.0*X(1)*(X(1)*X(1)-X(2))+2.0*(X(1)-1.0))                  XXXXXX
      DELX(2)=-(200.0*(X(2)-X(1)*X(1)))                                      XXXXXX
C
C               *NOTE* IEFE COUNTS NO. OF EFFECTIVE FUNCTION EVALUATIONS
C
      IEFE=IEFE+N
C
C               *NOTE* IF NO CONSTRAINTS THEN RETURN
C
      IF(NCONS+NEQUS.EQ.0)  RETURN
C
C               *NOTE* INITIALIZE CDERIV SO THAT ONLY NON-ZERO
C                      ELEMENTS NEED BE ENTERRED
C
      DO  36  I=1,IDIM
      DO  36  J=1,N
   36 CDERIV(I,J)=0.0
C
```

```
C  STEP 11***DEFINE PARTIAL DERIVATIVES OF CONSTRAINTS
C
C                *NOTE* SET INTO CDERIV(I,J) THE FIRST PARTIAL DERIV.
C                       OF CONSTRAINT I W.R.T. X(J)
C                *NOTE* INEQUALITIES MUST BE ENTERED BEFORE EQUALITIES
C
      CDERIV(1,1)=-EXP(-1.0-X(1))                                          XXXXXX
      CDERIV(1,2)=-1.0                                                     XXXXXX
      CDERIV(2,1)=-2.0*X(1)                                                XXXXXX
      CDERIV(2,2)=1.0                                                      XXXXXX
C
C                *NOTE* IECE COUNTS NO. OF EFFECTINE FN. EVALUATINNS
C
      IECE=IECE+(NCONS+NEQUS)*N
      RETURN
      END
```