DEVELOPMENT OF A USER ORIENTED OPTIMIZATION SYSTEM FOR COMPUTER AIDED DESIGN PACKAGES

By

VIRENDRA KUMAR JHA, B.Sc, B. Tech.

A Thesis

Submitted to the school of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

(Feb.) 1971

MASTER OF ENGINEERING (1971) McMASTER UNIVERSITY

Hamilton, Ontario.

TITLE: Development of a User Oriented Optimization System For Computer Aided Design Packages.

AUTHOR: Virendra Kumar Jha, B.Sc., B. Tech. (I.I.T. Delhi)

SUPERVISOR: Professor J. N. Siddall

NUMBER OF PAGES: 129

SCOPE AND CONTENTS:

A new user oriented optimization system is described which is particularly useful for integration into user oriented design packages. Four new subroutines have been developed for the system, one being for integer or mixed integer nonlinear problems. A description is given of the problem of handling constraints while solving optimization problems. The technique of integration into a design package is discussed. Solutions of four sample problems have been included to demonstrate use of the subroutines.

ACKNOWLEDGEMENT

The author is grateful to Professor J. N. Siddall, for suggesting this project, and for his guidance and encouragement throughout the progress of this work.

The financial support provided by the National Research Council, Research Grant No. A7105, is gratefully acknowledged.

CONTENTS

APPENDIX

ILLUSTRATIONS

TEXT

CHAPTER - 1

INTRODUCTION

Optimization problems have long been of interest to scientists and engineers. Problems of optimization are those in which maximization or minimization of a function is sought. The function may be of one or more variables and with or without constraints. Often the problem may be to design a product in such a way that it meets certain specifications, while at the same time some objective function, such as cost or profit is minimized or maximized.

The field of optimization has attracted very wide interest in recent times, mainly because optimization problems can be encountered in all fields, in design engineering, in commerce, in government, in military service and so on. Development of high speed computers has made possible the use of various optimization techniques for solving these problems. However engineers or others who encounter optimization problems can not be expected to have the time and knowledge to write their own programs for optimization, therefore the availability of general optimization subroutines to engineers and others would save their time and energy.

An unfortunate characteristic of optimization is that no one technique is best for all types of problems.

Relative success of any method depends upon the form of the functions describing the given problem. It is a very difficult task to predict which method would be best for a particular problem, unless of course the problem is linear. Realizing this difficulty, a multitechnique optimization package OPTIPAC [2] was developed at McMaster University. This is a fully integrated package containing nine different methods. Any number may be called in one run to compare results. It soon became apparent that there was a need for a coordinated package of independent subroutines. These individually require much less memory and full variable dimensioning is possible. They can be much more conveniently integrated into a design package. This lead to the development of the OPTISEP [1] package, to which this thesis has made a major contribution. New programs have been developed and added to some of the most useful subroutines extracted from OPTIPAC, to make up the OPTISEP system. In contrast to OPTIPAC, new subroutines can be easily added at any time. The two systems share a strong emphasis on easily used documentation and easily used programs.

The convenient use of OPTISEP subroutines in a specific user oriented design package has proved to be a very valuable feature. It has been demonstrated that an engineer having only modest experience with and

understanding of both programming and optimization can use these subroutines, and write design packages using them.

In addition to adapting six of the OPTIPAC subroutines, four new techniques developed for this thesis have been added to the package. The first subroutine, SIMPLEX, is based upon a direct-search technique named simplex, first described by Himsworth, Spendley and Hext [3] in 1962. The technique was later on developed by Nelder and Mead [4] in 1963. This method has nothing in common with the standard simplex method for solving linear programming problems. It derives its name from the geometric figure simplex which plays an essential role in this method.

The second subroutine, MEMGRAD, is based upon a recent paper published by A. Miele and J. W. Cantrell [13] in 1969. This method makes use of the derivative and the step size during previous iteration to improve the current iteration and hence has been named the memory gradient method.

The third subroutine, DAVID, is based upon an algorithm originally proposed by Davidon [11] in 1959, and later on developed by Fletcher and Powell [12] in 1963. This technique makes use of the derivative of the function.

The fourth subroutine INTEGER is for a special class of problems, where there is an additional requirement that

some or all the variables have to be integers. Until now the methods of integer programming were used only for linear integer programming, and not for nonlinear ones. A few methods developed for nonlinear integer programming were developed for special cases, but could not be used for the general case. In this subroutine a branch and bound technique of integer programming has been used. This subroutine works quite satisfactorily on all types of nonlinear integer programming problems.

This thesis includes the underlying theory behind various methods used in writing the programs. Flow charts have been included to explain the logic of the methods. Complete Fortran listings of the programs and the documentation for the user have been included in the appendix. Test problems have been included to demonstrate the use of the subroutines.

CHAPTER - 2

HANDLING OF CONSTRAINTS IN SOLVING OPTIMIZATION PROBLEMS

Optimization problems without any constraints are rarely encountered in actual practice. Most of the problems are associated with certain constraints, which must be satisfied by the optimum solution. Constraints may be either equality or inequality, or both. For equality constraints, the value of constraining function should be equal to zero at the optimum point, where as for inequality constraints, it should be greater than or equal to zero, or any specified quantity.

Unfortunately most of the techniques developed for minimizing a function are applicable to minimizing an unconstrained function only, and hence can not be applied directly to solve a general optimization problem. The optimization problem must be suitably transformed into an unconstrained function before any minimization technique can be used.

The transformation of the constrained optimization problem into an unconstrained function is normally accomplished by defining an artificial objective function which is a function of the objective function and the constraints. Such an artificial unconstrained objective function has its minima lying in some feasible region.

However it is also possible in the case of special types of constraints to transform the independent design variables such that constraints are automatically taken care of. Constraints of the type in which a variable is constrained between upper and lower limits can be handled in this way. For example if a variable is to be greater than or equal to zero, the following transformation [5] could be used.

$$x_i = \text{abs}(\tilde{x}_i) \tag{2.1}$$

where x_i is the variable which is to be positive and $\tilde{x}_i$ is the unconstrained variable.

If the variable x_i is constrained between 0 and 1 the following transformation [5] could be used.

$$x_i = \text{Sin}^2 \tilde{x}_i \tag{2.2}$$

$$\text{or} \quad x_i = \frac{e^{-\tilde{x}_i}}{e^{+\tilde{x}_2} + e^{-\tilde{x}_i}} \tag{2.3}$$

For a general case where any variable is constrained between upper limit u_i and lower limit ℓ_i, the following transformation could be used.

$$x_i = \ell_i + (u_i - \ell_i) \sin^2 \tilde{x}_i \tag{2.4}$$

These transformations do not offer a general solution to the problem of handling constraints because they are restricted to special types of constraints.

Therefore the more general approach of transforming the function instead of the variable, is generally used.

A typical optimization problem has the following form:-

To minimize (2.5)

$$U = U(x_1, x_2, x_3, \ldots, x_n)$$

subject to the following constraints

$$\psi_j = \psi_j(x_1, x_2, x_3, \ldots x_n) = 0, \; j = 1, m$$

$$\phi_k = \phi_k\,(x_1, x_2, x_3, \ldots x_n) \geq 0, \; k = 1, p$$

where n is the number of variables

m is the number of equality constraints

p is the number of inequality constraints.

The general form of the transformed unconstrained artificial objective function is

$$P(x_1, x_2, \ldots x_n, r) = U + \sum_{i=1}^{p} \lambda_i(r) \cdot G(\phi_i(x)) + \sum_{i=1}^{m} \lambda_i(r) \cdot S(\psi_i(x)) \qquad (2.6)$$

where r is a weighting parameter, and $\lambda_i(r)$ are weighting functions. G and S are functions of inequality constraints and equality constraints respectively. The difference between various transformations of this type is the difference in the ways the functions G, S, and weighting functions are selected. A method generally proceeds by selecting a sequence of parameter r_t such that $r_t \geq 0$ and $r \to \infty$ as $t \to \infty$. For each value of r, the unconstrained artifical function

is optimized, and t is the number of such optimizations. Functions G and S are selected such that as $t\to\infty$, the quantity

$$F = \sum_{i=1}^{p} \lambda_i(r), G(\phi_i(x)) + \sum_{i=1}^{m} \lambda_i(r) \cdot S(\psi_i(x))$$

tends to zero. However parameter r_t may also be chosen such that as $t\to\infty$, $r_t\to 0$, then functions G and S are accordingly defined so that $F\to 0$ as $t\to\infty$. As $t\to\infty$, the optimum of the unconstrained artificial function converges to the optimum of the constrained optimization problem. Thus the constrained optimization problem is converted into an unconstrained optimization problem and solved.

The following transformations have been proposed for this purpose.

(a) Caroll [8] suggested that the problems with inequality constraints only, can be solved by transforming it into a function of the following type.

$$P(x,r_t) = U + \sum_{i=1}^{k} r_t \cdot G(\phi_i(x)) \qquad (2.7)$$

where parameter $r_t>0$ decreases as t increases and tends to zero as t tends to infinity. Either of the two functional forms could be used to define $G(\phi_i(x))$, which are

$$G(\phi_i(x)) = \sum_{i=1}^{k} \frac{1}{\phi_i(x)} \qquad (2.8)$$

and

$$G(\phi_i(x)) = -\sum_{i=1}^{k} \log(\phi_i(x)) \qquad (2.9)$$

G functions have been selected of this form because they tend to infinity as any constraint ϕ approaches zero. Because of this property, the value of the artificial unconstrained function immediately increases if the optimum tends to go near the constraint, and hence the point stays in the feasible region. This effect is more predominant in the initial stages of optimization; later on as t increases, the value of parameter r_t becomes smaller, and then the increase in the value of unconstrained artificial function because of small value of inequality constraint, is nullified by the small value of r, because in function (2.7), the contribution of G to the unconstrained function is the product of r_k and G. Because of this, as $t \to \infty$, the solution of the unconstrained function tends to the actual optimum.

(b) Fiacco and McCormick [9] have further developed this approach and have suggested the following transformation, which is applicable to solve any general optimization problem.

$$p(x, r_t) = U + r_t \sum_{\ell=1}^{k} \frac{1}{\phi_\ell(x)} + r_t^{-\frac{1}{2}} \sum_{j=1}^{m} (\psi_j(x))^2 \qquad (2.10)$$

It can be analytically proved that as $t \to \infty$ the solution of this unconstrained function approaches the solution of the actual problem. In this transformation the inequality constraints have been handled in a manner

similar to the one proposed by Caroll, and the same intuitive logic holds true. For equality constraints, Fiacco & Mccormic have introduced an additional term $(\psi_j)^2/\sqrt{r_t}$. Intuitively the addition of such a term can be explained as follows. As computation proceeds, the value of r_t decreases, this would in turn increase the value of the function $(\psi_j)^2/\sqrt{r_t}$, and since no minimization algorithm would permit an increase in the function, the magnitude of ψ_j would necessarily decrease to nullify the increase due to $1/\sqrt{r_t}$. In the limiting case as $t \to \infty$, ψ_j must tend to zero, otherwise the function $[\psi_j^2]/\sqrt{r}$ would tend to infinity. Thus inclusion of this term forces the equality constraint equal to zero when the optimum is reached.

The prerequisite for use of these transformations is that the solution is started from a feasible point for the inequalities. Fiacco and Mccormick suggest than an additional term for violated inequality constraints should be included in the transformed unconstrained function, similar to the one used for equality constraints. Addition of such a term would make violated inequality constraint equal to zero and would force a feasible solution.

(c) Another approach is to transform the constrained optimization problem into an unconstrained function in which violated constraints are severely penalized. The strategy was developed for direct search [7]. The unconstrained function has the following form.

$$P(x_1 x_2 \ldots x_n, r) = U + 10^{20} \sum_{j=1}^{m} |\psi_j(x)| + 10^{20} \sum_{k=1}^{p} \text{ABS (violated inequality constraint)}. \qquad (2.11)$$

This type of function puts a sort of wall around the feasible region and any feasible point stays in the feasible region. This type of transformation usually stalls quickly and does not handle equality constraints well. An infeasible start is permitted. All these transformations were tried for the optimization subroutines developed for this thesis. The one finally used has basically the same form as proposed by Fiacco and Mccormick. This has been found to give satisfactory answers, because of the high penalty there is some times a tendency to stall at inequality constraints, but this at least keeps the solution feasible. The unconstrained artificial objective function used for the subroutines of this thesis is as follows:

$$P(x_1, x_2, \ldots x_n) = U + r_t \sum_{i=1}^{p} \frac{1}{\phi_i(x)} + r_t^{-1/2} \sum_{j=1}^{m} (\psi_j(x))^2 + 10^{20} \sum_{i=1}^{p} \text{ABS (violated inequality constraint)} \qquad (2.12)$$

The selection of a sequence of reduction in the value of parameter r_t has been found to have significant affect upon convergence. Larger reduction in value or r_t helps in convergence. Generally useful values have been recommended in the documentation of these subroutines.

An interesting result was observed while using transformation (2.12). The value of parameter r_t was changed after every step, instead of changing it after each optimization of the unconcstrained function, as required by the algorithm. Convergence of the method to the optimum solution was faster as compared to the latter case. This feature has not been incorporated in the subroutines developed, because of the risk that reducing r_t after every step might force the solution to converge to a false optimum, as happens when too small a value of r_t is selected in the initial stages of optimization.

This has been a brief account of the problem of handling constraints in optimization, and has been included here to give some insight into the problem.

CHAPTER - 3

DESCRIPTION OF THE OPTISEP PROGRAMS

General Description

For using any of the optimization subroutines, the user writes a small main program, defining the input parameters etc. He also provides service subroutines to define the objective function and the constraints of his problem. These subroutines together with the small main program make up the user's input deck. Other subroutines necessary for execution may be stored on permanent file. Input parameters can be varied by the user to improve the efficiency of the method for his particular problem.

All subroutines have variable dimensioning; this helps in keeping the memory space required in the computer to a minimum. The user has the option of printing out input data and intermediate steps, by appropriately choosing the values of IDATA and IPRINT. If a method fails to find the optimum after a specified number of iterations, it exits without returning to the main program, and the results at the last iteration are printed out. If the optimum is found, then the optimum values are returned to the main program, and user has the option of printing out the final result by calling subroutine ANSWER, which has been written to print results in a standard format, or providing his own output. A typical calling program is

shown in Figure 1.

Service subroutines -

Information about the problem to be optimized is supplied through three service subroutines. The objective function, the equality constraints, and the inequality constraints are evaluated in subroutines UREAL, EQUAL, and CONST respectively. This convention for defining input was used in OPTIPAC, in order to standardize the input, and these subroutines are interchangeable between OPTIPAC and OPTISEP.

The user formulates his problem in the following way.

Minimize the objective function defining the optimization criterion.

$$U = U(x_1, x_2, x_3, \ldots x_n)$$

subject to equality constraints defining feasibility

$$\psi_j = \psi_j(x_1, x_2, x_3 \ldots x_n) = 0, \; j = 1, \; m.$$

and inequality constraints defining feasibility

$$\phi_k = \phi_k(x_1, x_2, x_3 \ldots x_n) \geq 0, \; k = 1, \; p$$

where x_i are independent or design variables.

n is the number of design variables.

m is the number of equality constraints.

p is the number of inequality constraints.

The user must formulate his problem in this manner. A problem of maximization can be solved by minimizing the negative of the function to be maximized. Similarly

inequality constraints of the form $\phi_k \leq 0$ can be converted to $\phi_k \geq 0$ by multiplying throughout by -1.

The input to the service subroutines is the x_i array containing current values of the design variables. The corresponding values of U, ϕ_k, and ψ_j are returned to the optimization subroutine that calls them. The objective function and the constraints can be expressed directly as FORTRAN arithmetic statements, such as

```
U = x(1) + x(2) + x(3) + 4 .*x(2)
PHI(1) = (X(1) **2) + (x(2) **3) + 5.
PSI(1) = x(3)*x(4) + 2.*x(1) *x(3)
```

If other statements are necessary in order to define U, PHI or PSI, they may be included in the service subroutine or incorporated in auxilary subroutines.

Additional details on the service subroutines are provided by the documentation of OPTISEP, included in the appendix.

Method Subroutines -

(a) Simplex, Direct Search Method (subroutine SIMPLEX)

A set of n+ 1 points in n dimensional space define a space called a simplex. This geometric figure plays an essential role in this method, and accounts for the name simplex.

Before going into the logic of the method, the following notation is defined.

Let x_h be the vertex corresponding to $f(x_h) = \max(f(x_i))$

where i = 1, n+1 . and x is the vector defining point i of the simplex.

Let x_s be the vertex corresponding to $f(x_s) = \max(f(x_i)),\ i \neq h$

Let x_ℓ be the point corresponding to $f(x_\ell) = \min f(x_i)$

Let x_0 be the centroid of all x_i, $i \neq h$ and is given by

$$x_0 = \frac{1}{n} \sum_{\substack{i=1 \\ i \neq h}}^{n+1} x_i \qquad (3.1)$$

The three basic operations used in the method are defined below.

Reflection - where x_h is reflected and new point x_r is obtained by the relation

$$x_r = x_0 + \alpha(x_0 - x_h) \qquad (3.2)$$

α is the reflection coefficient and is ≤ 1

Expansion - where x_r is expanded in the direction along which further improvement of the function value is expected. The relation used is

$$X_e = X_0 + \gamma(x_r - x_0) \qquad (3.3)$$

γ is the expansion coefficient and is > 1

Contraction - where simplex is contracted, the new point x_c is obtained by the following relation

$$x_c = x_0 + \beta(x_h - x_0) \qquad (3.4)$$

β is the contraction coefficient and satisfies $0 < \beta < 1$. The values of the function to be minimized are given by Uh, Us, Uℓ, Ur, Ue, Uc, Uo at points x_h, x_s, x_ℓ, x_r, x_e, x_c, x_0 respectively.

The simplex algorithm is as follows

(i) (n+1) points are initially generated in n dimensional space to form a simplex.

(ii) The function be minimized is evaluated at each of the vertices in order to determine x_h, x_s, x_ℓ, and x_0.

(iii) A reflection move is attempted and functional value evaluated at the reflected point x_r.

(iv) If Us $\geq$ Ur $\geq$ Uℓ, then x_h is replaced by x_r and the process is restarted beginning with step (ii).

(v) If however Ur < Uℓ, an expansion move is tried to see if the function continues to decrease in the direction of x_r x_0. The expansion succeeds if Uℓ > Ue, and in that case x_h is replaced by x_e. If the expansion does not succeed, x_h is replaced by x_r. In either case the process is restarted from step (ii).

(vi) If the reflection move in step (iii) yields x_r such that Uh > Ur > Us, x_h is replaced by x_r and a contraction move is made, however if Ur > Uh, a contraction move is made without replacing x_h by x_r.

(vii.) If the contraction fails, the last simplex is shrunk about the point of lowest function value x_ℓ by the relation,

$$x_i = \frac{1}{2}(x_i + x_\ell) \tag{3.5}$$

and the process is restarted from step (ii).

(ix) The search is presumed to have reached optimum of the corresponding artificial unconstrained objective function if

$$\frac{1}{n}\left\{\sum_{j=1}^{n+1}(U_j - U_0)^2\right\}^{1/2} \leq G \tag{3.6}$$

where G is a given small quantity, provided as a convergence criterion.

In subroutine SIMPLEX, the constraints of the problem are taken care of by forming an artificial unconstrained objective function which is of the form [9].

$$P(x_1,x_2,\ldots x_n,r) = U(x_1,x_2,\ldots x_n) + r_1 \sum_{k=1}^{p} \frac{1}{\phi_k(x_1,x_2,\ldots x_n)}$$

$$+ \sum_{j=1}^{m} \psi_j(x_1,x_2,\ldots x_n)^2 / \sqrt{r_1}$$

$$+ 10^{20} \sum \text{ABS (violated inequality constraints)} \tag{3.7}$$

r_1 is a positive constant (r_1 = 1.0 is normally taken as starting value). The value of r_1 is reduced by multiplying it by a factor REDUCE, after each optimum of the function P(3.7) is found. The optimum of the constrained problem is assumed to have been reached when after two successive optimum of the function P(3.7) the value of objective function U does not change significantly.

The value of the artificial objective function is returned to the subroutine by calling Subroutine OPTIMF 2. Subroutine ANSWER is used to print the results in the standard format.

The available experience with this method shows that it is very good. Given the sufficient number of iterations, it almost always converges to the optimum eventually.

Initial size of the simplex has been found to have some effect upon the efficiency of the method. It is better to start with a fairly large simplex.

The program logic is given in Figure 2.

(b) Memory Gradient Method (subroutine MEMGRAD)

This method [13] is an extension of the Fletcher and Reeves [10] method, the step size δx is determined from the relation

$$\delta x = -\alpha(g(x)) + \beta(\hat{\delta x}) \qquad (3.8)$$

where α and β are scalors chosen at each iteration so as to yield greatest decrease in the optimization function. The quantity $\hat{\delta x}$ is the previous step size. Selection of step δx depends on previous gradients and steps, hence the name memory gradient. The convergence property of the Fletcher and Reeves methods for quadratic functions is very good, this method retains that property, and in addition has one extra degree of freedom in the system of correction for δx, which should hopefully improve convergence.

The following quantities are defined

x the position vector at a particular stage

U the value of function at x

g(x) = the gradient at x, giving partial derivatives of the function at x, with respect to x_1, x_2 $x_3, x_4 \ldots x_n$

$\tilde{x}$ = the point following x

$\hat{x}$ = the point preceeding x

The algorithm is as follows.

(i) For a given point, g(x) is computed numerically. The vector $\hat{\delta x}$ is known from the previous iteration. $\hat{\delta x}$ is assumed = 0 for the first iteration.

(ii) Optimum values of the multipliers α and β are found by following a special search technique. α and β are those values which give the minimum value of the function.

$$\text{Let } f(\tilde{x}) = f(x - \alpha g(x) + \beta \delta\hat{x}) = F(\alpha,\beta) \tag{3.9}$$

α and β are actually the solutions of the symultaneous equations.

$$\tilde{g(x)}^T g(x) = 0 \tag{3.10}$$

$$\tilde{g(x)}^T \delta\hat{x} = 0 \tag{3.11}$$

where T denotes a transpose vector. These equations ensure that α and β are selected such that the new gradient vector $\tilde{g(x)}$ is orthogonal to the previous gradient vector and the previous step.

To begin the search, nominal values are given to α_0 and β_0, the starting values of α and β. The step sizes $\delta\alpha$ and $\delta\beta$ which are to be added to α_0 and β_0 are given by

$$\delta\alpha = - \mu (D1/D3) \text{ sign } (D4/D3) \tag{3.12}$$

$$\delta\beta = - \mu (D2/D3) \text{ sign } (D4/D3) \tag{3.13}$$

where

$$D1 = F\alpha F\beta\beta - F\beta F\alpha\beta \tag{3.14}$$

$$D2 = F\beta F\alpha\alpha - F\alpha F\alpha\beta \tag{3.15}$$

$$D3 = F\alpha\alpha F\beta\beta - F\alpha\beta^2 \tag{3.16}$$

$$D4 = F\alpha^2 F\beta\beta - 2F\alpha F\beta F\alpha\beta + F_\beta^2 F\alpha\alpha \tag{3.17}$$

and where $F\alpha$, $F\beta$, $F\alpha\alpha$, $F\beta\beta$ are computed at $(\alpha 0, \beta_0)$, that is at the point $\tilde{x}_0$ given by

$$\tilde{x}_0 = x - \alpha_0 g(x) + \beta_0 \hat{\delta x} \tag{3.18}$$

The symbol μ, which is $0 \leq \mu \leq 1$, is a scaling factor for the increments $\delta\alpha$ and $\delta\beta$.

$F\alpha$, $F\beta$, $F\alpha\alpha$, $F\beta\beta$, and $F\alpha\beta$ are given by the following

$$F\alpha = - g^T(\tilde{x}_0)\ g(x) \tag{3.19}$$

$$F\beta = g^T(\tilde{x}_0)\ \hat{\delta x}$$

$$F\alpha\alpha = \{g[\tilde{x}_0 + \varepsilon_1 g(x)] - g[\tilde{x}_0 - \varepsilon_2 g(x)]\}^T\ g(x)/2\varepsilon_1 \tag{3.20}$$

$$F\beta\beta = \{g[\tilde{x}_0 + \varepsilon_2 \hat{\delta x}] - g[\tilde{x}_0 - \varepsilon_2 \hat{\delta x}]\}^T\ \hat{\delta x}/2\varepsilon_2 \tag{3.21}$$

$$F\alpha\beta = \{g[\tilde{x}_0 - \varepsilon_1 g(x)] - g[\tilde{x}_0 + \varepsilon_1 g(x)]\}^T\ \hat{\delta x}/2\varepsilon_1 \tag{3.22}$$

where ε is a small number and $\varepsilon_1 = \varepsilon/|g(x)|$

$$\varepsilon_2 = \varepsilon/|\hat{\delta x}|$$

Values of $\delta\alpha$ and $\delta\beta$ are computed by equations (3.12) and (3.13) for $\mu = 1$. α and β are calculated by $\alpha = \alpha_0 + \delta\alpha$, $\beta = \beta_0 + \delta\beta$. If $F(\alpha,\beta) < F(\alpha_0,\beta_0)$, $\mu = 1$ is acceptable, otherwise it is replaced by a smaller value until $F(\alpha,\beta) < F(\alpha_0,\beta_0)$. At this stage one search step for α and β is complete. The values of α and β are replaced by α_0 and β_0 for the next search step. The procedure is repeated until abs $(\delta\alpha/\alpha_0)$ and abs $(\delta\beta/\beta_0)$ becomes very small. Values of α and β at this stage are optimum values for the current iteration of the memory gradient method. However for the first iteration these equations are not valid because then $\hat{\delta x} = 0$. For this case $\delta\alpha$ is given by

$$\delta\alpha = - \mu(F\alpha/F\alpha\alpha)\ \text{sign}\ (F\alpha\alpha) \tag{3.23}$$

β remains zero for the first iteration.

(iii) The correction $\delta\alpha$ is determined by equation (3.8)

(iv) The new position of $\tilde{x}$ is computed by

$$\tilde{x} = x + \delta x \tag{3.24}$$

The optimum is assumed to have been reached when the value of the function does not decrease by more than a small specified quantity.

In subroutine MEMGRAD, values of the partial derivatives of the artificial objective function are returned by subroutine PARTIAL. Subroutine SUPPLY numerically calculates partial derivatives of the actual objective function U, of the inequality constraints, and of the equality constraints, which are called by PARTIAL, where these values are suitably combined to give derivatives of the artificial unconstrained objective function. Constraints of the problem are taken care of by forming an artificial unconstrained objective function, of a form similar to that used in SIMPLEX.

The value of parameter r is reduced each time after optimizing the unconstrained artificial objective function, the process continues until the difference between the two values of the actual objective function, corresponding to two successive optima, is insignificant.

Available experience with MEMGRAD shows that its convergence is faster in most of the cases than that of

SIMPLEX, but it hangs up more often. It is good for well behaved functions.

A flow chart explaining the logic of the program is given in Figure 3.

(c) Davidon Fletcher and Powell Method. (subroutine DAVID)

This method [12] is a gradient type of method. The use of the knowledge of the function and its gradient at a previous iteration is made to improve the current iteration. The directional vector d_i is generated at the i^{th} iteration in such a way that it is orthogonal to all previous vectors (d_i, i=1, 2, ...i-1) rather than just the $(i-1)^{st}$ as in ordinary gradient minimization. This vector d_i then defines the down hill direction for the function. At the $(i+1)^{st}$ iteration, vector x is computed by

$$x_{i+1} = x_i + \lambda d_i \qquad (3.25)$$

where λ is a scalar parameter, giving the optimum step length. The complete algorithm is as follows:

(i) compute $d_i = - H_{i-1} g_i$

(ii) Compute λ to minimize $f(x_i+\lambda d_i)$. In order to find the right value of λ, the function is assumed to be of one variable λ, and a search strategy to minimize a function of one variable is applied. There are various techniques which can be used "Polynomial search' is the one which has been used in this program. The search begins by establishing bounds on the value of λ. In order to establish bounds,

first a small value of λ say λ_1 is chosen. The value of λ is increased in steps by using the relation

$$\lambda_k = \lambda_1 (1+r+r^2+r^3 \ldots r^{k-1}) \qquad (3.26)$$

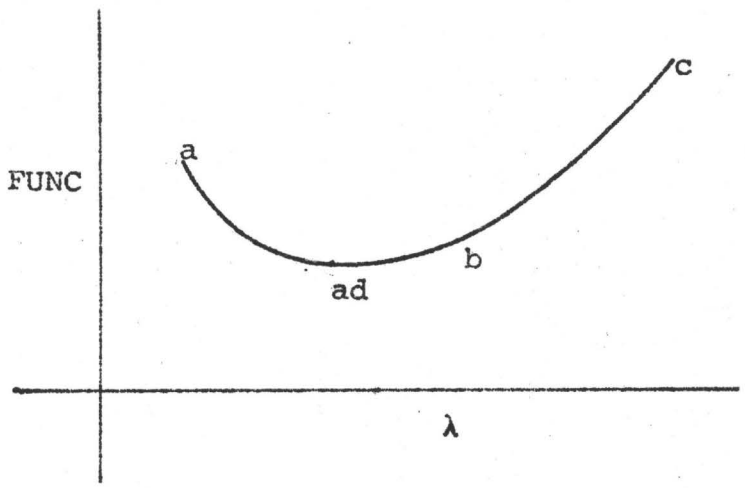
until λ_k is such that $F(x_i + \lambda_k d_i) > F(x_i + \lambda_{k-1} d_i)$. The value of r is arbitrarily chosen around 1 or 2. Values of λ which bound the minimum of the function are given by λ_{k-2}, λ_{k-1} and λ_k. Let a, b, c denote these values, and let Fa, Fb, Fc be the value of the function, corresponding to these values of λ respectively. The turning point of the approximate polynomial passing through these points is given by

$$ad = \frac{1}{2} \frac{(b^2 - c^2)\ Fa + (c^2 - a^2)\ Fb + (a^2 - b^2)\ Fc}{(b - c)\ Fa + (c - a)\ Fb + (a - b)\ Fc} \qquad (3.27)$$

Let Fd be the value of the function corresponding to λ = ad. Shrinking of the interval is done as follows.

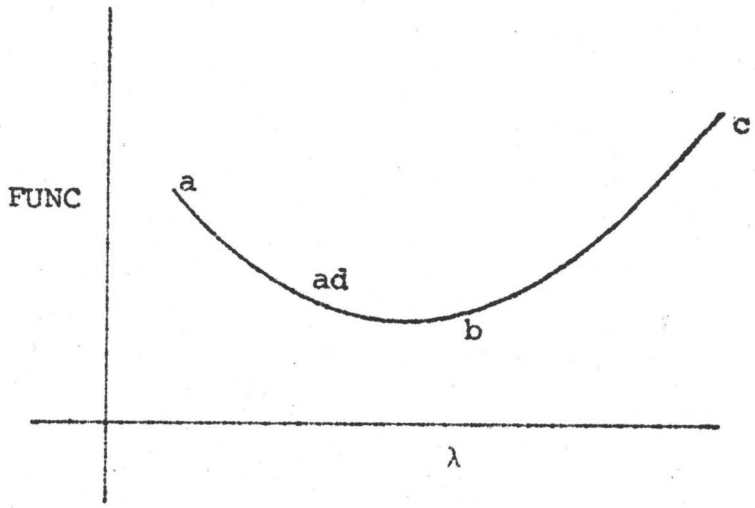
There are four possible situations, sketched below. The replacement of one point by another is done as indicated.

1. b > ad and Fb > Fd



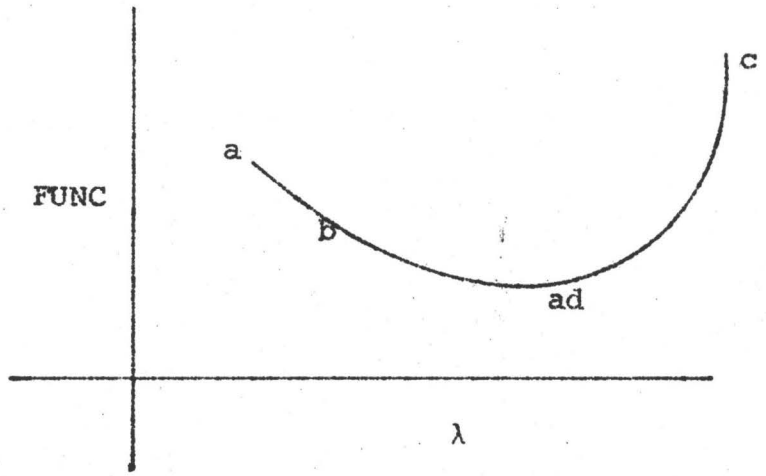
In this case for the next polynomial fit b takes the value of ad and c takes the value of b.

2. b > ad and Fb < Fd



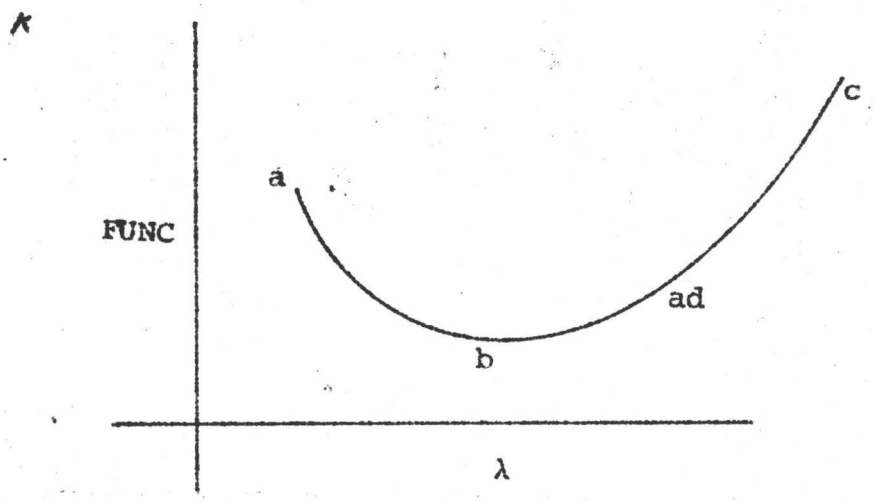
In this case a takes the value of ad, and b and c remain the same for the next polynomial fit.

3. b < ad and Fb > Fd



In this case a takes the value of b and b takes the value of ad, for the next polynomial fit.

4. b < ad and Fb < Fd



In this case c is replaced by ad, for the next polynomial fit.

Interval bounds are successively reduced by repeated polynomial fits, and the value of λ determined to any value of desired accuracy.

(iii) Having determined the value of λ, the next point x_{i+1} is determined by the relation (3.25) which is

$$x_{i+1} = x_i + \lambda d_i$$

(iv) The Matrix H_i to be used in the next iteration to determine orthogonal directions. It is given by

$$H_i = H_{i-1} + \frac{\lambda\, d_i\, d_i^T}{g_i^T H_{i-1}\, g_i} - \frac{H_{i-1}\, \gamma_i\, \gamma_i^T H_{i-1}}{\gamma_i^T H_{i-1}\, \lambda_i} \qquad (3.28)$$

where $\gamma_i = g_{i+1} - g_i$

In the first step H_0 is set as a unit matrix so that the first step becomes equivalent to a step in the steepest descent method. There are rigourous analytical proofs available to show that,

(a) Computation of the directional vector d_i as done in step (i) satisfies the condition of orthogonality.

(b) H_i is positive definite if H_{i-1} is positive definite. These two properties imply that convergence is faster and for a quadratic function, the minimization is reached in a finite number of steps, which is equal to N, the number of variables. However, (b) condition requires that the exact optimum value of λ be known.

In actual practice the exact value of λ may not be found by the search procedure employed, and hence the matrix H_i as computed from H_{i-1} may not be positive definite, under these circumstances matrix H should be reset to a unit

matrix.

Subroutine DAVID using this method has full variable dimensioning like other methods.

Subroutine FIND returns the exact value of λ to subroutine DAVID. Constraints of the problem are taken care of by forming an artificial unconstrained objective function similar to the one used in SIMPLEX, subroutine OPTIMF2 computes this function. Partial derivatives are computed numerically in subroutine PARTIAL and returned whenever PARTIAL is called.

This is a good method for well behaved functions; it is quite fast but tends to hang up quite easily. The high magnitude of the penalty term used in the formulation of unconstrained artificial objective function (3.7), was found to be a source of difficulty. The partial derivatives of the artificial function near the constraints were too steep because of these high penalty terms, and hence the method did not work in the expected way. This problem was eliminated by suitably modifying the partial derivatives when ever the derivative would have been too steep because of high penalty terms. This was done by adding a small penalty while computing the derivatives, instead of a high one. With this modification performance of the method has considerably improved. However, when the function defining the objective function itself is too steep,

such problems may still be encountered.

The flow chart explaining the logic of the method is shown in Fig [4].

(d) Non Linear Integer Programming (subroutine INTEGER)

Quite a few optimization problems require that some or all of the design variables should have integer values. This type of problem arises whenever there are indivisibilities; for instance it is not too meaningful to schedule 3.25 flights between two cities, or assign 6.8 machines for a particular job. In the past various methods have been applied to linear integer programming. In design engineering, functions are very rarely linear and there is a great need for a program of integer programming which handles nonlinear functions.

General methods of integer programming can be broadly classified in four catagories.

1. Cutting Plane Methods
2. Rounding Methods
3. Branch and Bound Methods
4. Partition Methods

Cutting plane methods [14] are only suitable for linear programming problems. The rounding methods [15] are not good in the sense that an optimal integer solution is found only if the non integer optimum solution is very close.

The branch and bound technique [16] works on the general idea of scanning all feasible integer solution in a systematic way, and is one which could be applied to non-linear integer programming. The partition algorithm [17] has been successful only for small problem and hence can not be used for any general problem.

Considering all this, there are only two approaches left for nonlinear integer programming. One is to linearize the function at a point and use Gomory's cutting plane method [23]. The other alternative is to use a branch and bound method. The first approach of linearizing the nonlinear functions and subsequently applying cuts, to make the solution integer was basically an attempt of integrating the two techniques together. Griffith and Steward [24] have developed a method of successive linear approximation for solving nonlinear problems, and Gomory [23] has proposed a cutting plane method of solving linear integer programming problems. The attempted algorithm is as follows.

(i) Obtain the continuous optimum solution of the non linear problem.

(ii) Starting with this optimum solution $(x_1^0, x_2^0, \ldots x_n^0)$ approximate the functions by expansion in a Taylor's series, in which terms above linear are dropped. Functions U, ϕ and ψ are thus approximated as follows.

$$U = U(x_1^0, x_2^0, \ldots x_n^0) + \sum_{i=1}^{n} (x_i - x_i^0) \frac{\partial U(x_1^0, x_2^0, \ldots x_n^0)}{\partial x_i} \tag{3.29}$$

$$\psi_j(x_1^0, x_2^0, \ldots x_n^0) + \sum_{i=1}^{n} (x_i - x_i^0) \frac{\partial \psi_j(x_1^0, x_2^0, \ldots x_n^0)}{\partial x_i} = 0, \; j = 1,m \tag{3.30}$$

$$\phi_k(x_1^0, x_2^0 \ldots x_n^0) + \sum_{i=1}^{n} (x_i - x_i^0) \frac{\partial \phi_k(x_1^0, x_2^0 \ldots x_n^0)}{\partial x_i} \geqslant 0, \; k = 1,p \tag{3.31}$$

These equations can be rewritten in the following form.

$$U - U_0 = \sum_{i=1}^{n} c_i \delta x_i \tag{3.32}$$

$$\sum_{i=1}^{n} u_{ji} \delta x_i = - \psi_j^0 \tag{3.33}$$

$$\sum_{i=1}^{n} v_{ki} \delta x_i \geqslant - \phi_k^0 \tag{3.34}$$

where

$$c_i = \frac{\partial U(x_1^0, x_2^0, \ldots x_n^0)}{\delta x_i} \tag{3.25}$$

$$U^0 = U(x_1^0, x_2^0 \ldots x_n^0) \tag{3.36}$$

$$u_{ji} = \frac{\partial \psi_j(x_1^0, x_2^0, \ldots x_n^0)}{\delta x_i} \tag{3.37}$$

$$\phi_k^0 = \phi_k(x_1^0, x_2^0, \ldots x_n^0) \tag{3.38}$$

$$v_{ki} = \frac{\partial \phi_k(x_1^0, x_2^0, \ldots x_n^0)}{\partial x_i} \tag{3.39}$$

$$\delta x_i = x_i - x_i^0 \tag{3.40}$$

Thus the problem is linearized and δx_i becomes the variables to be determined. In order to ensure that δx_i is positive, the following substitution is used.

$$\delta x_i = \delta x_i^+ - \delta x_i^- \tag{3.41}$$

where δx_i^+ and δx_i^- are positive. Another constraint is added to the above set of equations to limit the step size δx_i, to a small amount so that linearization remains valid.

(iii) Use revised simplex method of solving linear programming, to solve this set of linear equations together with additional constraints to keep the variables integer.

(iv) Compute x_{i+1} using $x_{i+1} = x_i + \delta x_i$

(v) Repeat the procedure beginning with step (ii), till solution is reached.

This algorithm failed to produce any results. Problem came when using revised simplex method. It failed to find a feasible solution. The possible reason for its failure may have been the constraint on size of δx_i. The linear approximation of the function is valid only when the step size δx_i is less than a small specified quantity, and the step size δx_i, required to make the solution integer may be bigger than the limiting

size, this thing might have caused the problem. When this approach failed, the branch and bound method was tried and it has shown appreciable success.

The branch and bound technique was first proposed by Land and Doig [16] and consisted of a systematic search of continuous solutions in which variables to be integers are successively forced to take integer values. The method as proposed by Land and Doig had substantial practical difficulties for computer applications, it requires recording of all the solutions which could involve excessive storage space. The method was modified from a computational point of view by R. J. Dakin [18]. The logic of subroutine INTEGER is based upon this modified method. This method, called a tree-search algorithm, is simple in concept, but like all other integer methods, it is lengthly, requiring many optimization runs.

The algorithm starts by finding a normal continuous solution to the given problem. Let the solution for the i^{th} variable be

$$k_i < x_i < k_{i+1} \tag{3.42}$$

where k_i is an integer. For the next trial, it is assumed that one of the variables say x_1 must be either equal to k_1 or k_1+1. The tree thus begins with the two branches. The integer solution for x_1 is forced by adding the constraint $x_1 \leq k_1$ in one branch, and $x_1 \geq k_1+1$ in the other. This again generates two branches. One of these branches is arbitrarily abandoned. As the adding of constraints for each variable in turn is continued, one of the previously integerized variable may become non-integer. For example x_1 previously pushed to k_1, may begin to drift below k_1. It then must be re-examined with constraints $x_1 \leq k_1-1$ in one branch and $x_1 \geq k_1$ in the second. This process continues until the desired integer solution is reached, or until a non-feasible solution is reached.

The last node having an unexplored branch is then searched, following the same procedure as before. A record is kept of the current best integer solution. An illustrative tree for three integer variables is shown in Fig. 8. The nodes are numbered as they are generated, so that when a solution is reached, or an infeasibility, the search returns to the next lower node, and a marker is checked to see if both branches have been explored. If it

has not, a new node is generated on the unexplored branch. When all possible nodes have been generated and branches explored, the search terminates.

For solving the nonlinear problem during the first step and subsequent steps with additional constraints, the Hook and Jeeves [19] direct search method has been used. It is incorporated in subroutine SOLVE. The constraints of the problem are taken care of by forming an unconstrained artificial objective function of the type used in SIMPLEX.

Subroutine INTEGER can be used for solving all integer or mixed integer, linear or nonlinear problems. The user simply specifies the number of variables to be made integer. The problem must be formulated in such a way that the variables to be made integers are the first k design variables of the problem, beginning with x_1, where k is the number of variables to be made integer. Thus if three out of five variables are to be made integer, then those variables should be x_1, x_2 and x_3.

Additional constraints for making a variable integer are supplied by subroutine ADDL, which returns the right-constraint at a particular stage. An additional statement card CALL ADDL(X, PHI), must be included in subroutine CONST, just before the RETURN statement.

Subroutine INTEGER has been tried on various test problems both linear and nonlinear and has been found to

work satisfactorily. In some of the cases the Hook and Jeeves search may fail to find non-integral solution of the problem, and in that case it is necessary to find a non-integral solution of the problem by using any other method of nonlinear optimization. Subroutine INTEGER is then used, using the optimum non-integral solution as starting values for INTEGER. A flow chart explaining the logic for subroutine INTEGER is given in Fig. 6, and the logic for subroutine ADDL in Fig. 7.

CHAPTER - 4

ILLUSTRATIVE PROBLEMS

Many problems have been solved by using subroutine SIMPLEX, MEMGRAD, DAVID, AND INTEGER. A few of those have been included here to demonstrate the use of these subroutines. The first two problems have been taken from a book by Siddall [6], and have been solved by subroutines SIMPLEX, MEMGRAD, and DAVID. The other two problems, in which an integer optimum solution is required, have been solved by subroutine INTEGER, especially written for solving nonlinear problems requiring an integer optimum solution.

The problems are as follows:

Problem 1 - Design of a Pressure Vessel.

Problem is to optimize the design of an unfired cylindrical welded pressure vessel. The ASME Code for unfired pressure vessel specifies that the shell thickness shall be the greater of the following

$$t_1 = \frac{PR}{SE - 0.6P} \tag{4.1}$$

t_1 is based on circumferential stress. (4.2)

or $$t_1 = \frac{PR}{2SE + 0.4P}$$

t_1 is based on longitudinal stress

where P = design Pressure

R = Inside radius

S = maximum allowable stress

E = Joint efficiency

The heads are to be semi-ellipsoidal in which half the minor axis equals one quarter of the inside diameter. The code specifies that the head thickness shall be determined by

$$t_2 = PR/(SE - 0.1P) \qquad (4.3)$$

All joints are to be single welded butt joints with backing strips. The efficiency from the code for such joints is 0.90. Volume of the vessel is to be 2000 imp gallon, and a design pressure of 500 p.s.i. The material is to be SA201B, table UCS-23 in the code gives an allowable stress of 15000 p.s.i. There is a length limitation of 30' and diameter limitation of 15 ft. maximum. Furthermore vessel must accomodate a heating coil 100" long and 40" in diameter.

Formulation

The design variables are

x_1 = thickness of cylindrical portion of pressure vessel

x_2 = thickness of the cap of pressure vessel

x_3 = O.D. of the cylindrical portion

x_4 = length of cylindrical portion of the pressure vessel.

The optimization criterion is to minimize the material

cost, that is the volume of the material.

Volume of the cylindrical portion is

$$= \frac{\pi}{4} (x_3^2 - (x_3 - 2x_1)^2) \times 4$$

Volume of the cap (both ends) is

$$= \frac{4}{3} \pi \left[\frac{x_3^3}{16} - \frac{(x_3 - 2x_2)^3}{16} \right] = \frac{\pi}{12} \left[x_3^3 - (x_3 - 2x_2)^3 \right]$$

Total volume of pressure vessel

$$= \frac{\pi}{4} \left[(x_3^2 - (x_3 - 2x_1)^2 \right] x_4 + \frac{\pi}{12} \left[x_3^3 - (x_3 - 2x_2)^3 \right]$$

Therefore the optimization function is

U = total volume of pressure vessel.

Constraints on the design are as follows:

(1) constraints on thickness of shell.

$$\phi_1 = x_1 - \frac{PR}{SE - 0.6P}$$

$$\phi_2 = x_1 - \frac{PR}{2SE + 0.4P}$$

(ii) constraint on thickness of ellipsoidal portion.

$$\phi_3 = X_2 - \frac{PR}{SE - 0.1P}$$

(iii) constraint on maximum length

$$\phi_4 = 30 - x_4 + \frac{(x_3 - 2x_1)}{2}$$

(iv) constraint on maximum diameter

$$\phi_5 = 15 - x_3$$

(v) constraint on minimum length

$$\phi_6 = \frac{x_3 - 2x_1}{2} + x_4 - \frac{100}{12}$$

(vi) constraint on minimum diameter

$$\phi_7 = (x_3 - 2x1) - \frac{40}{12}$$

(vii) constraints to keep design variable positive

$$\phi_8 = x_1$$

$$\phi_9 = x_2$$

$$\phi_{10} = x_3$$

$$\phi_{11} = x_4$$

Formulation for programming.

There are four design variables.

$$x(1) = x_1 \ , \ x(2) = x_2 \ , \ x(3) = x_3 \ , \ x(4) = x_4 \ .$$

There are 11 inequality constraints

$$PHI(1) = \phi_1, \ PHI(2) = \phi_2 \ \ldots \ PHI(11) = \phi_{11}$$

There is no equality constraint.

Function to be minimized is

U = volume of material

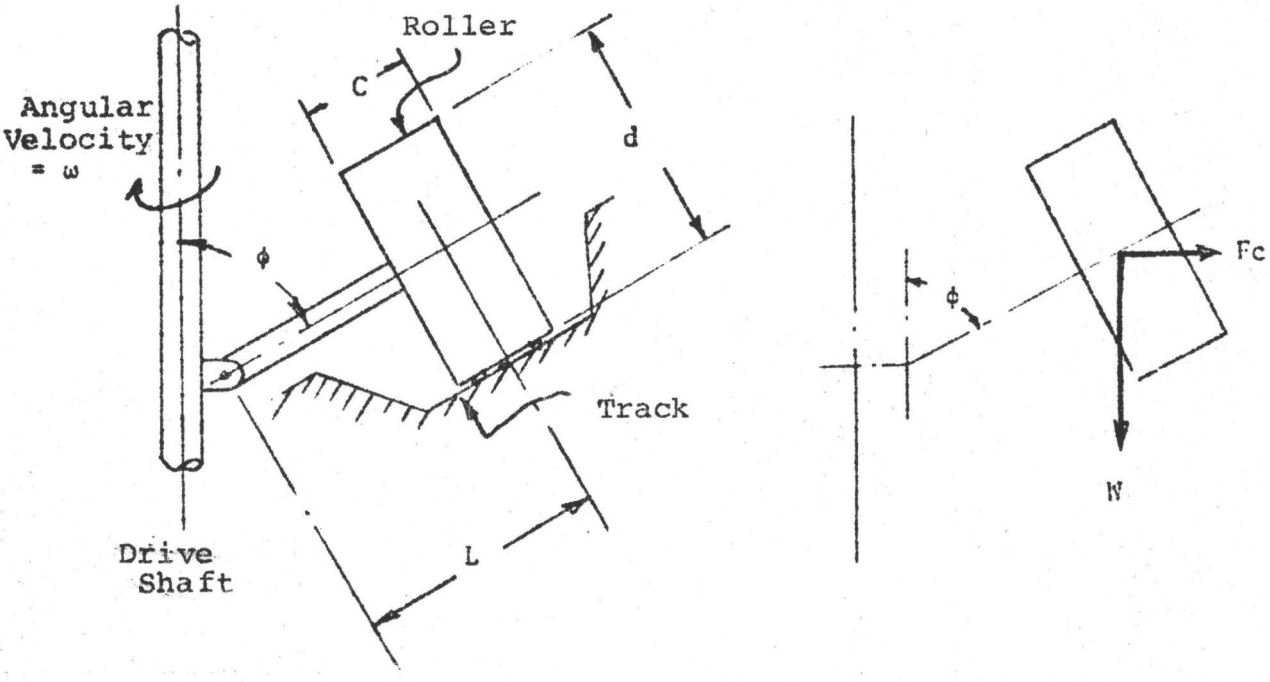
The small main program was prepared, for each method, using the documentation of OPTISEP. Following are the results.

Method	Material ($ft.^3$)	Outside dia.(ft.)	Length (ft.)	Thickness t_1 (ft.)	Thickness t_2 (ft.)	C.P. Time seconds
SIMPLEX	5.6232	3.46	6.66	.0632	.062	10.6
MEMGRAD	5.6387	3.46	6.67	.0632	.0625	61.3
DAVID	5.7022	3.46	6.79	.0631	.062	16.6

For this problem SIMPLEX has been fastest, and has also given the best solution. The dimensions of optimum pressure vessel would be as noted for SIMPLEX method.

Problem 2 - Design of a Rock Crusher

Problem is to design a rock crusher having a set of rollers rotating about a vertical axis and rolling in a track. Variables d, c, ϕ, ω and L are to be selected to yield the maximum crushing force, taking into consideration both weight and centrifugal force. The roller is steel with



BHN No. 200. One equation of constraint is provided by Hertz contact stress, which has a critical value of 70000 p.s.i. A factor of safety of 2 is to be applied. Angular velocity should be limited to a maximum of 200 rpm and arm length L to a maximum of 7 feet, length of the roller c and diameter d should not be more than 3' and 5' respectively.

Formulation

The design variables are

x1 = ω, the angular velocity

x2 = ϕ, the angle between arm and the vertical axis

x3 = c, the length of the roller

x4 = d, the diameter of the roller

x5 = L, the arm length.

The criterion for optimization is to maximize the crushing force.

density of the material is 0.282 lb/in^3.

weight of the crushing roller is

$$WT = \pi \cdot x3 \cdot x4^2 \cdot 0.282 \cdot 144 \cdot 12.0/4.0 \qquad (4.4)$$

Centrifugal force Fc is given by

$$FC = WT \cdot x1^2 \cdot x5 \cdot \sin x2/32.2 \qquad (4.5)$$

Crushing force is given by

$$FORCE = Fc \cdot \cos x2 + WT \cdot \sin x2 \qquad (4.6)$$

The objective criterion is to maximize FORCE or minimize -FORCE, therefore the optimization function is

$$U = - \text{FORCE}$$

constraints on the design are as follows -

(i) constraints to keep the design variables positive

$\phi_1 = x1$

$\phi_2 = x2$

$\phi_3 = x3$

$\phi_4 = x4$

$\phi_5 = x5$

(ii) constraint on maximum contact stress (given by Hertz relation)

$$\text{Maximum stress SMAX} = \frac{2 \cdot \text{FORCE}}{\pi \cdot B \cdot x3} \qquad (4.7)$$

$$\text{where} \quad B = \frac{4 \cdot \text{FORCE} \cdot (1-\mu^2) \cdot x4}{\pi \cdot x3 \cdot E} \qquad (4.8)$$

μ is the Poisson's ratio

E is the Young's modulus of elasticity for a factor of safety equal to 2, stress constraints

$$\phi_6 = \frac{70000}{2.0} \times 144.0 - \text{SMAX}$$

(iii) constraint on maximum angular velocity is

$$\phi_7 = \frac{2 \times 250 \times \pi}{60} - X1$$

(iv) constraints on size of roller and arm length are

$\phi_8 = 7. - x5$

$\phi_9 = 3. - x3$

$\phi_{10} = 5. - x4$

Formulation for Programming

There are five design variables

x(1) = x1, x(2) = x2, x(3) = x3, x(4) = x4, x(5) = x5

There are 10 inequality constraints

PHI(1) = ϕ_1, PHI(2) = ϕ_2,... $\phi(10) = \phi_{10}$

There is no equality constraint

U = - FORCE

The small main program was prepared, for each method, using the documentation of OPTISEP. Following are the results.

Method	X_1	X2	X_3	X4	X5	force in lb.	C.P. TIME (seconds)
MEMGRAD	11.19	0.784	2.998	4.974	4.43	$2.651*10^5$	18.362
SIMPLEX	12.24	0.68	3.00	4.999	3.80	$2.666*10^5$	42.114
DAVID	11.29	0.80	3.00	4.995	4.33	$2.662*10^5$	3.675

For this problem Davidon Fletcher and Powell's method has been the fastest. Same starting point was used for all the methods. Simplex has given the configuration, which yields maximum crushing force, though all other methods tend to converge to the same point. The best configuration for the rock crusher will have the following values of design variables.

ω - the angular velocity = 12.24 rad/sec

ϕ - the angle between vertical axis and the arm = 0.68 radius

= 39^0

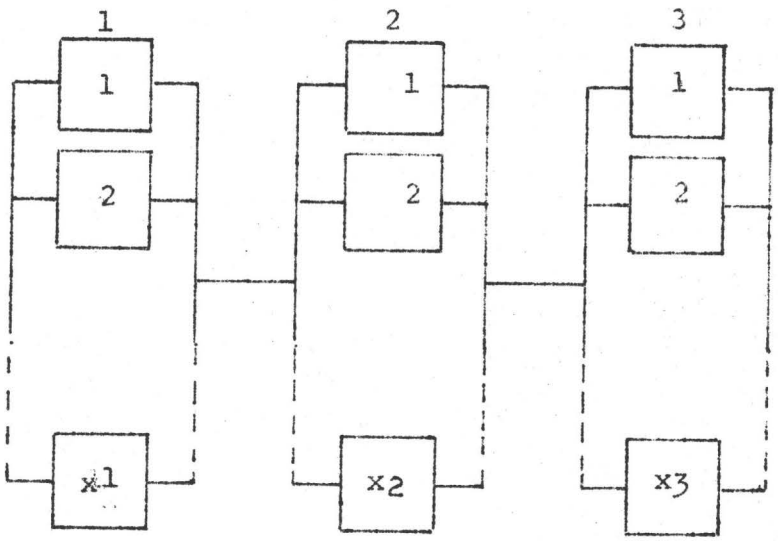
c - the length of the roller = 3'

d - the diameter of the roller = 4.999'

L - the arm length = 4.33'

Problem 3 - Optimizing reliability of a system

Components are to be connected in series parallel configuration as shown below in figure. Reliability of components in each stage is equal and equal to R_i. A return P comes only if the system does not fail. Problem is to determine optimum number of redundant components to be attached in each stage.



The reliability, cost and return data is as follows:

stage	R_i	C_i	P
1	0.333	0.200	10.0
2	0.500	1.000	
3	0.750	1.000	

This problem has been discussed by Siddall [6].

Formulation

The design variables are as follows.

x1 = total number of components in the first stage

x2 = total number of components in the second stage

x3 = total number of components in the third stage

Since the profit P only occurs if the system does not fail, therefore expected profit is P*Rd, where Rd is the reliability of the system given by

$$Rd = \prod_{i=1}^{3} [1 - (1 - R_i)^{x_i}] \tag{4.9}$$

Total cost of components is

$$C_T = \sum_{i=1}^{n} c_i x_i \tag{4.10}$$

The net profit is

$$U = P.Rd - C_T \tag{4.11}$$

The objective criterion is to maximize this U, subject to the constraints that all variables are integers and > 1. There has to be at least one component in each stage, otherwise the system would fail. The constraints to keep $x_{is} \geq 1$ can be handled by using the following transformation.

$$x_i = 1 + \text{abs}(x_i' - 1) \qquad (4.12)$$

x_i' is the new unconstrained variable.

Thus the problem is reduced to maximizing U as given by (4.11), subject to no constraints.

Formulation for programming.

There are three design variables

x(1) = x1, x(2) = x2, and x(3) = x3.

The function to be minimized is

U = - U

There are no constraints

The results obtained by subroutine INTEGER are as follows

x(1) = 7

x(2) = 3

x(3) = 2

Maximum net profit = 1.322

C.P. Time in Seconds = 7.713

Thus the optimum configuration should have seven

components in the first stage, three components in the second stage, and two components in the third stage.

Problem 4 - Optimizing a war strategy

The problem in basic form has been discussed by Bracken and McCormic [20]. The problem is to assign weapons of 2 types to 3 different targets such that total damage is maximized. The following table gives the probabilities that the targets will be undamaged by weapons, total number of weapons available, minimum number of weapons to be assigned, and military value of the target.

	Probability that weapon i will not damage target j			Number of weapons available
		Targets		
Weapon	1	2	3	
1	1.0	.95	0.85	100
2	0.84	0.98	1.00	150
Minimum number to be assigned	15	20	10	
Military Value	60	80	40	

Formulation

The design variables are as follows.

x_{11} = the number of type 1 weapon assigned to target 1

x_{12} = the number of type 1 weapon assigned to target 2

x_{13} = the number of type 1 weapon assigned to target 3

x_{21} = the number of type 2 weapon assigned to target 1

x_{22} = the number of type 2 weapon assigned to target 2

x_{23} = the number of type 2 weapon assigned to target 3

The objective function is to maximize is the total expected target damage

$$u = 60[1 - (1.00^{x_{11}} \times 0.85^{x_{21}})] + 80[1 - (.95^{x_{12}} \times .98^{x_{22}})] + 40[1 - (.85^{x_{13}} \times 1^{x_{23}})] \qquad (4.12)$$

The constraints on the total number of weapons are

$$\phi_1 = x_{11} + x_{12} + x_{13} \leq 100$$

$$\phi_2 = x_{21} + x_{22} + x_{23} \leq 150$$

The constraints on the minimum assignment of weapons are

$$\phi_3 = x_{11} + x_{21} \geq 15$$

$$\phi_4 = x_{12} + x_{22} \geq 20$$

$$\phi_5 = x_{13} + x_{23} \geq 10$$

The constraints to keep the variables positive are

$$\phi_6 = x_{11} \geq 0,\ \phi_7 = x_{12} \geq 0,\ \phi_8 = x_{13} \geq 0$$

$$\phi_9 = x_{21} \geq 0,\ \phi_{10} = x_{22} \geq 0,\ \phi_{11} = x_{23} \geq 0$$

Formulation for programming

There are six design variables.

$x(1) = x_{11}$, $x(2) = x_{12}$, $x(3) = x_{13}$

$x(4) = x_{21}$, $x(5) = x_{22}$ and $x(6) = x_{23}$

The objective function to be minimized is

$U = - u$

There are eleven constraints

$PHI(1) = 100.- (x(1) + x(2) + x(3))$

$PHI(2) = 150.- (x(4) + x(5) + x(6))$

$PHI(3) = x(1) + x(4) - 15.$

$PHI(4) = x(2) + x(5) - 20.$

$PHI(5) = x(3) + x(6) - 10.$

PHI(6) to PHI(11) = ϕ_6 to ϕ_{11} respectively.

The problem was solved by subroutine INTEGER and following are the results.

Maximum expected target damage = 179.5

Weapon	No. of weapons assigned Targets		
	1	2	3
1	0	63	37
2	42	108	0

C. P. time in seconds = 107.7.

CHAPTER - 5

USE IN DESIGN PACKAGES

In spite of the wide accessibility of computers, designers, so far, have not fully made use of them. One of the reasons for this poor response may be the non-availability of easily usable design packages, for solving design problems. Designers, being too busy with other problems, find it too time consuming to write their own programs in FORTRAN for each problem. Therefore in order that designers can use computers in a meaningful way, it is necessary to develop computer aided design packages. Such packages should be usable by any designer who has almost no knowledge of programming.

Computer aided design packages can be easily prepared using any of the optimization subroutines, developed for this thesis. The user's effort for using these packages would be even less than that required for using any of the optimization subroutines of OPTIPAC/OPTISEP. The user need only supply values of a few input parameters and call an executive subroutine to perform the necessary operations. He would not even write his own serivce subroutines to define the objective function and the constraints, because for a specific problem, constraints can be defined once for all, and made a part of the package.

The executive subroutine acts as a sort of coordinator, calling the optimization subroutine and printing out the solution which gives the optimum values of the design variables.

The availability of various simple to use optimization subroutines is of great use for the development of such packages. Experience has indicated that a given optimization method will not necessarily work with any given problem, therefore, the fact that subroutines for many optimization methods are available, significantly increases the probability that any given problem can be solved with at least one of the available subroutines.

At times it may be necessary to tune the input parameters for any optimization method to yield the best results for a given problem. While using such design packages, the user will not have to do this at all, because the best values of input parameters may be internally fixed. The designer would be ensured of the best solution on the first trial.

The various stages for the development of such design packages is as follows:

1. Identification of the problem
2. Formulation for optimization.
3. Selecting the best method of optimization.
4. Formulation of the executive subroutine.

5. Preparation of documentation.

The first stage for the development of any design package, using any of the optimization subroutines, would be to identify the problem, that is to determine what should be the design criterion, what are the limitations on the design, and what are the design variables, etc.

The second stage would be to formulate the problem for optimization, as specified in the documentation for OPTISEP/OPTIPAC. Subroutine UREAL, CONST, and EQUAL, should be written to define the problem, these subroutines should be written in a general form, so that they do not have to be changed, for different materials. Material properties may be transferred to these subroutines through common statements.

Once the problem is formulated it should be tried on all the available methods, in order to select the best one. The input parameters for the respective optimization subroutine may be tuned, if necessary. These parameters may then be used in the executive subroutine and for all practical purposes, for the designer, these parameters would be the constants internally defined.

Having selected the method to be used, the executive subroutine can be prepared. Only those input parameters which must be changed whenever a different problem is run, should be included in the arguments of the executive

subroutine. Most of the dimension statements may also be internally defined, so that the designer dimensions only a minimum of arrays in the main program.

Preparation of meaningful documentation is of great importance. Improper documentation can greatly reduce the effectiveness of a design package. Documentation should be such that anyone using it would easily follow the requirements for use by just going through it once. Documentation for such packages should include a definition of the configuration being modelled, a step-by-step procedure for using the package, a brief idea about the design procedure and the assumptions made, limitations of the design, the optimization technique used, etc. Output formats should be well written so that print out gives all necessary details for the design.

Such packages can be prepared for various designs, a few examples are as follows. Optimizing configuration of components in series and parallel for maximizing reliability of a system, designing a heat exchanger, designing a pressure vessel, designing a flywheel, designing a gear, designing a compression spring, etc. Examples are given below to illustrate how simple the calling program would be for some of the design packages explained here.

The first illustrative example is for a reliability package. If a designer has to allocate some components in a series parallel configuration, for maximizing reliability

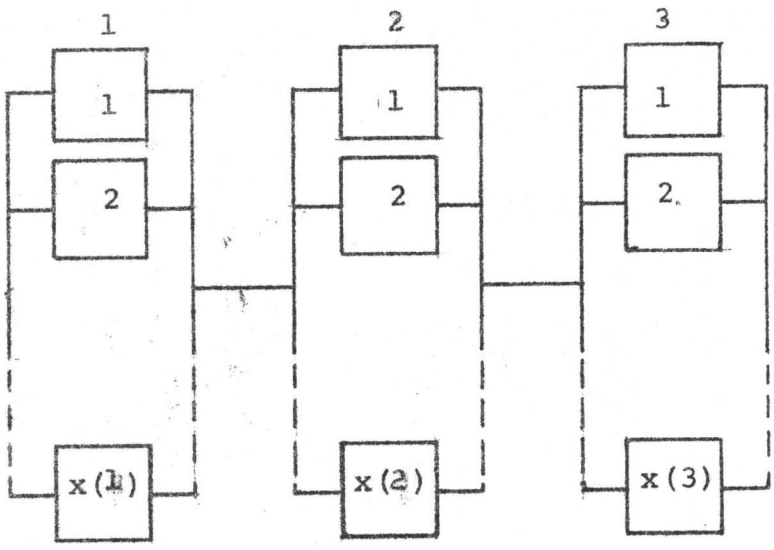
of the system, subject to a given maximum cost, he would just specify a few input parameters and call RELIAB to get the desired solution.

The second example is for a pressure vessel design package. The designer will just specify the properties of the material used, the maximum pressure in the vessel, the required capacity, the limitations on the size of the pressure vessel and would call VESSEL to get the optimum design.

The third example is for a compression spring design package. The designer will supply a few input parameters concerning the maximum load to which it will be subjected, its stiffness, limitations on size, and would call subroutine SPRING to get the desired design.

The fourth example is for a flywheel design package. The designer will specify parameters concerning material properties, performance characteristics, and Geometry.

The examples are illustrated on the following pages.



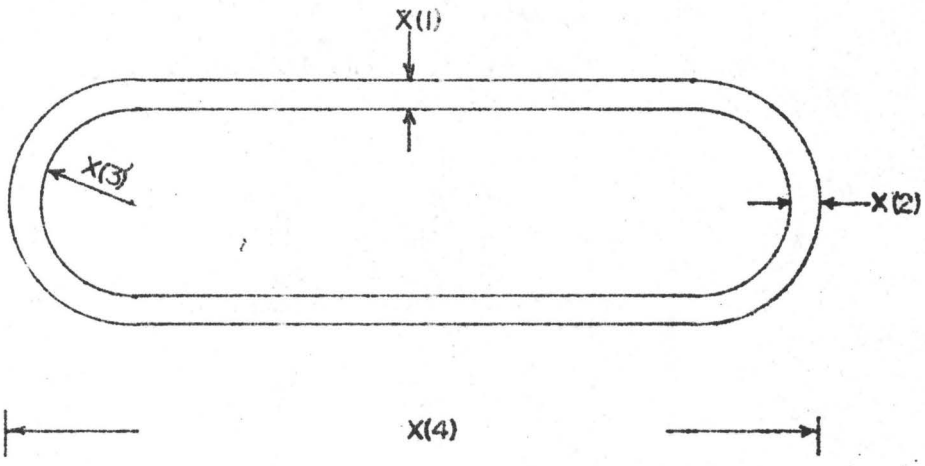
X(1) - no. of components in the first stage

X(2) - no. of components in the second stage

X(3) - no. of components in the third stage.

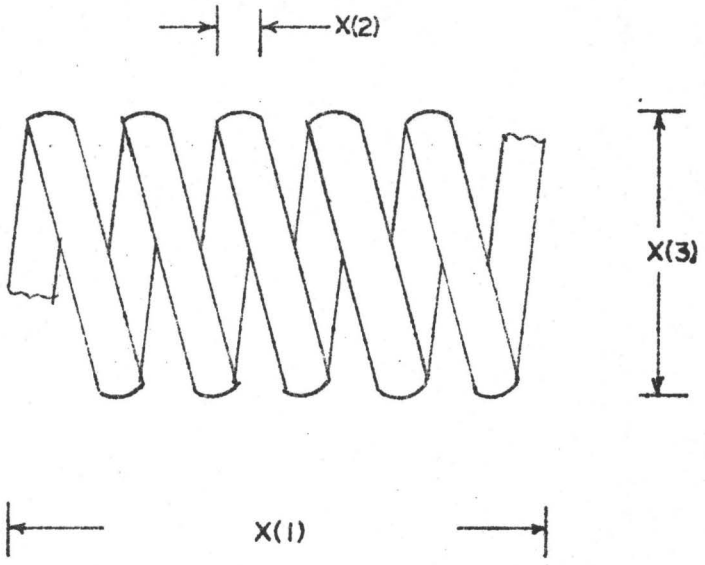
```
R1 = 0.92  (reliability of each component in first stage)
R2 = 0.95  (reliability of each component in second stage)
R3 = 0.90  (reliability of each component in third stage)
CR1 =  2.0 (cost of each component in first stage)
CR2 =  4.5 (cost of each component in second stage)
CR3=   1.0 (cost of each component in third stage)
CMAX = 20000.  (total maximum cost)
CALL RELIAB (R1, R2, R3, CR1, CR2, CR3, CMAX)
STOP
END
```

Example 1 Calling program for a reliability maximization package.



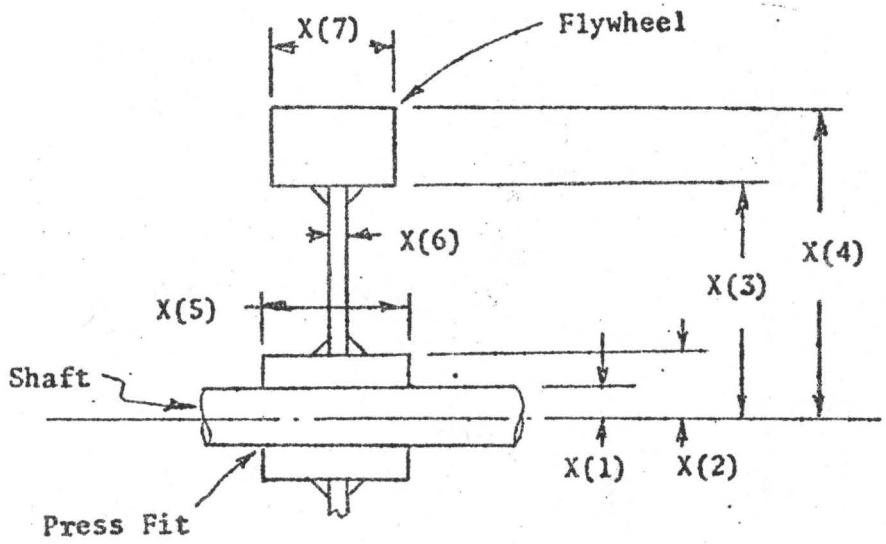

```
SMAX = 60000.
EFF1 = 0.9
PRESS = 500.
CAPATY = 2000.
AMXLN = 15.
AMNLN = 3.
CALL VESSEL (PRESS, EFF1, CAPATY, AMXLN, AMNLN, SMAX)
STOP
END
```

Example 2 Calling program for a pressure vessel design package.




```
PMAX = 50
STIF = 30
HOLE = 1.5
AMXLN = 5.
ALIFE = 5000.
CALL SPRING (PMAX, STIF, HOLE, AMXLN, ALIFE)
STOP
END
```

Example 3 Calling program for a spring design package.




```
POISS = 0.3
SYP = 70000.
E = 3.E7
FS = 3.5
SPEEDF = 3000.
TORQ = 20.
CS = .07
CU = .27
SHAFTD = 2.75
DIAFLY = 30.
CALL FLYWHL (POISS, SYP, E, FS, SPEEDF,
             TORQ, CS, CU, SHAFTD, DIAFLY)
STOP
END
```

Example 4 Calling program for a flywheel design package.

Thus using the subroutines developed for this thesis, many useful design packages can be developed. Such packages can then be stored in the library of a computer, so that designers can use them for solving design problems.

CHAPTER - 6

DISCUSSION AND CONCLUSIONS

This chapter includes the general discussion about all the methods, the various points observed during repeated use of these methods, the problems and their possible remedies, and possible changes which can be made for future development of such a package.

All the subroutines are in user oriented form. Particular attention has been given to keeping the documentation as simple as possible, and to ensuring that the user does a minimum of program writing and punching. Dimensioning of arrays, which is currently being done by the user in the main program, could be done automatically by using the dynamic storage allocation approach [21]. This is done by having a dummy blank common array say XX with a large dimension say 30000 or more. A few statements can then be added to allocate the memory to desired arrays, using values of input parameters. This approach would eliminate the dimensioning job of the user, but a few more statements would have to be added by the user in the main program. Therefore as a trade-off, to keep the size of the main program to a minimum, automatic dimensioning has not been incorporated in these subroutines. A future trial of this approach is recommended to see if it really makes it easier

for the user.

At times it is necessary to use additional data in some of the function subroutines. This data is normally transferred from the main program, to the subroutines, through common statements. It has been observed that use of blank common for this purpose invariably causes problems, because then values of data in the blank common may get mixed up with values of variables in other common statements, already existing in the programs. To avoid such a confusion it is always advisable to use labelled common statements for transferring the data.

Another source of blow ups in these programs may be an attempt made to raise a negative quantity to a fractional power, like taking square root of a negative quantity. To avoid such a problem only the absolute value of any quantity should be raised to a fractional power. Similarly SINE of a large number can cause trouble, if it so happens, IF statements may be included to prevent value of the variable from becoming too large.

At times it may be a good idea to apply weighting factors to some of the inequality constraints. When any inequality constraint is very important, or has a very small magnitude, application of a weighting factor would increase the probability that the particular inequality constraint is satisfied.

For a convergence criterion, a small quantity G has been used in all programs. Sometimes the value of the objective function itself may be so small that its order is the same as that of G, and in that case the difference between two successive values of the objective function may always be less than G, and as a result the program would indicate it as optimum, which may not be really true. Therefore when the order of the function is of the order of the convergence criterion, it is better to further reduce the value of convergence criterion. Similarly when the value of objective function is of a very large order, the two successive values of objective function may take excessive time to differ by the small amount of the order of G; in that case value of convergence criterion may be suitably increased. Generally the values of parameters recommended in the documentation should be used.

Subroutine, SIMPLEX, has been found to be very satisfactory in practice. It handles both equality and inequality constraints nicely. It may require larger number of iterations to converge, but in most of the cases it ultimately converges. Whenever this method hangs up one of the first changes the user should try is to increase the number of iterations.

The simplex size also has some effect upon the convergence of this method. It is better to have an adequately large simplex size to start with. The simplex

size is a function of F, RMAX, and RMIN. Generally adequate values are indicated in the documentation.

The parameter REDUCE also has significant effect upon convergence. The smaller the value of REDUCE, the faster is the convergence. But selection of too small a value for REDUCE would make the penalty term involving equality constraints weighted very heavily, and this would result in a very elongated and narrow valley, which would make the constrained minimization difficult. There is no formula for right choice of REDUCE. The best values based on experience have been indicated in the documentation.

Subroutine DAVID, works satisfactorily and is quite fast, but when the objective function is not well behaved and has steep valleys, this method has difficulties. Because of very steep slopes it becomes difficult to select the right step size. The penalty term incorporated in the artificial objective function for avoiding violation of any inequality constraints, results in steep gradients near the constraints. This difficulty has been resolved by properly modifying the value of derivatives near the constraints.

One of the ways of keeping the design or independent variables positive is to use transformation (2.1) where variables are forced to take absolute value whenever the value of the objective function is computed. But for, DAVID, this may cause problems, because DAVID'S logic is such that

sometimes it becomes necessary to revert back to a previous position, which is not possible in certain cases, if this approach is used. For example, if x(1) is equal to 1.5, and a step is taken of magnitude -2.0, the result would be -0.5. When the absolute value of x(1) will be returned it will be +0.5, now suppose this is not a better point and it is decided to revert back to the previous step. Under normal circumstances the previous step would be reached by just adding +2.0 to the current value of x(1), but in this case it would give the previous value of x(1) as 2.5 and not the right value which is 1.5. This may cause problems in logic of the program. To avoid this it is suggested that while using gradient methods, constraints to keep variables positive should be included in service subroutine CONST instead of using transformation (2.1).

Various search strategies are possible to determine the best value of the step size λ during any iteration. Polynomial search has been used in the program having been found faster then golden section or Fibonacci search.

DAVID has quadratic convergence. For a quadratic function it reaches the optimum in N iterations, where N is the number of variables.

One source of error in this method may be the numberical computation of derivatives. In future it would

be useful to try Powell's method [22], which does not require calculation of derivatives, since it also has quadratic convergence, and the problem of derivatives will not exist.

MEMGRAD has been found to behave similarly to DAVID. In this method the incremental step size used for computing derivatives has been found to have a significant effect upon results.

Subroutine INTEGER has been written for solving non-linear problems requiring integer optimum solutions. It has been found to work quite satisfactorily. The execution time increases in proportion to the number of variables, since the number of branches to be searched is proportional to the number of variables.

Subroutine INTEGER is informed of infeasibility by OPTIMF2, using NVIOL, the counter of violated inequality constraints. In subroutine OPTIMF2, any constraint having negative magnitude of the order of 1.E-10 or less is not considered as violated. Whenever the Hook and Jeeves [19] search fails to find the first noninteger optimum solution, it is preferable to use another method, find the optimum, and feed that optimum as starting values for subroutine INTEGER. But in some cases it has been observed that in spite of this starting point, a message of the type, "Method has failed to find non integral solution" comes out.

In this situation a nearly optimum solution has been found, but it has not been considered feasible by the logic of the program, because one or more of the constraints may have been violated, though by a very small amount, say 1.E-8, which is more than the specified value of ZERO in subroutine OPTIMF2. In such cases it is better to run the program again by slightly increasing the value of ZERO in subroutine OPTIMF2.

In subroutine INTEGER the optimum integer solution does not have exact integer values. The logic of the program treats any value, which does not differ from the integer value by more than .001, as an integer value.

If subroutine INTEGER returns a message that "No integer solution could be found", then it is better to give another trial to subroutine INTEGER, after changing the sequence of the variables. This would result in a different sequence of search and there is some possibility that some integer solution is found. In actual practice the probability is not very high that such a situation will occur.

Subroutine INTEGER has been written in such a way that with very minor modifications any other optimization technique could be used, instead of Hook and Jeeves direct search. In future, if a very good method for optimization is developed, then subroutine SOLVE can incorporate that

method, without requiring much change in subroutine INTEGER.

These subroutines have made a significant contribution to the package OPTISEP [1]. These have been tried in the past, and will be tried in future on various types of problems. More knowledge will be gained from the resulting feed back, which can then be used to further improve the package.

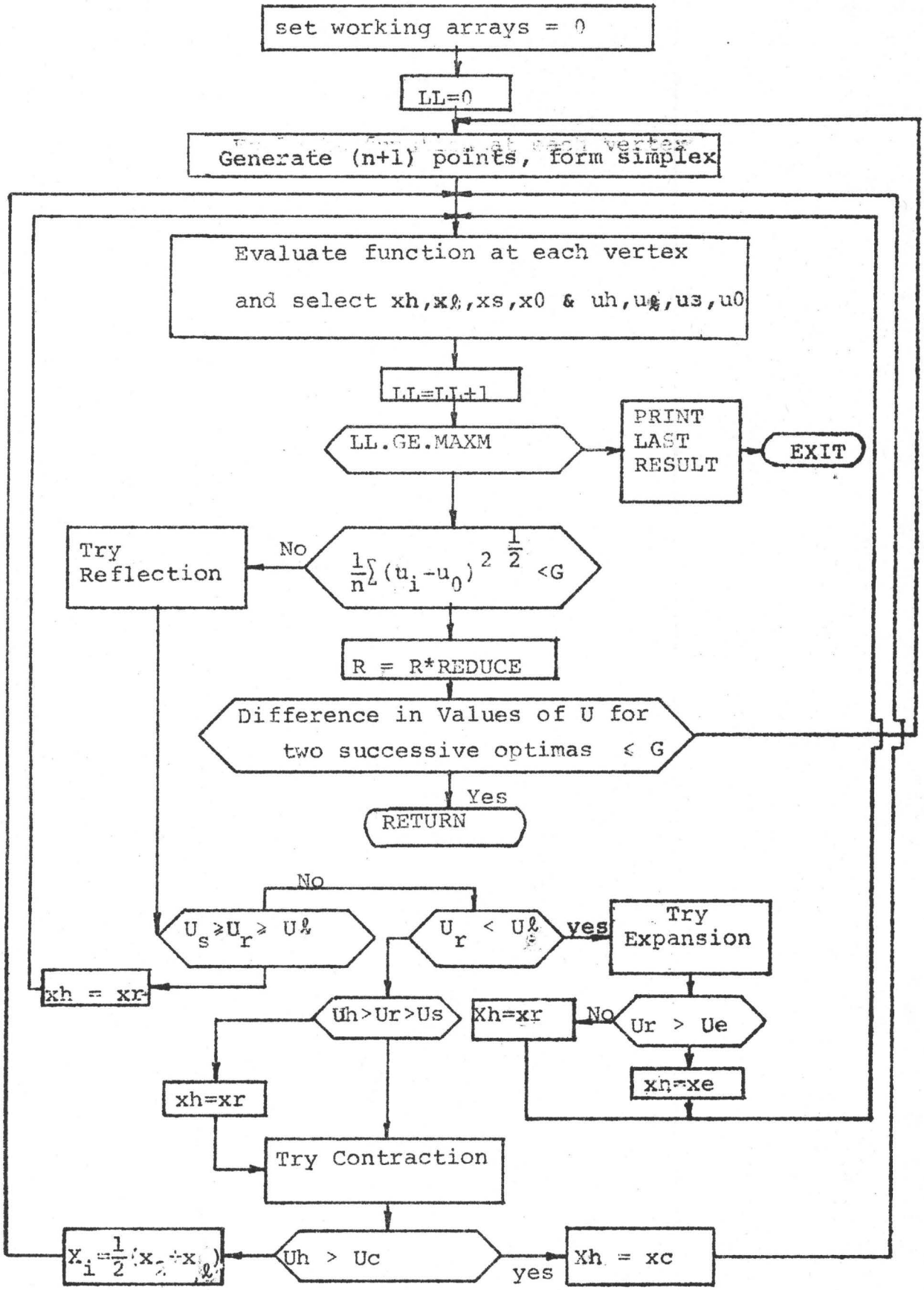
As explained in the last chapter, these subroutines can be used to develop various user oriented computer aided design packages. As illustrated in that chapter, the calling programs for such packages would be even simpler than those of OPTIPAC and OPTISEP. The engineer borrows many theoretical tools from physics, mathematics, economics, etc, and in his day to day work he must use these himself. With the development of such design packages in future, the engineer would be able to use computerized design also, as a theoretical tool in his day to day life.

ILLUSTRATIONS

```
      PROGRAM TST (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION RMAX(4),RMIN(4),XSTRT(4),X(4),PHI(11),PSI(1),XA(4,5),
     1XJ(4),XH(4),XS(4),XL(4),XO(4),XR(4),XE(4),XC(4),STEP(4),FUN(5)
      N=4
      NN=5
      IDATA=1
      NCONS=11
      F=1.0
      MAXM=300
      IPRINT=10
      G=1.E-4
      NEQUS=0
      R=1.0
      REDUCE=0.05
      ALPHA=1.0
      BETA=0.5
      GAMA=2.0
      DATA RMAX/2.0,2.0,20.,40./,RMIN/0.,0.,0.,0./,XSTRT/1.,1.,10.,20./
      CALL SIMPLEX(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,NN,ALPHA,BETA,GAMA,
     1REDUCE,R,F,G,MAXM,IPRINT,IDATA,U,X,PHI,PSI,XA,XJ,FUN,XH,XS,XL,XO,
     2XR,XE,XC,STEP)
      CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
      STOP
      END
```

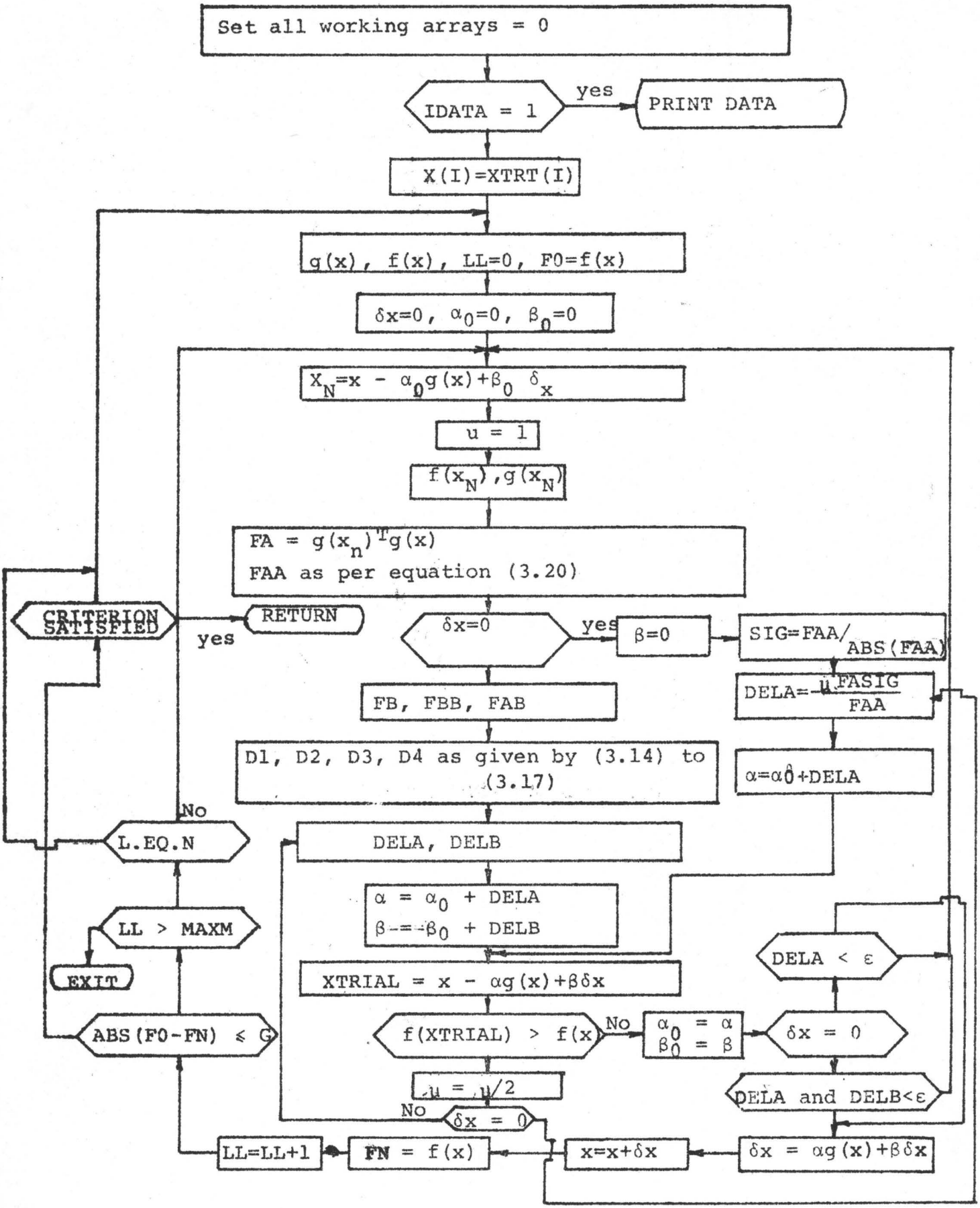
A TYPICAL CALLING PROGRAM.

Fig. 1



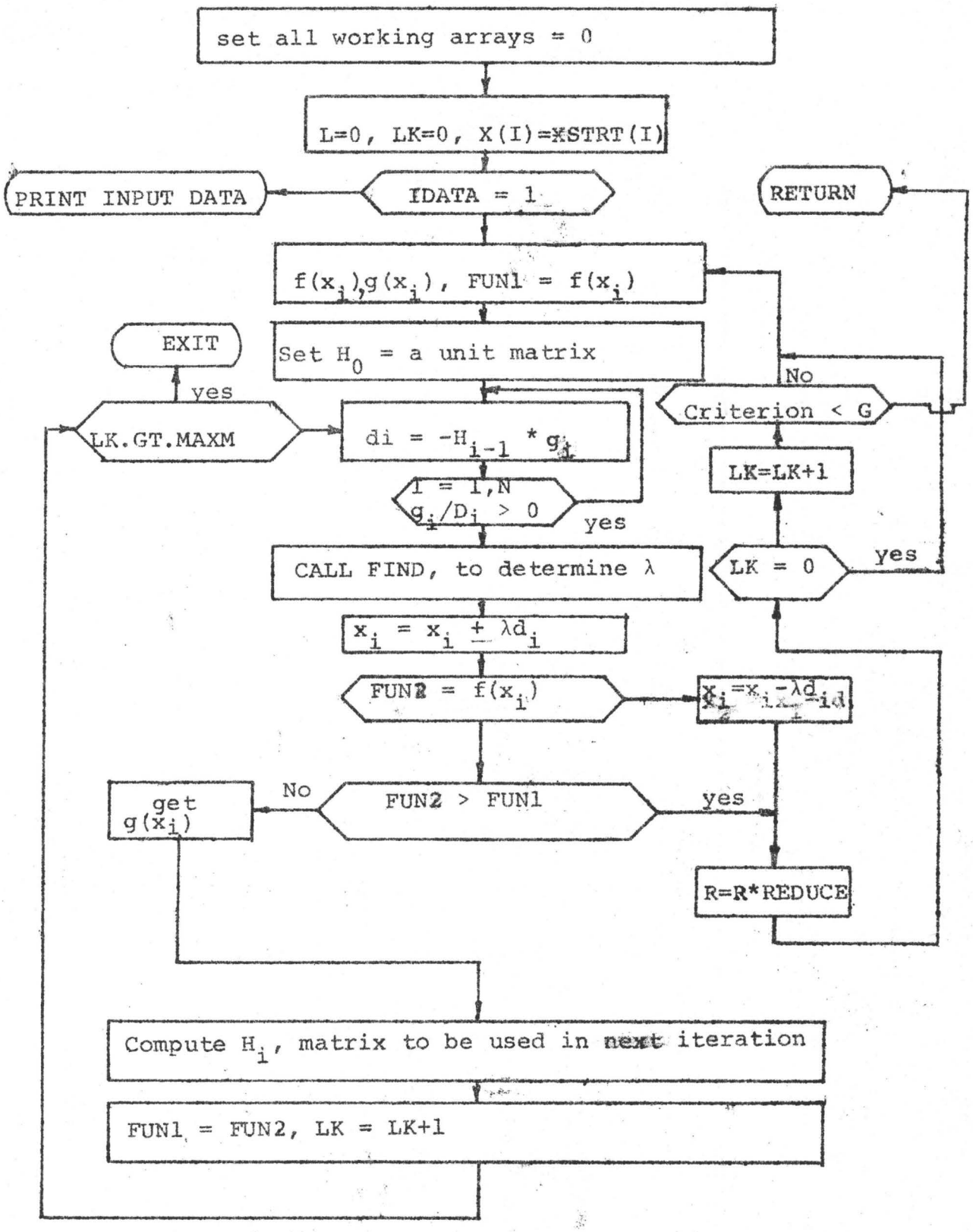
FLOW CHART "SIMPLEX'

Fig. 2



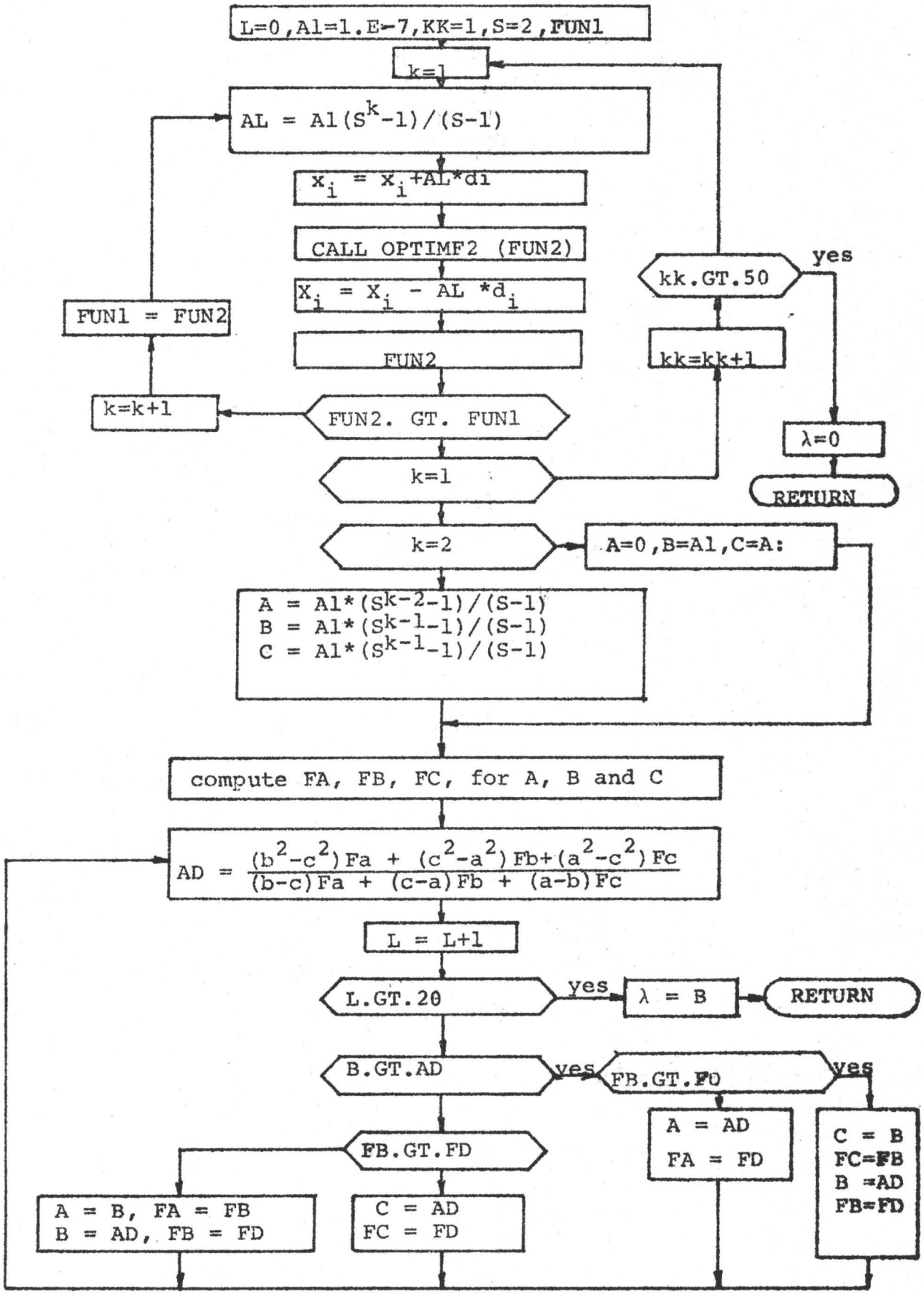
FLOW CHART 'MEMGRAD'

Fig. 3



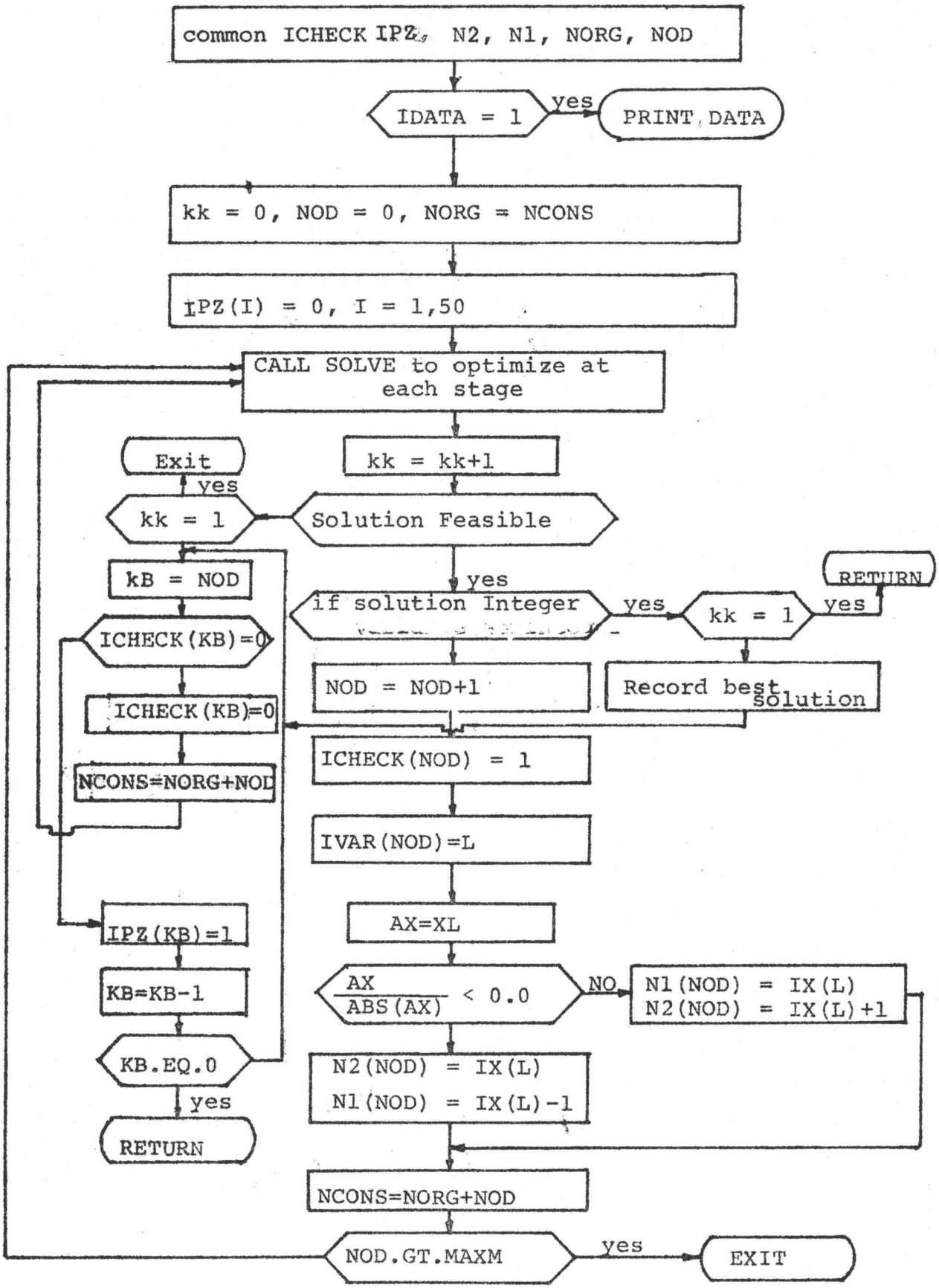
FLOW CHART SUBROUTINE 'DAVID'

FIG. 4



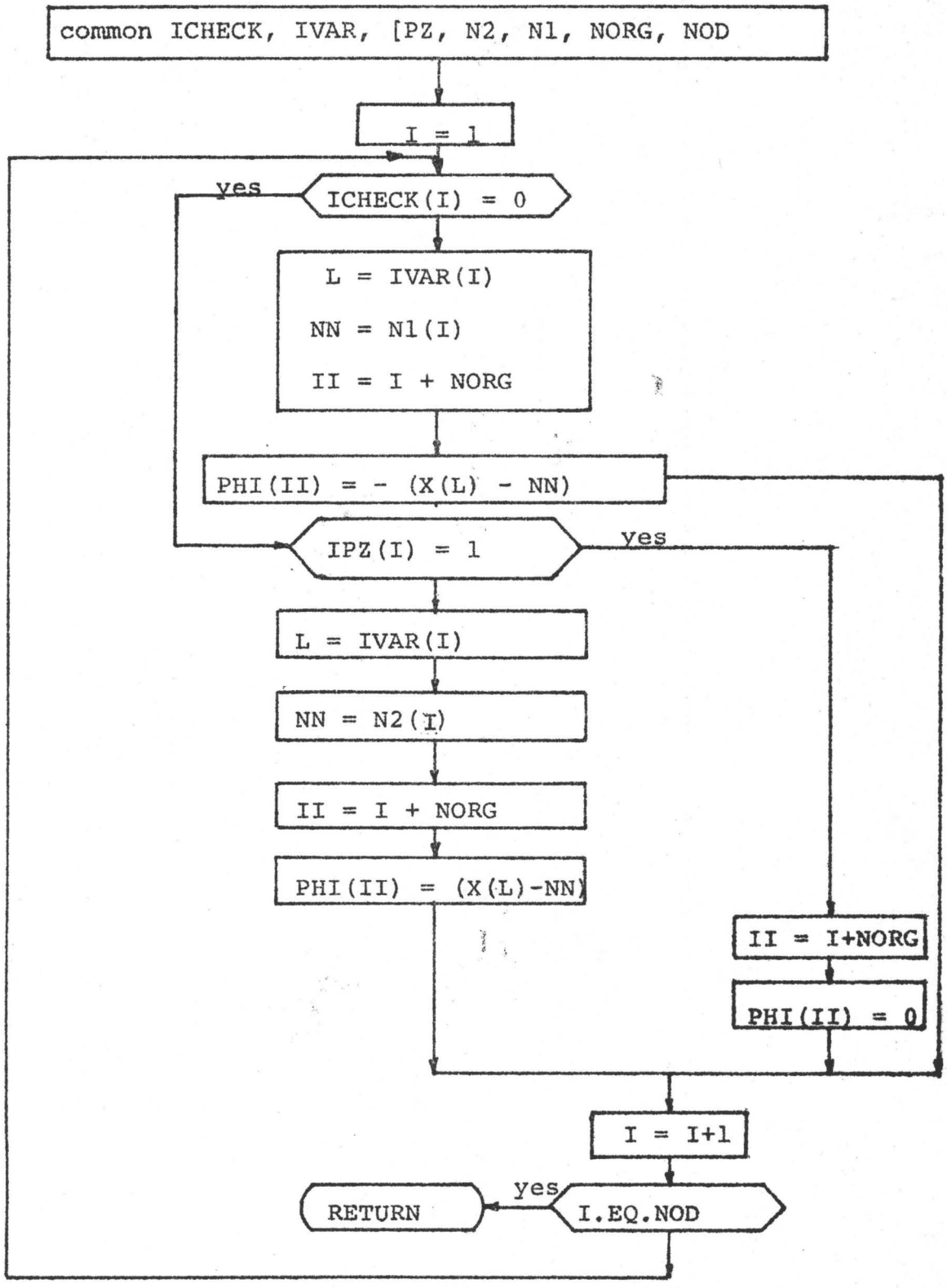
FLOW CHART FOR SUBROUTINE 'FIND'

Fig. 5



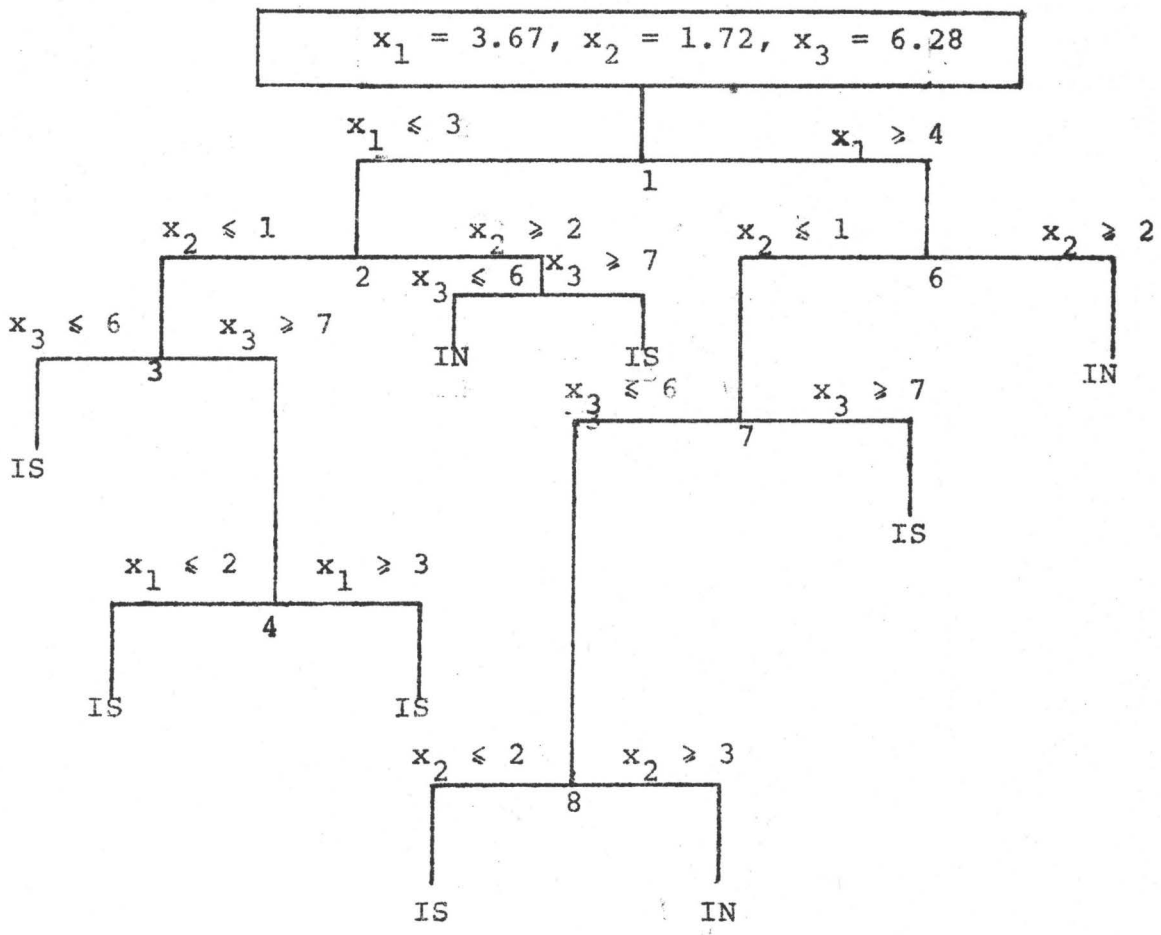
FLOW CHART SUBROUTINE 'INTEGER'

FIG. 6



FLOW CHART SUBROUTINE 'ADDL'

FIG. 7

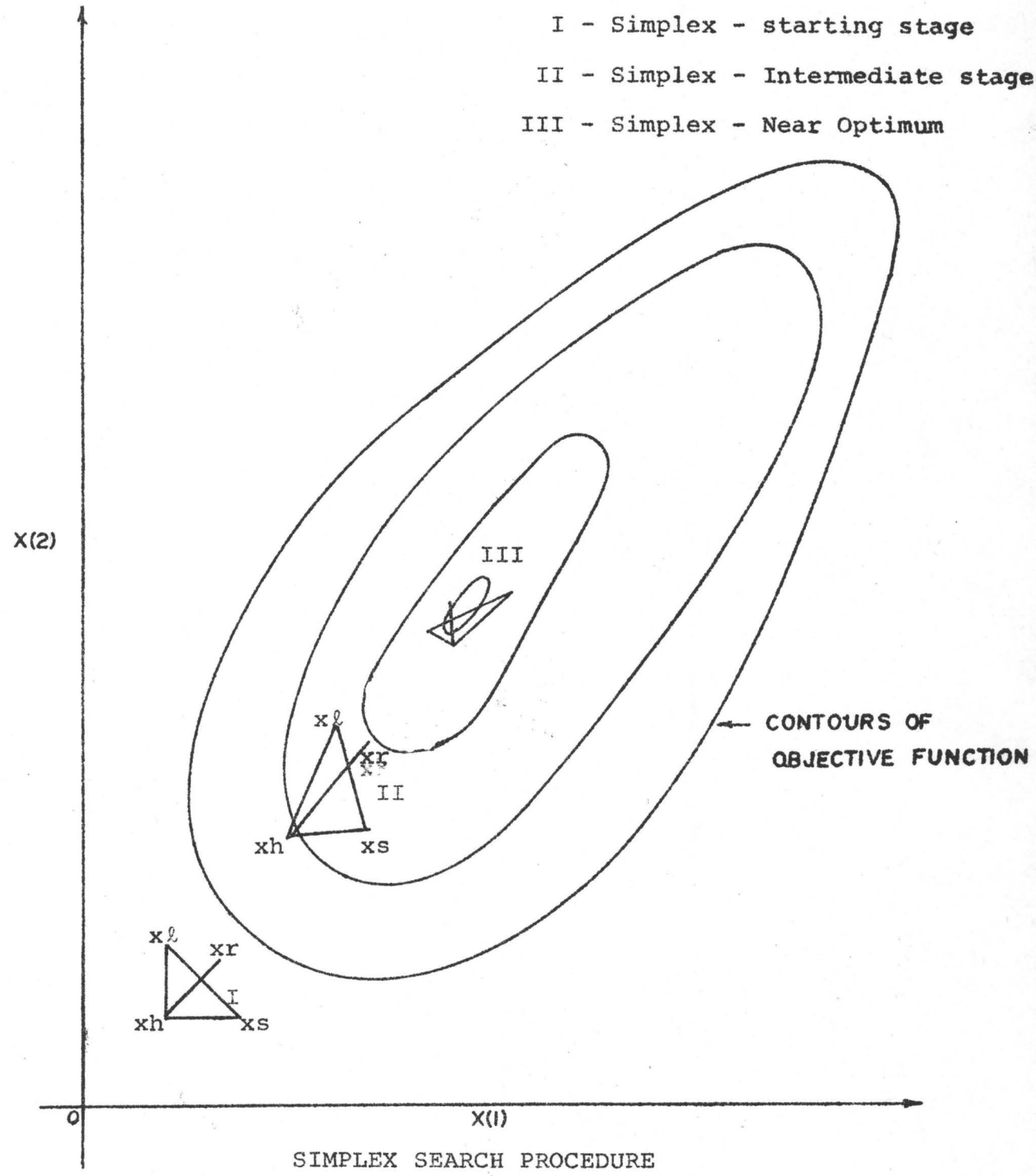


IS - Integer Solution

IN - Infeasible Solution

A TYPICAL TREE SEARCH FOR INTEGER VARIABLES

Fig. 8



SIMPLEX SEARCH PROCEDURE

Fig. 9

APPENDIX

APPENDIX A. DOCUMENTATION FOR THE SYSTEM

SUBROUTINE UREAL(X,U)

Purpose

To calculate the value of the objective function at a point

$$U = U(x_1, x_2, \ldots x_n)$$

where U = minimum at the optimum

Method

The objective function may be defined by

(a) a simple arithmetic FORTRAN statement such as

```
U = X(I)**2 + 2*SIN (X(2))
```

(b) by a complex analysis which may, for convenience, be in one or more separate subroutines. It could, for example, involve a solution of differential equations or eigenvalue equations.

Input Variables

X(I) the current values of the independent variables

Output Variables

U The value of the objective function corresponding to the input X(I) values.

How to Set Up Subroutine UREAL

The following cards must be punched by the user:

```
SUBROUTINE UREAL(X,U)
DIMENSION X(1)
U=arithmetic function
RETURN
END
```

If a more complex analysis is needed to define U, then subroutine UREAL would be punched as follows:

```
SUBROUTINE UREAL(X,U)
DIMENSION X(1)
      The coding required for analysis;  it
      may include any legal FORTRAN statements
      and CALL's to auxiliary subroutines.  The
      final value of the objective function must
      be placed in U.
RETURN
END
```

Miscellaneous

If additional data is required to perform the analysis, the necessary READ statements should be inserted in the MAIN program and the data transferred from MAIN to UREAL through labelled COMMON blocks.

Where possible, the user should include conditional STOP's in his coding to prevent invalid results from being returned to the optimization procedure.

SUBROUTINE EQUAL(X,PSI,NEQUS)

Purpose

To calculate the values of the equality constraints at a point

$$\psi_j = \psi_j(x_1, x_2, \ldots x_n) \qquad j=1,m$$

where $\psi_j=0$ at a feasible point.

Method

The equality constraint functions may be defined by:

(a) simple arithmetic FORTRAN statements such as

```
PSI(1) = X(I) + X(2)**2
```

(b) by a complex analysis which may, for convenience, be in one or more separate subroutines. It could, for example, involve a solution of differential equations or eigenvalue equations.

Input Variables

X(I)	the current values of the independent variables
NEQUS	the number of equality constraints

Output Variables

PSI(I)	the value of the equality constraints corresponding to the input X(I) values

How to Set Up Subroutine EQUAL

The following cards must be punched by the user:

```
SUBROUTINE EQUAL(X,PSI,NEQUS)
DIMENSION X(1),PSI(1)
PSI(1)=          arithmetic function
PSI(2)=          arithmetic function
....
...
PSI(NEQUS)=      arithmetic function
RETURN
END
```

If a more complex analysis is needed to define the PSI(I) values, then EQUAL would be punched as follows:

```
SUBROUTINE EQUAL(X,PSI,NEQUS)
DIMENSION X(1),PSI(1)
          The coding required for analysis;  it may
          include any legal FORTRAN statements and
          CALL's to auxiliary subroutines.  The final
          values of the constraints must be stored in
          the PSI(I) array.
RETURN
END
```

Note: If the user's problem has no equality constraints, then subroutine EQUAL may be omitted altogether.

Miscellaneous

If additional data is required to perform the analysis, the necessary READ statements should be inserted in the MAIN program and the data transferred from MAIN to EQUAL through labelled COMMON blocks.

Where possible, the user should include conditional STOP's in his coding to prevent invalid results from being returned to the optimization procedure.

SUBROUTINE CONST(X,NCONS,PHI)

Purpose

To calculate the values of the inequality constraints at a point

$$\phi_k = \phi_k(x_1, x_2, \ldots x_n) \qquad k=1,p$$

where $\phi_k \geq 0$ at a feasible point

Method

The inequality constraint functions may be defined by:

(a) simple arithmetic FORTRAN statements such as

```
PHI(I) = X(I) + X(2)**2
```

(b) by a more complex analysis which may, for convenience, be in one or more separate subroutines. It could, for example, involve a solution of differential equations or eigenvalue equations

Input Variables

X(I) the current values of the independent variables

NCONS the number of inequality constraints

Output Variables

PHI(I) the values of the inequality constraints corresponding to the input X(I) values.

How to Set Up Subroutine CONST

The following cards must be punched by the user:

```
SUBROUTINE CONST(X,NCONS,PHI)
DIMENSION X(1),PHI(1)
PHI(1)=              arithmetic function
PHI(2)=              arithmetic function
....
....
PHI(NCONS)=          arithmetic function
RETURN
END
```

If a more complex analysis is needed to define the PHI(I) values, then CONST would be punched as follows:

```
SUBROUTINE CONST(X,NCONS,PHI)
DIMENSION X(1),PHI(1)
          The coding required for analysis;  it may include
          any legal FORTRAN statements and CALL's to
          auxiliary subroutines.  The final values of the
          constraints must be stored in the PHI(I) array.
RETURN
END
```

Note: If the user's problem has no inequality constraints, then subroutine CONST may be omitted altogether.

Miscellaneous

If additional data is required to perform the analysis, the necessary READ statements should be inserted in the MAIN program and the data transferred from MAIN to CONST through labelled COMMON blocks.

Where possible, the user should include conditional STOP's in his coding to prevent invalid results from being returned to the optimization procedure.

```
SUBROUTINE SIMPLEX(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,
                   NN,ALPHA,BETA,GAMA,REDUCE,R,F,G,
                   MAXM,IPRINT,IDATA,U,X,PHI,PSI,XA,
                   XJ,FUN,XH,XS,XL,XO,XR,XE,XC,STEP)
```

Purpose

To minimize $U = U(x_1, x_2, \ldots x_n)$

subject to $\psi_j(x_1 x_2, \ldots . x_n) = 0 \qquad j = 1,m$

$\phi_k(x_1, x_2, \ldots . x_n) > 0 \qquad k = 1,p$

Method

Equality and inequality constraints are taken care of by defining an artificial unconstrained objective function,

$$P(x_1, x_2, \ldots . x_n) = U(x_1, x_2, \ldots . x_n) + r_1 \sum_{k=1}^{p} \frac{1}{\phi_k(x_1, x_2, \ldots . x_n)}$$

$$+ \sum_{j=1}^{m} \frac{\psi_j(x_1, x_2, \ldots . x_n)^2}{\sqrt{r_1}}$$

where r_1 is a positive constant (r_1=1.0 is normally assumed as starting value). Value of r is reduced by multiplying it by a constant factor 'REDUCE' after each iteration (i.e. $r_{i+1} = r_1$ x REDUCE).

The simplex method of search sets up a set of n+1 points in n-dimensional space, called the simplex. It gropes towards the optimum by flipping, expanding or contracting the simplex, the logic used depending on an evaluation of each corner.

In the logic three parameters are required -- an acceleration factor γ (γ>1), a contraction factor β (0<β<1), and a step length factor α.

The simplex is first generated by using some starting point XSTRT(I) plus n additional points

XSTRT(I) + F* (RMAX(I) - RMIN(I)

The search is considered to have found optimum if $\{\frac{1}{n}\sum_{j=1}^{n+1}(U_j-U_o)^2\}^{1/2} < G$ where G is a small quantity used as a stopping criterion.

Reference

1. Kowalik, J. and M.R. Osborne, "Methods for Unconstrained Optimization Problems", Elsevier, 1968.

Special Features

The following programming parameters must be defined by the calling program. Generally adequate values are as follows:

F = 0.1

G = 1. E-4

MAXM = 50

R = 1.0

REDUCE = 0.05

ALPHA = 1.0

BETA = 0.5

GAMA = 2.0

XSTRT(I) = (RMAX(I) + RMIN(I))/2.0

The value of F has a significant effect upon convergence. Out of various values tried F=0.1 has proved to be the best.

Simplex is a good method for problems with inequality constraints only. It tends to stall on equality constraints and it is better to start as far as possible from the equality constraint lines.

Input Variables

N	number of design or independent variables
NN = N+1	number of simplex points generated
IPRINT	prints results every IPRINTth iteration, set=0 for no intermediate output
IDATA	= 1, all input data to be printed out = 0, no input data to be printed out
NCONS	the number of inequality constraints
F	fraction of (RMAX(I) - RMIN(I)) used as step size in forming initial simplex
MAXM	maximum number of iterations allowed
G	small quantity used as convergence criterion
NEQUS	the number of equality constraints
R	penalty function parameter in calculating artificial unconstrained objective function
REDUCE	reduction factor for R
ALPHA	reflection coefficient
BETA	contraction coefficient
GAMA	expansion coefficient
RMAX(I)	estimated upper bounds on X(I), dimensioned with the value of N
RMIN(I)	estimated lower bounds on X(I) dimensioned with the value of N
XSTRT(I)	starting value for X(I) dimensioned with the value of N

Output Variables

U	minimum value of objective function, evaluated in UREAL
X(I)	optimum values of independent variables, dimensioned with value of N

PHI(I) inequality constraint functions, evaluated in CONST, dimensioned with NCONS

PSI(I) equality constraint functions, evaluated in EQUAL, dimensioned with the value of NEQUS

NVIOL counter of number of inequality constraints violated

Working Arrays

XA(I,J) dimensioned with value of N,NN

XJ(I) dimensioned with value of N

XH(I) dimensioned with value of N

XS(I) dimensioned with value of N

XL(I) dimensioned with value of N

XO(I) dimensioned with value of N

XR(I) dimensioned with value of N

XE(I) dimensioned with value of N

XC(I) dimensioned with value of N

STEP(I) dimensioned with value of N

FUN(I) dimensioned with value of NN

Programming Information

SIMPLEX has full variable dimensioning. The calling programme must provide dimensioning as given above.

If printout of the optimum is desired directly from SIMPLEX then the statement CALL SIMPLEX may be followed immediately by CALL ANSWER (U,X,PHI,PSI,N,NCONS,NEQUS). This prints out the optimum point and the values of ϕ's and ψ's.

If NCONS or NEQUS is zero then it is dimensioned 1 in the calling programme. If the method has not converged after MAXM then the current answer is printed out and SIMPLEX exits without return.

Subroutines called are OPTIMF2, CONST, EQUAL and UREAL.

```
SUBROUTINE DAVID(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,G,F,
                 MAXM,IPRINT,IDATA,R,REDUCE,U,X,PHI,
                 PSI,H,GS,D,GN,GA,Y,DT,C,YT,PHX,PSX,
                 PART,PAST,CH,UX)
```

Purpose

To minimize $U = U(x_1,x_2,\ldots,x_n)$

subject to $\phi_k(x_1,x_2,\ldots,x_n) > 0 \qquad k = 1,p$

$\psi_j(x_1,x_2,\ldots,x_n) = 0 \qquad j = 1,m$

Method

Subroutine DAVID uses the Davidon-Fletcher-Powell gradient method of search in which, at the k+1 step, the new value of an independent variable is

$$x_i^{k+1} = x_i^k + \lambda^k d_i^k$$

where λ^k defines an optimum step length and d_i^k is a function of the partial derivatives at x_i^k and all of the derivatives at the previous steps.

The search is considered to be optimum if the value of U does not change significantly in two successive steps.

Subroutine FIND is called to determine λ^k, and subroutine PARTIAL evaluates the partial derivatives by numerical calculation. Subroutine OPTIMF2 is called to form the unconstrained artificial objective function, described in SEEK3. The reader is referred to SEEK3 for a more detailed description of the use of penalty functions with successive optimization.

Reference

1. Kowalik, J., and M.R. Osborne, "Methods for Unconstrained Optimization Problems", Elsevier, 1968.

Special Features

The following program parameters must be set by the user, generally adequate values are indicated.

R = 1.0

REDUCE = 0.05

F = 1.0×10^{-6}

G = 1.0×10^{-4}

MAXM = 50

XSTRT(I) = (RMAX(I) + RMIN(I))/2.0, a known feasible start is preferable

Input Variables

N number of design or independent variables

IPRINT prints results every IPRINT step, set = 0 for no intermediate output

IDATA =1, all input data is printed out

=0, input data is not printed out

NCONS the number of inequality constraints

NEQUS the number of equality constraints

F fraction of (RMAX(I) - RMIN(I)) used as step size for computing partial derivative

R penalty function parameter

REDUCE reduction factor for R

G a small value used as convergence criterion

MAXM maximum number of iterations allowed

RMAX(I) estimated upper bound for variable X(I), dimensioned with the value of N

RMIN(I) estimated lower bound for variable X(I), dimensioned with the value of N

XSTRT(I) starting value for X(I), dimensioned with N.

Output Variables

X(I) optimum values of the independent variables, dimensioned with the value of N

U optimum value of objective function, evaluated in UREAL

PHI(I) inequality constraint function, evaluated in CONST, dimensioned with value NCONS

PSI(I) equality constraint function, evaluated in EQUAL, dimensioned with value of NEQUS

Working Arrays

H(I,J) dimensioned with value of N,N

GS(I) dimensioned with value of N

D(I) dimensioned with value of N

GN(I) dimensioned with value of N

GA(I) dimensioned with value of N

Y(I) dimensioned with value of N

DI(I,J) dimensioned with value of N,N

C(I,J) dimensioned with value of N,N

YI(I,J) dimensioned with value of N,N

PHX(I,J) dimensioned with value of (N,NCONS)

PSX(I,J) dimensioned with value of (N,NEQUS)

PART(I) dimensioned with value of N

PAST(I) dimensioned with value of N

CH(I) dimensioned with value of N

UX(I) dimensioned with value of N

Programming Information

DAVIDON has full variable dimensioning. The calling program must provide the dimensioning as given above.

If printout of the optimum is desired directly from DAVID then the statement CALL DAVID should be followed by

CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)

This prints out the optimum point and the values of the ϕ's and ψ's.

If the input value of NCONS or NEQUS is zero, it must be set at 1 in the argument of PHI,PSI,PHX and PSX in the calling programme DIMENSION statement.

If the method has not converged after MAXM iterations the current answer is printed out and DAVID exits without return. However, there is no way of knowing if DAVID has hung up on a constraint or valley and is indicating a false optimum.

Subroutines called are FIND,OPTIMF2,PARTIAL,SUPPLY,UREAL,CONST and EQUAL.

```
SUBROUTINE MEMGRAD(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,
                   F,G,MAXM,IPRINT,IDATA,R,REDUCE,
                   U,X,PHI,PSI,GO,GNEW,GA1,GA2,GB1,
                   GB2,XA,XB,XC,XD,DELX,FGA,FGAB,
                   PHX,PSX,UX,PART,PAST,CH,XNEW,XTRIAL)
```

Purpose

To minimize $U = U(x_1,x_2,\ldots,x_n)$

subject to $\psi_j = (x_1,x_2,\ldots,x_n) = 0 \qquad j = 1,m$

$\phi_k = (x_1,x_2,\ldots,x_n) \geqslant 0 \qquad k = 1,p$

Method

This method is an extension of that of Davidon, Fletcher and Powell. The step size δx_i is determined from the relation

$$\{\delta x_i\} = -\alpha\{\frac{\delta U}{\delta x_i}\} + \beta\{\delta \hat{x}_i\}$$

where α and β are scalars chosen at each iteration so as to yield the greatest decrease in the optimization function. The quantity $\delta \hat{x}_i$ is the previous step size. Thus, two parameters must be optimally chosen rather than one with Davidon, Fletcher, Powell. Selection of these parameters depends on previous gradients and steps, hence the name memory gradient.

The complete algorithm can be summarized as follows.

(1) For a given point x_i, the gradient $\frac{\delta U}{\delta x_i}$ is computed, and the vector $\delta \hat{x}_i$ is known from previous iterations. All $\delta \hat{x}_i = 0$ is assumed for the first iteration.

(2) Optimum values of the scalars α and β are found by a special search technique. The initial values are arbitrary.

The optimum is assumed to have been reached when the change in the value of U between successive steps is less than an arbitrary small quantity G.

Constraints of the problem are taken care of by forming an artificial unconstrained objective function similar to that used in SEEK3. Restarting of the alorithm beginning with $\delta\hat{x}_i = 0$ after N iterations, helps in convergence, and this has been incorporated in subroutine MEMGRAD.

References

1. Miele, A. and J.W. Cantrell,"Study on a Memory Gradient Method for the Minimization of Functions", Journal of Optimization Theory and Application, Vol.3, No.6, 1969.

Special Features

The following program parameters must be set by the user. Generally, adequate values are indicated.

R = 1.0

F = 1.0×10^{-6}

G = 1.0×10^{-4}

MAXM = 50

XSTRT(I) = (RMAX(I) + RMIN(I))/2.0, a known feasible start is preferable

REDUCE = 0.05

Input Variables

N number of design or independent variables

IPRINT prints results every IPRINT iteration, set = 0, for no intermediate output

IDATA = 1, all input data is printed out

= 0, input data is not printed out

NCONS the number of inequality constraints

NEQUS the number of equality constraints

F fraction of (RMAX(I) - RMIN(I)) used as increment for computing partial derivatives

G a small number used as a convergence criterion

R penalty function parameter

REDUCE reduction factor for R

MAXM maximum of iterations allowed

RMAX(I) upper bound for variable X(I), dimensioned with the value of N

RMIN(I) lower bound for the variable X(I), dimensioned with the value of N

XSTRT(I) starting value of X(I), dimensioned with N

Output Variables

X(I) optimum value of independent variable, dimensioned with N

U optimum value of objective function, evaluated in UREAL

PHI(I) inequality constraint function, evaluated in CONST, dimensioned with value of NCONS

PSI(I) equality constraint function, evaluated in EQUAL, dimensioned with NEQUS

Working Arrays

The following working arrays are dimensioned with the value of N.

GO,GNEW,GA1,GA2,GB1,GB2,XA,XB,XC,XD,DELX,FGA,FGB,FGAB,UX,PART,PAST, CH,XNEW,XTRIAL.

Other working arrays are dimensioned as follows

PHX dimensioned with the values of (N,NCONS)

PSX dimensioned with the values of (N,NEQUS)

Programming Information

Partial derivatives are calculated internally by numerical approximation in PARTIAL.

MEMGRAD has full variable dimensioning. The calling programme must provide the dimensioning as given above.

If printout of the optimum is desired directly from MEMGRAD, CALL MEMGRAD in the calling program should be followed by CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS). This prints out the values of ϕ's and ψ's.

However, there is no way of knowing if MEMGRAD has hung up on a constraint or valley and is indicating a false optimum.

If the input value of NCONS or NEQUS is zero, it must be set at 1 in the arguments of PHI or PSI in the calling program DIMENSION statement.

If the method does not converge after MAXM iterations, the current answer is printed out and MEMGRAD exits without return.

Subroutines called are OPTIMF2,ANSWER,PARTIAL,SUPPLY,UREAL, CONST and EQUAL.

```
SUBROUTINE INTEGER(N,RMAX,RMIN,NCONS,NEQUS,
                   XSTRT,F,G,R,REDUCE,MAXNOD,
                   MAXM,K,IPRINT,INDEX,IDATA,
                   U,X,PHI,PSI,NVIOL,WORK1,
                   WORK2,WORK3,WORK4,IX,DIF,
                   XB)
```

Purpose

To minimize $U = U(x_1, x_2, x_3, \ldots x_n)$

subject to $\phi_k(x_1, x_2, \ldots x_n) \geq 0 \quad k=1,p$

$\psi_j(x_1, x_2, \ldots x_n) = 0 \quad j=1,m$

and $(x_1, x_2, x_3 \ldots x_\ell)$ to be integers

where ℓ is such that $0<\ell\leq N$

Method

The method is based upon the branch and bound technique of integer programming. The procedure consists of a systematic search of continuous solutions in which variables to be made integer are successively forced to take integer value. If some variable say $x_s = n_s+f_s$ is to be integer, where n_s and f_s are integer and fractional part respectively, two alternative problems are formulated and solved. These can be considered as two branches coming out of a node. One contains the additional constraint $x_s \leq n_s$. The other contains the additional constraints $x_s \geq n_s + 1$. Procedure is then repeated for each of the two solutions so obtained. Search at a particular branch is terminated when either an integer solution is reached or when no feasible solution is possible. All the possible branches are searched, and the best integer solution reached this way is the optimum integer solution.

Hooke & Jeeves direct search technique is used to solve optimization problems at each stage. Constraints of the problem are taken care of by forming an artificial objective function similar to SEEK3.

Reference

1. Dakin, R.J., " A tree Search Alogarithm for Mixed Integer Programming Problems", Computer Journal, Vol. 8, April 1965 - January 1966, pp. 250-255.

2. Land, A.M. and A.G. Doig, "An Automatic Method of Solving Discrete Programming Problems", Econometrica, July 1960, Vol. 28.

Special Features

The following program parameters must be set by the user. Generally adequate values are indicated.

F = .01

MAXM = 300

MAXNOD = 25

G = .01

R = 1.0

REDUCE = .04

XSTRT(I) = (RMAX(I)+RMIN(I))/2, a known feasible start is preferable.

Input Variables

N number of design or independent variables

NCONS the number of inequality constraints

NEQUS the number of equality constraints

F	fraction of (RMAX(I)-RMIN(I)), used as initial step size
G	fraction of initial step size used as minimum step length
R	penalty function parameter
REDUCE	reduction factor for R
MAXNOD	maximum number of branches to be searched to get integer solution
MAXM	maximum number of search cycle
K	number of design variables which must be integers
IPRINT	prints result every IPRINT cycle, set at zero for no intermediate output
INDEX	set equal to 1
IDATA	= 1, all input data is printed out = 0, input data is not printed out
RMAX(I)	estimated upper bounds on X(I), dimensioned with the value of N
RMIN(I)	estimated lower bounds on X(I), dimensioned with the value of N
XSTRT(I)	starting value of X(I), dimensioned with the value of N

Output Variables

U	minimum value of the objective function, evaluated in UREAL
X(I)	optimum value of the independent variables, dimensioned with the value of N
PHI(I)	inequality constraint functions, dimensioned with the value of (NCONS+MAXNOD)

PSI(I)	equality constraint functions, dimensioned with the value of NEQUS
R	current value of penalty function multiplier
NVIOL	number of inequality constraints violated

Working Arrays

WORK1	dimensioned with value of N
WORK2	dimensioned with value of N
WORK3	dimensioned with value of N
WORK4	dimensioned with value of N
IX	dimensioned with value of K
DIF	dimensioned with value of K
XB	dimensioned with value of N

Programming Information

Subroutine INTEGER has full variable dimensioning. The calling program must provide dimensioning as above.

If printout is directly desired from INTEGER, then statement CALL INTEGER in the calling program should be followed by

```
CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
```

If search of all branches is not over after MAXNOD branches have been searched, then INTEGER exits without return, and last best solution is printed out.

If NEQUS=0, it must be set at one in the arguments of PSI, in the calling program DIMENSION statement.

If K out of N design variables are to be integers, the problem should be formulated such that variables have the following order

$$(x_1, x_2, x_3, \ldots x_k, x_{k+1}, \ldots x_n)$$

where the first k variables are to be integers.

Statement CALL ADDL(X,PHI), should always be inserted in subroutine CONST, just before RETURN statement.

If an initial continuous solution cannot be found in INTEGER, it may be possible to first obtain one by an alternate library subroutine from OPTISEP, say SIMPLEX, and begin INTEGER with this solution.

Subroutines called are SOLVE, SEARCH, ANSWER, OPTIMF2, ADDL, UREAL, CONST and EQUAL.

V. Jha

APPENDIX B. FORTRAN LISTING OF THE PROGRAMS.

```
      SUBROUTINE SIMPLEX(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,NN,ALPHA,BETA,
     1 GAMA,REDUCE,R,E,G,MAXM,IPRINT,IDATA,U,X,PHI,PSI,XA,XJ,FUN,XH,XS,
     2 XL,XO,XR,XE,XC,STEP)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),XA(N,NN),
     1 XJ(1),FUN(1),XH(1),XS(1),XL(1),XO(1),XR(1),XE(1),XC(1),STEP(1)
      COMMON KO,NNDEX
C       CLEARING ALL THE ARRAYS BEFORE USE
      DO 1 J=1,NN
      DO 1 I=1,N
      XA(I,J)=0.0
      XH(I)=0.0
      XJ(I)=0.0
      FUN(I)=0.0
      XS(I)=0.0
      XL(I)=0.0
      XO(I)=0.0
      XR(I)=0.0
      XE(I)=0.0
    1 XC(I)=0.0
      WRITE(6,301)
  301 FORMAT(15X,*OPTIMIZATION BY SIMPLEX METHOD*,/)
      KOUNT=0
      DO 2 I=1,N
      STEP(I)=E*(ABS(RMAX(I)-RMIN(I)))
    2 XA(I,1)=XSTRT(I)
      LL=0
      KKK=0
      IF(IDATA.NE.1)GO TO 299
      WRITE(6,302)IPRINT
      WRITE(6,303)IDATA
      WRITE(6,304)N
      WRITE(6,305)NCONS
      WRITE(6,306)E
      WRITE(6,307)MAXM
      WRITE(6,308)G
      WRITE(6,309)NEQUS
      WRITE(6,310)(RMAX(I),I=1,N)
      WRITE(6,311)(RMIN(I),I=1,N)
      WRITE(6,312)(XSTRT(I),I=1,N)
C       GENERATING N+1 POINTS TO FORM A SIMPLEX
  299 DO 5 J=2,NN
      DO 5 I=1,N
      IF(I.EQ.(J-1))GO TO 4
      XA(I,J)=XA(I,1)
      GO TO 5
    4 XA(I,J)=XA(I,1)+STEP(I)
    5 CONTINUE
   80 CONTINUE
      LL=LL+1
      KOUNT=KOUNT+1
C       NOW WE COMPUTE ARTIFICIAL OBJECTIVE FUNCTION AT VARIOUS POINTS
      DO 10 J=1,NN
      DO 15 I=1,N
   15 XJ(I)=XA(I,J)
      CALL OPTIM2(XJ,UART,PHI,PSI,NCONS,NEQUS,NVIOL,P)
   10 FUN(J)=UART
C       NOW WE ARRANGE FUNCTION VALUES IN ASCENDING ORDER TO SELECT HIGHEST
C       LOWEST,AND NEXT TO THE HIGHEST VALUES
      DO 20 K=1,N
      KK=K+1
      DO 20 J=KK,NN
      IF(FUN(K).LE.FUN(J)) GO TO 20
      TEMP=FUN(K)
      FUN(K)=FUN(J)
      FUN(J)=TEMP
      DO 45 I=1,N
      TEMP=XA(I,K)
      XA(I,K)=(A(I,J)
   45 XA(I,J)=TEMP
   20 CONTINUE
C       SELECTING VECTORS XH,XL,XO,XS ETC.
C       XH IS THE VECTOR GIVING MAXIMUM VALUE OF OBJECTIVE FUNCTION
C       XL IS THE VECTOR GIVING LOWEST VALUE OF THE OBJECTIVE FUNCTION
C       XO IS THE AVERAGE OF ALL POINTS OTHER THAN HIGHEST POINT XH
C       XS IS THE VECTOR WITH SECOND HIGHEST VALUE OF OBJECTIVE FUNCTION
      DO 30 I=1,N
```

```
      XH(I)=XA(I,N+1)
      XS(I)=XA(I,N)
      XL(I)=XA(I,1)
   30 XO(I)=0.0
      DO 35 I=1,N
      DO 35 J=1,N
   35 XO(I)=XO(I)+(1./FLOAT(N))*XA(I,J)
      UH=FUN(N+1)
      US=FUN(N)
      UL=FUN(1)
      CALL OPTIME2(XO,UO,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      IF(IPRINT.LE.0)GO TO 801
      IF(LL.GT.1)GO TO 804
      WRITE(6,801)
  801 FORMAT(1H1)
      WRITE(6,802)
  802 FORMAT(1HO,*INTERMEDIATE OUTPUT FOR SIMPLEX*,/)
      WRITE(6,799)
  799 FORMAT(1HO,*UART IS THE VALUE OF ARTIFICIAL UNCONSTRAINED OBJECTIV
     1E FUNCTION AT THE CENTROID*,/)
      WRITE(6,803)
  803 FORMAT(1HO,*STEP NO.          U                UART                VA
     1RIABLES X(I) AT THE CENTROID OF THE SIMPLEX*,/)
  804 IF(IPRINT.NE.KOUNT)GO TO 801
      KOUNT=0
      CALL UREAL(XO,U)
      WRITE(6,800)LL,U,UO,(XO(I),I=1,N)
  800 FORMAT(I5,3X,6E16.8,26(/,40X,4E16.8))
C     CRITERION FOR OPTIMUM
  801 USUM=0.0
      DO 300 I=1,NN
      UDIF=(FUN(I)-UO)
      UDIFSQ=UDIF*UDIF
  300 USUM=USUM+UDIFSQ
      CRIT=SQRT(USUM/FLOAT(N))
      IF(CRIT.LT.C)GO TO 400
      IF(LL.LE.MAXM)GO TO 350
C       WE TRY REFLECTION NOW
      DO 40 I=1,N
   40 XR(I)=XO(I)+ALPHA*(XO(I)-XH(I))
      CALL OPTIME2(XR,UR,PHI,PSI,NCONS,NEQUS,NVIOL,R)
C       IF US IS GREATER THAN UR AND UR GREATER THAN UL, WE REPLACE XH
C       BY XR AND RESTART FROM NEWLY FORMED SIMPLEX
      IF(US.GE.UR.AND.UR.GE.UL)GO TO 50
C       IF ABOVE CONDITION NOT MET WE TAKE NEXT STEP TO SEE IF UR IS LT UL
      GO TO 60
   50 DO 70 I=1,N
   70 XA(I,NN)= XR(I)
      GO TO 80
C       IF UR.LT.UL WE TRY EXPANSION HOPING THAT FURTHER IMPROVEMENT IS
C       POSSIBLE
   60 IF(UR.LT.UL) GO TO 90
      GO TO 100
   90 DO 110 I=1,N
  110 XE(I)=XO(I)+GAMA*(XR(I)-XO(I))
      CALL OPTIME2(XE,UE,PHI,PSI,NCONS,NEQUS,NVIOL,R)
C       IF EXPANSION IS SUCCESSFUL WE REPLACE XH BY XE OTHERWISE BY XR
      IF(UE.LT.UL) GO TO 120
      GO TO 50
  120 DO 130 I=1,N
  130 XA(I,NN)=XE(I)
      GO TO 80
C       IF UR IS GT. UH WE DONT REPLACE XH BY XR OTHERWISE WE DO
  100 IF(UR.GT.UH)GO TO 150
      IF(UH.GT.UR.AND.UR.GT.US)GO TO 155
      GO TO 225
C       CHANGE XH BY XR
  155 DO 160 I=1,N
  160 XH(I)=XR(I)
      CALL OPTIME2(XH,UH,PHI,PSI,NCONS,NEQUS,NVIOL,R)
C       WE NOW MAKE CONTRACTION MOVE
  150 DO 180 I=1,N
  180 XC(I)=XO(I)+BETA*(XH(I)-XO(I))
      CALL OPTIME2(XC,UC,PHI,PSI,NCONS,NEQUS,NVIOL,R)
```

```
C       WE CHECK IF CONTRACTION HAS BEEN SUCCESSFUL
        IF (UH.GT.UC)GO TO 200
C       IF ABOVE MOVE IS NOT SUCCESSFUL WE REPLACE ALL POINTS OF SIMPLEX
C       AND RESTART AGAIN FROM THIS CHANGED SIMPLEX
  225   DO 220 J=1,NN
        DO 220 I=1,N
  220   XA(I,J)=0.5*(XA(I,J)+XL(I))
        GO TO 80
  200   DO 210 I=1,N
        XH(I)=XC(I)
  210   XA(I,N+1)=XH(I)
        GO TO 80
C       WE CHANGE THE OPTIMUM POINT IN AN ARRAY X
  350   KO=1
        DO 500 I=1,N
  500   X(I)=XL(I)
        WRITE(6,600)MAXM
  600   FORMAT(1HO,*SIMPLEX HAS HUNG UP AFTER*,I4,*ITERATIONS*,/)
        CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
        CALL FIT
  400   CALL UREAL(XL,UNEW)
        IF(NCONS.EQ.0.AND.NEQUS.EQ.0)GO TO 402
        KKK=KKK+1
        IF(KKK.EQ.1)GO TO 401
        IF(ABS(UOLD-UNEW).LT.G)GO TO 402
  401   DO 403 I=1,N
        XA(I,1)=XL(I)
  403   CONTINUE
        R=R*REDUCE
        UOLD=UNEW
        GO TO 299
  402   KO=0
        U=UNEW
        DO 501 I=1,N
  501   X(I)=XL(I)
  302   FORMAT(61HOINTERMEDIATE OUTPUT EVERY IPRINT(TH) CYCLE. . . .  IPRI
     1NT  =,I6)
  303   FORMAT(61HOINPUT DATA IS PRINTED OUT FOR IDATA=1 ONLY. . . .  IDAT
     1A   =,I6)
  304   FORMAT(61HONUMBER OF INDEPENDENT VARIABLES . . . . . .
     1 N  =,I6)
  305   FORMAT(61HONUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . .  NCO
     1NS  =,I6)
  306   FORMAT(61HOFRACTION OF RANGE USED AS STEP SIZE . . . . . .
     1 F  =,F19.8)
  307   FORMAT(61HOMAXIMUM NUMBER OF MOVES PERMITTED . . . . . .  MA
     1XM  =,I6)
  308   FORMAT(61HOSTEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
     1 G  =,F19.8)
  309   FORMAT(61HONUMBER OF EQUALITY CONSTRAINTS. . . . . .  NEQ
     1US  =,I6)
  310   FORMAT(61HOESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . RMAX(
     1I)  =,//(5E16.8))
  311   FORMAT(61HOESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . RMIN(
     1I)  =,//(5E16.8))
  312   FORMAT(61H-STARTING VALUES OF X(I) . . . . . .XSTRT(
     1I)  =,//(5E16.8))
        RETURN
        END
```

CD TOT 0209

```
      SUBROUTINE MEMGRAD(N,PMAX,PMIN,NCONS,NEQUS,XSTRT,E,G,MAXM,IPRINT,
     1IDATA,R,REDUCE,U,X,PHI,PSI,GO,GNEW,GA1,GA2,GB1,GB2,XA,XB,XC,XD,
     2DELX,EGA,EGB,EGAB,PHX,PSX,UX,PART,PAST,CH,XNEW,XTRIAL)
      DIMENSION X(1),XSTRT(1),PMAX(1),PMIN(1),GO(1),GNEW(1),DELX(1)
     1 ,XA(1),XB(1),XC(1),XD(1),GA1(1),GA2(1),GB1(1),GB2(1),EGA(1),
     2 EGB(1),EGAB(1),XTRIAL(1),PHI(1),PSI(1),CH(1),XNEW(1),UX(1),
     3 PART(1),PAST(1),PHX(N,1),PSX(N,1)
      COMMON KO,NUDFX
C     OPTIMIZATION BY THE MEMORY GRADIENT METHOD
      LK=0
C     CLEARING ALL THE ARRAYS BEFORE USE
      DO 52 I=1,N
      GO(I)=0.0
      GNEW(I)=0.0
      XA(I)=0.0
      XB(I)=0.0
      XC(I)=0.0
      XD(I)=0.0
      GA1(I)=0.0
      GA2(I)=0.0
      GB1(I)=0.0
      GB2(I)=0.0
      XTRIAL(I)=0.0
      XNEW(I)=0.0
      EGA(I)=0.0
      EGB(I)=0.0
   52 EGAB(I)=0.0
      DO 1 I=1,N
      CH(I)=E*ABS(PMAX(I)-PMIN(I))
    1 X(I)=XSTRT(I)
      WRITE(6,301)
  301 FORMAT(1HO,*OPTIMIZATION BY MEMORY GRADIENT METHOD*,/)
C     ALL INPUT DATA IS PRINTED OUT FOR IDATA=1
      IF(IDATA.NE.1)GO TO 299
      WRITE(6,302)IPRINT
      WRITE(6,303)IDATA
      WRITE(6,304)N
      WRITE(6,305)NCONS
      WRITE(6,306)E
      WRITE(6,307)MAXM
      WRITE(6,308)G
      WRITE(6,309)NEQUS
      WRITE(6,310)(PMAX(I),I=1,N)
      WRITE(6,311)(PMIN(I),I=1,N)
      WRITE(6,312)(XSTRT(I),I=1,N)
  299 L=1
      LK=1
      KOUNT=0
  200 DO 20 I=1,N
   20 DELX(I)=0.0
      DELA=0.0
      DELB=0.0
      LL=0
      JJ=1
C     SUBROUTINE PARTIAL RETURNS VALUES OF GRADIENTS REQUIRED
  150    CALL    PARTIAL(X,N,NCONS,NEQUS,PHI,PSI,GO,R,    CH,UX,PSX,PHX,
     1 PART,PAST)
C     SUBROUTINE OPTIME2 RETURNS VALUE OF ARTIFICIAL OBJECTIVE FUNCTION
        CALL  OPTIME2(X,FUN),PHI,PSI,NCONS,NEQUS,NVIOL,R)
C     NOW WE START SEARCH FOR THE BEST VALUES OF ALPHA AND BETA
C     ALPHA AND BETA ARE THE SCALORS SO CHOSEN THAT STEP SIZE DELX =
C     -ALPHA*G+BETA(DELX OF LAST ITERATION) GIVES MAXIMUM DECREASE IN
C     THE FUNCTION VALUE
C     STARTING VALUES OF ALPHA AND BETA ARE CHOSEN AS AO=0.0 AND BO=0.0
      AO=0.0
      BO=0.0
      MMM=0
   70 DO 3 I=1,N
    3 XNEW(I)=X(I)-AO*GO(I)+BO*DELX(I)
      CALL OPTIME2(XNEW,FUNOLD,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      EMU=1.0
      CALL PARTIAL(XNEW,N,NCONS,NEQUS,PHI,PSI,GNEW, R,    CH,UX,PSX,PHX,
     1PART,PAST)
      SUM1=0.0
      DO 5 I=1,N
    5 SUM1=SUM1+GNEW(I)*GO(I)
```

```
      FA=-SUM1
      FB=0.0
      DO 6 I=1,N
    6 FB=FB+GNEW(I)*DELX(I)
      SUM1=0.0
      SUM2=0.0
      EPS=1.E-4
      DO 7 I=1,N
      SUM1=SUM1+(GO(I)**2)
    7 SUM2=SUM2+(DELX(I)**2)
      IF(JJ.EQ.1)SUM2=1.E-8
      IF(ABS(SUM1).LT.1.E-20)SUM1=1.E-20
      IF(ABS(SUM2).LT.1.E-20)SUM2=1.E-20
      EPS1=EPS/SQRT(ABS(SUM1))
      EPS2=EPS/SQRT(ABS(SUM2))
      DO 8 I=1,N
      XA(I)=XNEW(I)+EPS1*GO(I)
      XB(I)=XNEW(I)-EPS1*GO(I)
      XC(I)=XNEW(I)+EPS2*DELX(I)
    8 XD(I)=XNEW(I)-EPS2*DELX(I)
      CALL PARTIAL(XA,N,NCONS,NEQUS,PHI,PSI,GA1,R,CH,UX,PSX,PHX,
     1PART,*AST)
      CALL PARTIAL(XB,N,NCONS,NEQUS,PHI,PSI,GA2,R,CH,UX,PSX,PHX,
     1PART,PAST)
      CALL PARTIAL(XC,N,NCONS,NEQUS,PHI,PSI,GB1,R,CH,UX,PSX,PHX,
     1PART,PAST)
      CALL PARTIAL(XD,N,NCONS,NEQUS,PHI,PSI,GB2,R,CH,UX,PSX,PHX,
     1PART,PAST)
      DO 13 I=1,N
      EGA(I)=GA1(I)-GA2(I)
      EGB(I)=GB1(I)-GB2(I)
   13 EGAB(I)=GB2(I)-GB1(I)
      SUM1=0.0
      SUM2=0.0
      SUM3=0.0
      DO 14 I=1,N
      SUM1=SUM1+EGA(I)*GO(I)
      SUM2=SUM2+EGB(I)*DELX(I)
   14 SUM3=SUM3+EGAB(I)*GO(I)
      EAA=SUM1/(2.*EPS1)
      EBB=SUM2/(2.*EPS2)
      EAB=SUM3/(2.*EPS2)
      D1=EA*EBB-EB*EAB
      D2=EB*EAA-EA*EAB
      D3=EAA*EBB-EAB*EAB
      D4=EA*EA*EBB-2.*EA*EB*EAB+EB*EB*EAA
      IF(D3.EQ.0.0)D3=0.00001
      B=D4/D3
      SIGN=+1.0
      IF(B.LT.0.0)SIGN=-1.0
      NB=0
      IF(JJ.EQ.1)GO TO 51
      GO TO 60
C     FOR FIRST ITERATION DELA AND DELB ARE GIVEN BY FOLLOWING STATEMENTS
C     DELA AND DELB ARE STEPS BY WHICH VALUES OF ALPHA AND BETA ARE
C     UPDATED TILL BEST VALUES OF ALPHA AND BETA ARE FOUND
   51 SIG=1.0
      IF(EAA.LT.0.0)SIG=-1.0
      IF(ABS(EAA).LT.1.E-6)EAA=1.E-6*SIG
      AK3=1.E-4*A0
   24 DELA=-EMU*(EA/EAA)*SIG
      ALPHA=A0+DELA
      BETA=B0+DELB
      DO 21 I=1,N
   21 XTRIAL(I)=X(I)-ALPHA*GO(I)+BETA*DELX(I)
      CALL OPTIMF2(XTRIAL,FUNEW,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      IF(FUNEW.LE.FUNOLD)GO TO 22
      EMU=EMU/4.0
      NN=NN+1
      IF(NN.GT.50)GO TO 80
      GO TO 24
   22 IF(ABS(DELA).LT.ABS(AK3))GO TO 80
      A0=ALPHA
      B0=BETA
```

```
      NNN=NNN+1
      IF(NNN.GT.20 )GO TO 80
      GO TO 70
C     FOR ITERATION OTHER THAN THE FIRST FOLLOWING STATEMENTS ARE USED TO
C     COMPUTE DELA AND DELB
   60 DELA=-EMU*(D1/D3)*SIGN
      DELB=-EMU*(D2/D3)*SIGN
      AK1=(1.E-4)*A0
      AK2=(1.E-4)*B0
       ALPHA=A0+DELA
      BETA=B0+DELB
      DO 15 I=1,N
   15 XTRIAL(I)=X(I)-ALPHA*G0(I)+BETA*DELX(I)
      NN=NN+1
      CALL OPTIME2(XTRIAL,FUNEW,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      IF(FUNEW.LE.FUNOLD)GO TO 62
      EMU=EMU/4.0
      IF(NN.GT.50)GO TO 80
      GO TO 60
C     SEARCH FOR ALPHA AND BETA IS ASSUMED TO BE COMPLETE IF MAGNITUDE
C     OF DELA AND DELB BECOMES INSIGNIFICANT
   62 IF(ABS(DELA).LE.ABS(AK1).AND.ABS(DELB).LE.ABS(AK2))GO TO 80
      A0=ALPHA
      B0=BETA
      NNN=NNN+1
      IF(NNN.GT.20 )GO TO 80
      GO TO 70
C     BEST VALUES OF ALPHA AND BETA HAVE BEEN FOUND, CALCULATE STEP FOR X
   80 DO 16 I=1,N
      DELX(I)=-(ALPHA*G0(I)-BETA*DELX(I))
   16 X(I)=X(I)+DELX(I)
         CALL  OPTIME2(X,FUN2,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      KOUNT=KOUNT+1
      IF(IPRINT.LE.0)GO TO 901
      IF(L.GT.1)GO TO 804
      WRITE(6,801)
  801 FORMAT(1H1)
      WRITE(6,802)
  802 FORMAT(1HO,* INTERMEDIATE OUTPUT FOR MEMORY GRADIENT METHOD*,/)
      WRITE(6,799)
  799 FORMAT(1HO,* UART IS THE ARTIFICIAL UNCONSTRAINED OPTIMIZATION FUN
     1CTION*,/)
      WRITE(6,803)
  803 FORMAT(1HO,*STEP NO.* ,6X,*U*,12X,*UART*,30X,*INDEPENDENT VARIABLE
     1S X(I)*,/)
  804 IF(IPRINT.NE.KOUNT)GO TO 901
      KOUNT=0
      CALL UREAL(X,U)
      WRITE(6,300)L,U,FUN2,(X(I),I=1,N)
  300 FORMAT(I5,2X,6E16.8,/26(40X,4E16.8))
C     CRITERION FOR OPTIMUM
  901 IF(ABS(FUN1-FUN2).LE.G)GO TO 89
      L=L+1
C     IF SOLUTION DOES NOT CONVERGE AFTER MAXM NUMBER OF ITERATIONS THAN
C     RESULTS AT LAST ITERATION ARE PRINTED OUT
      IF(L.GT.MAXM)GO TO 100
      LL=LL+1
      JJ=JJ+1
C     PROCESS IS RESTARTED FROM THE VERY BEGINNING AFTER N ITERATIONS
      IF(LL.EQ.N)GO TO 200
      GO TO 150
  100 KO=1
      WRITE(6,600)MAXM
  600 FORMAT(1HO,*MEMGRAD HAS HUNG UP AFTER*,I4,*ITERATIONS*,/)
      CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
      CALL EXIT
   89 R=R*REDUCE
      CALL UREAL(X,UNEW)
      IF(LK.EQ.1)GO TO 88
      IF(ABS(UOLD-UNEW).LE.G)GO TO 90
   88 UOLD=UNEW
      LK=LK+1
      L=L+1
      GO TO 200
```

```
 90 KO=0
    U=UNEW
302 FORMAT(61H0INTERMEDIATE OUTPUT EVERY IPRINT(TH) CYCLE. . . .  IPRI
   1NT   =,I6)
303 FORMAT(61HOINPUT DATA IS PRINTED OUT FOR IDATA=1 ONLY. . . .  IDAT
   1A    =,I6)
304 FORMAT(61HONUMBER OF INDEPENDENT VARIABLES . . . . . .
   1 N   =,I6)
305 FORMAT(61HONUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . .  NCO
   1NS   =,I6)
306 FORMAT(61HOFRACTION OF RANGE USED AS STEP SIZE . . . . . .
   1 F   =,F10.8)
307 FORMAT(61HOMAXIMUM NUMBER OF MOVES PERMITTED . . . . . .  MA
   1XM   =,I6)
308 FORMAT(61HOSTEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
   1 G   =,F10.8)
309 FORMAT(61HONUMBER OF EQUALITY CONSTRAINTS. . . . . .  NEQ
   1US   =,I6)
310 FORMAT(61HOESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . BMAX(
   1I)   =,//(5E16.8))
311 FORMAT(61HOESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . BMIN(
   1I)   =,//(5E16.8))
312 FORMAT(61H-STARTING VALUES OF X(I) . . . . . .XSTRT(
   1I)   =,//(5E16.8))
    RETURN
    END
```

CD TOT 0250

```
      SUBROUTINE DAVID(N,RMAX,RMIN,NCONS,NEQUS,XSTRT,G,E,MAXM,IPRINT,IDA
     1TA,R,REDUCE,U,X,PHI,PSI,H,GS,D,GN,GA,Y,DT,C,YT,PHX,PSX,PART,PAST,
     2 CH,UX)
C       DAVIDON FLETCHER AND POWELL METHOD OF OPTIMIZATION
      DIMENSION X(1),RMAX(1),RMIN(1),XSTRT(1),H(N,N),GS(1),D(1),GN(1),
     1 GA(1),Y(1),DT(N,N),YT(N,N),C(N,N),PHI(1),PSI(1),PHX(N,1),PSX(N,1)
     2,PART(1),PAST(1),CH(1),UX(1)
      COMMON KO,NNDEX
C       CLEARING ALL THE ARRAYS BEFORE USE
      DO 31 I=1,N
      GS(I)=0.0
      D(I)=0.0
      GN(I)=0.0
      GA(I)=0.0
      Y(I)=0.0
      PART(I)=0.0
      PAST(I)=0.0
      CH(I)=0.0
      UX(I)=0.0
      DO 31 J=1,N
      DT(I,J)=0.0
      YT(I,J)=0.0
      C(I,J)=0.0
   31 H(I,J)=0.0
      DO 50 I=1,N
      CH(I)=E*(ABS(RMAX(I)-RMIN(I)))
   50 X(I)=XSTRT(I)
      LK=1
      L=0
      WRITE(6,301)
  301 FORMAT(1HO,*OPTIMIZATION BY DAVIDON FLETCHER AND POWELL METHOD*,/)
      KOUNT=0
      IF(IDATA.NE.1)GO TO 299
      WRITE(6,302)IPRINT
      WRITE(6,303)IDATA
      WRITE(6,304)N
      WRITE(6,305)NCONS
      WRITE(6,306)E
      WRITE(6,307)MAXM
      WRITE(6,308)G
      WRITE(6,309)NEQUS
      WRITE(6,310)(RMAX(I),I=1,N)
      WRITE(6,311)(RMIN(I),I=1,N)
      WRITE(6,312)(XSTRT(I),I=1,N)
  299 CALL OPTIM2(X,FUN1,PHI,PSI,NCONS,NEQUS,NVIOL,R)
C       SUBROUTINE PARTIAL RETURNS THE GRADIENTS REQUIRED FOR COMPUTATION
C       TO START WITH MATRIX H IS CHOSEN AS A UNIT MATRIX
        CALL   PARTIAL(X,N,NCONS,NEQUS,PHI,PSI,GS, R,   CH,UX,PSX,PHX,
     1PART,PAST)
      JJ=0
   52 DO 1 I=1,N
      DO 1 J=1,N
    1 H(I,J)=0.0
      DO 2 I=1,N
      KK=I
    2 H(I,KK)=1.0
      JJ=JJ+1
  100 DO 3 I=1,N
    3 D(I)=0.0
      DO 4 I=1,N
      DO 5 J=1,N
    5 D(I)=(H(I,J)*GS(J))+D(I)
      IF(D(I).EQ.0)D(I)=1.E-60
    4 D(I)=-D(I)
C       IF D(I) DOES NOT ENSURE THAT FUNCTION WILL DECREASE THEN RESET
C       H MATRIX AS A UNIT MATRIX
      IF(JJ.GT.1)GO TO 300
      DO 53 I=1,N
      IF((GS(I)/D(I)).GT.0.)GO TO 52
   53 CONTINUE
      JJ=0
      L=L+1
      FUNC=FUN1
C       SUBROUTINE FIND RETURNS ALMDA, WHICH GIVES OPTIMUM STEP LENGTH
      CALL FIND(X,ALMDA,D,N,PHI,PSI,NCONS,NEQUS,FUNC,R)
      DO 6 I=1,N
```

```
    6 X(I)=X(I)+ALMDA*D(I)
      CALL OPTIME2(X,FUN2,PHI,PSI,NCONS,NEQUS,NVIOL,R)
C       IF FUNCTION STARTS INCREASING  PROGRAM IS RESTARTED WITH NEW R
      KOUNT=KOUNT+1
      IF(IPRINT.LE.0)GO TO 326
      IF(L.GT.1)GO TO 804
      WRITE(6,801)
  801 FORMAT(1H1)
      WRITE(6,802)
  802 FORMAT(1HO,* INTERMEDIATE OUTPUT FOR DAVIDON FLETCHER AND POWELL*,
     1 /)
      WRITE(6,799)
  799 FORMAT(1HO,* UART IS THE ARTIFICIAL UNCONSTRAINED OPTIMIZATION FUN
     1CTION*,/)
      WRITE(6,803)
  803 FORMAT(1HO,*STEP NO.*,6X,*U*,12X,*UART*,20X,*INDEPENDENT VARIABLE
     1S X(I)*,/)
  804 IF(IPRINT.NE.KOUNT)GO TO 326
      KOUNT=0
      CALL UREAL(X,U)
      WRITE(6,327)L,U,FUN2,(X(I),I=1,N)
  327 FORMAT(I5,3X,6E16.8,/26(40X,4E16.8))
C       CRITERION FOR OPTIMUM
  326 IF(ABS(FUN1-FUN2).LE.G)GO TO 88
      IF(L.GE.MAXM)GO TO 300
      IF(FUN2.LE.FUN1)GO TO 250
      DO 240 I=1,N
  240 X(I)=X(I)-ALMDA*D(I)
      FUN2=FUN1
      GO TO 88
  250 CONTINUE
          CALL   PARTIAL(X,N,NCONS,NEQUS,PHI,PSI,GN, R,   CH,UX,PSX,PHX,
     1 PART,PAST)
C     ******************************************************************
C       THIS SECTION COMPUTES MATRIX H TO BE USED IN THE NEXT ITERATION
      DO 7 I=1,N
    7 Y(I)=GN(I)-GS(I)
      DO 8 I=1,N
    8 GA(I)=0.0
      DO 9 I=1,N
      DO 9 K=1,N
    9 GA(I)=GA(I)+(H(I,K)*GS(K))
      PROD1=0.0
      DO 10 I=1,N
   10 PROD1=PROD1+GA(I)*GS(I)
      DO 11 I=1,N
   11 GA(I)=0.0
      DO 12 I=1,N
      DO 12 K=1,N
   12 GA(I)=GA(I)+(H(I,K)*Y(K))
      PROD2=0.0
      DO 13 I=1,N
   13 PROD2=PROD2+(GA(I)*Y(I))
      DO 14 I=1,N
      DO 14 J=1,N
   14 DT(I,J)=D(I)*D(J)
      DO 15 I=1,N
      DO 15 J=1,N
   15 YT(I,J)=Y(I)*Y(J)
      DO 16 I=1,N
      DO 16 J=1,N
   16 C(I,J)=0.0
      DO 17 I=1,N
      DO 17 J=1,N
      DO 17 K=1,N
   17 C(I,J)=(H(I,K)*YT(K,J))+C(I,J)
      DO 19 I=1,N
      DO 19 J=1,N
      SUM=0.0
      DO 20 K=1,N
   20 SUM=SUM+(C(I,K)*H(K,J))
   19 C(I,J)=SUM
      IF(ABS(PROD1).LT.1.E-30)PROD1=1.E-30
      IF(ABS(PROD2).LT.1.E-30)PROD2=1.E-30
```

```
      QUO1=ALMDA/PROD1
      QUO2=1./PROD2
      DO 21 I=1,N
      DO 21 J=1,N
      DT(I,J)=DT(I,J)*QUO1
   21 C(I,J)=C(I,J)*QUO2
      DO 22 I=1,N
      DO 22 J=1,N
   22 H(I,J)=H(I,J)+DT(I,J)-C(I,J)
        ************************************************************
      DO 23 I=1,N
   23 GS(I)=XN(I)
      FUN1=FUN2
      GO TO 100
  300 KO=1
      WRITE(6,325)L
  325 FORMAT(1HO,*DAVIDON HAS HUNG UP AFTER *,I4,*ITERATIONS*,/)
      CALL ANSWER(U,X,PHI,PST,N,NCONS,NEQUS)
      CALL EXIT
   88 R=R*REDUCE
      CALL UREAL(X,UNEW)
      IF(LK.EQ.1)GO TO 88
      IF(ABS(UOLD-UNEW).LE.GIGO TO 200
   88 UOLD=UNEW
      LK=LK+1
      GO TO 299
  200 KO=0
      U=UNEW
  302 FORMAT(41HOINTERMEDIATE OUTPUT EVERY IPRINT(TH) CYCLE. . . .  IPRI
     1NT   =,I6)
  303 FORMAT(41HOINPUT DATA IS PRINTED OUT FOR IDATA=1 ONLY. . . .  IDAT
     1A    =,I6)
  304 FORMAT(41HONUMBER OF INDEPENDENT VARIABLES . . . . . .
     1 N   =,I6)
  305 FORMAT(41HONUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . .  NCO
     1NS   =,I6)
  306 FORMAT(41HOFRACTION OF RANGE USED AS STEP SIZE . . . . . .
     1 F   =,F10.8)
  307 FORMAT(41HOMAXIMUM NUMBER OF MOVES PERMITTED . . . . . .  MA
     1XM   =,I6)
  308 FORMAT(41HOSTEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
     1 G   =,F10.8)
  309 FORMAT(41HONUMBER OF EQUALITY CONSTRAINTS. . . . . .  NEQ
     1US   =,I6)
  310 FORMAT(41HOESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . RMAX(
     1I)   =,//(5E16.8))
  311 FORMAT(41HOESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . RMIN(
     1I)   =,//(5E16.8))
  312 FORMAT(41H-STARTING VALUES OF X(I) . . . . . .XSTRT(
     1I)   =,//(5E16.8))
      RETURN
      END
```

CD TOT 0202

```
      SUBROUTINE FIND(X,ALMDA,D,N,PHI,PSI,NCONS,NEQUS,FUN1,R)
      COMMON KO,NNDEX
      DIMENSION X(1),D(1),PHI(1),PSI(1)
      L=0
      A1=1.E-7
      KK=1
   61 K=1
      KK=KK+1
      IF(KK.GT.50)GO TO 48
      S=2.0
C     ****************************************************************
C     THIS SECTION FINDS BOUNDS ON THE VALUE OF ALMDA
   60 AL=A1*(((S**K)-1.)/(S-1.))
      DO 1 I=1,N
    1 X(I)=X(I)+AL*D(I)
      CALL OPTIMF2(X,FUN2,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DO 2 I=1,N
    2 X(I)=X(I)-AL*D(I)
      IF(FUN2.GT.FUN1)GO TO 10
      K=K+1
      FUN1=FUN2
      IF(K.GT.75)GO TO 40
      GO TO 50
   10 IF(K.NE.1)GO TO 9
      A1=A1/2.
      GO TO 61
    9 IF(K.EQ.2)GO TO 11
      GO TO 12
   11 A=0.0
      B=A1
      C=AL
      GO TO 13
   12 A=A1*(((S**(K-2))-1.)/(S-1.)
      B=A1*(((S**(K-1))-1.)/(S-1.)
      C=AL
   13 CONTINUE
C     ****************************************************************
C     THIS SECTION FINDS THE EXACT VALUE OF ALMDA BY POLYNOMIAL SEARCH
C     BEST VALUE OF ALMDA IS BRACKETED WITHIN A AND C
      DO 3 I=1,N
    3 X(I)=X(I)+A*D(I)
      CALL OPTIMF2(X,FA   ,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DO 4 I=1,N
    4 X(I)=X(I)-A*D(I)
      DO 5 I=1,N
    5 X(I)=X(I)+B*D(I)
      CALL OPTIMF2(X,FB   ,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DO 6 I=1,N
    6 X(I)=X(I)-B*D(I)
      DO 7 I=1,N
    7 X(I)=X(I)+C*D(I)
      CALL OPTIMF2(X,FC   ,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DO 8 I=1,N
    8 X(I)=X(I)-C*D(I)
   10 AD1=(((B*B)-(C*C))*FA)+(((C*C)-(A*A))*FB)+(((A*A)-(B*B))*FC)
      AD2=2.*(((B-C)*FA)+((C-A)*FB)+((A-B)*FC))
      IF(ABS(AD2).LT.1.E-40)GO TO 46
      AD=AD1/AD2
      GO TO 47
   46 AD=(A+B)/2.
   47 CONTINUE
      IF(AD.LT.A)AD=(A+B)/2.
      IF(AD.GT.C)AD=(B+C)/2.
      L=L+1
      IF(L.GT.10)GO TO 21
C     AD IS THE MINIMUM OF THE POLYNOMIAL PASSING THROUGH A B AND C
      DO 51 I=1,N
   51 X(I)=X(I)+AD*D(I)
      CALL OPTIMF2(X,FD   ,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DO 52 I=1,N
   52 X(I)=X(I)-AD*D(I)
      IF(B.GT.AD)GO TO 15
      IF(FB.GT.FD)GO TO 16
      C=AD
      FC=FD
      GO TO 19
```

```
   16 A=B
      FA=FB
      B=AD
      FB=FD
      GO TO 19
   15 IF(FB.GT.FD)GO TO 17
      A=AD
      FA=FD
      GO TO 19
   17 C=B
      FC=FB
      B=AD
      FB=FD
      GO TO 19
C     ***************************************************************
   21 ALMDA=B
      IF(FA.LT.FB)ALMDA=A
      GO TO 49
   48 ALMDA=0.0
      GO TO 49
   40 ALMDA=AL
   49 RETURN
      END
```

CD TOT 0099

```
      SUBROUTINE SUPPLY(X,CH,PHI,PSI,PSX,PHX,UX,          N,NCONS,NEQUS,
     1 PART,*AST)
      DIMENSION X(1),UX(1),PSX(N ,1 ),PHX(N ,1 ),PHI(1),PSI(1),
     1 PART(1),PAST(1),CH(1)
      CALL UREAL(X,UO)
      DO 10 I=1,N
      X(I)=X(I)+CH(I)
      CALL UREAL(X,U)
      X(I)=X(I)-CH(I)
   10 UX(I)=(U-UO)/CH(I)
      IF(NCONS.EQ.0)GO TO 20
      CALL CONST(X,NCONS,PART)
      DO 30 I=1,N
      X(I)=X(I)+CH(I)
      CALL CONST(X,NCONS,PHI)
      X(I)=X(I)-CH(I)
      DO 30 J=1,NCONS
   30 PHX(I,J)=(PHI(J)-PART(J))/CH(I)
   20 IF(NEQUS.EQ.0)GO TO 40
      CALL EQUAL(X,PAST,NEQUS)
      DO 60 I=1,N
      X(I)=X(I)+CH(I)
      CALL EQUAL(X,PSI,NEQUS)
      X(I)=X(I)-CH(I)
      DO 60 J=1,NEQUS
   60 PSX(I,J)=(PSI(J)-PAST(J))/CH(I)
   40 RETURN
      END
```

CD TOT 0028

```
      SUBROUTINE PARTIAL(X,N,NCONS,NEQUS,PHI,PSI,G , P,    CH,UX,PSX,PHX,
     1 PART,*AST)
      DIMENSION X(1),G(1),PHI(1),PSI(1),UX(1),PHX(N,1),PSX(N,1),CH(1),
     1 PART(1),PAST(1)
      DIV=SQRT(P)
      ZERO=-1.0E-10
      CALL SUPPLY(X,CH,PHI,PSI,PSX,PHX,UX,N,NCONS,NEQUS,PART,PAST)
      NN=0
      DO 10 I=1,N
   10 G(I)=UX(I)
      IF(NCONS.EQ.0)GO TO 1
      CALL CONST(X,NCONS,PHI)
      DO 20 I=1,N
      DO 20 J=1,NCONS
      IF(PHI(J).GT.ZERO)GO TO 21
      NN=NN+1
      G(I)=G(I)+(10.E+20)*ABS(PHX(I,J))
      GO TO 20
   21 IF(PHI(J).LT.-ZERO)GO TO 20
      G(I)=G(I)-(R*PHX(I,J)/(PHI(J)**2))
   20 CONTINUE
    1 CONTINUE
      IF(NEQUS.EQ.0)GO TO 2
      DO 40 I=1,N
      DO 50 J=1,NEQUS
   50 G(I)=G(I)+2.*(PAST(J))*PSX(I,J)/DIV
   40 CONTINUE
    2 IF(NCONS.EQ.0)GO TO 3
      IF(NN.LT.0)GO TO 3
      DO 60 I=1,N
      X(I)=X(I)+CH(I)
      CALL CONST(X,NCONS,PHI)
      X(I)=X(I)-2.*CH(I)
      CALL CONST(X,NCONS,PART)
      X(I)=X(I)+CH(I)
      DO 70 J=1,NCONS
      IF(PHI(J).GT.ZERO)GO TO 65
      GO TO 71
   65 IF(PART(J).GT.ZERO)GO TO 70
      GO TO 72
   70 CONTINUE
      GO TO 60
   71 G(I)=1.
      GO TO 60
   72 G(I)=-1.0
   60 CONTINUE
    3 RETURN
     44+
```

CD TOT 0048

```
      SUBROUTINE INTEGER (N,RMAX,RMIN,NCONS,NEQUS,XSTRT,F,G,R,REDUCE,MAX
     1NOD,MAXM,K,IPRINT,INDEX,IDATA,U,X,PHI,PSI,NVIOL,WORK1,WORK2,WORK3,
     2WORK4,IX,DIF,XR)
      DIMENSION IX(1), DIF(1), XR(1), RMAX(1), RMIN(1), XSTRT(1), PHI(1)
     1, PSI(1), WORK1(1), WORK2(1), WORK3(1), WORK4(1), X(1)
      COMMON KO,NINDEX
      COMMON /SPL/ ICHECK(50),IVAR(50),IPZ(50),N2(50),N1(50),NORG,NOD
C     THIS SUBROUTINE IS TO SOLVE OPTIMIZATION PROBLEMS WHERE SOME OR ALL
C     THE VARIABLES MUST HAVE INTEGER VALUES.
C     LOGIC OF THE PROGRAM IS BASED UPON THE BRANCH AND BOUND TECHNIQUE OF
C     INTEGER PROGRAMMING.
C     INPUT DATA IS PRINTED OUT FOR IDATA=1
      IF (IDATA.NE.1) GO TO 1
      WRITE (6,54)
      WRITE (6,45) N
      WRITE (6,44) IPRINT
      WRITE (6,43) IDATA
      WRITE (6,46) NCONS
      WRITE (6,52) NEQUS
      WRITE (6,55) K
      WRITE (6,56) MAXNOD
      WRITE (6,49) F
      WRITE (6,51) G
      WRITE (6,50) MAXM
      WRITE (6,42) R
      WRITE (6,41) REDUCE
      WRITE (6,47) (RMAX(I),I=1,N)
      WRITE (6,48) (RMIN(I),I=1,N)
      WRITE (6,53) (XSTRT(I),I=1,N)
1     CONTINUE
      KOUNT=0
      NNK=0
      KK=0
      NOD=0
      NORG=NCONS
      DO 2 I=1,50
2     IPZ(I)=0
C     HOOK AND JEEVES DIRECT SEARCH METHOD HAS BEEN USED FOR OPTIMIZATION
3     CALL SOLVE (N,RMAX,RMIN,NCONS,NEQUS,XSTRT,F,G,R,REDUCE,MAXM,IPRINT
     1,INDEX,U,X,PHI,PSI,NVIOL,WORK1,WORK2,WORK3,WORK4)
C     OPTIMUM NON INTEGRAL SOLUTION IS PRINTED OUT FIRST
      IF (NVIOL.EQ.0.AND.KK.EQ.0) GO TO 4
      GO TO 6
4     WRITE (6,64)
      WRITE (6,57)
      WRITE (6,58) U
      WRITE (6,59) (I,X(I),I=1,N)
      IF (NCONS.EQ.0) GO TO 5
      WRITE (6,60)
      WRITE (6,61) (I,PHI(I),I=1,NCONS)
5     IF (NEQUS.EQ.0) GO TO 6
      WRITE (6,62)
      WRITE (6,63) (I,PSI(I),I=1,NEQUS)
6     CONTINUE
      IF (KK.EQ.0) GO TO 8
      KOUNT=KOUNT+1
C     INTERMEDIATE OUTPUT IS PRINTED OUT EVERY IPRINTH CYCLE
      IF (IPRINT.LE.0) GO TO 8
      IF (KK.GT.1) GO TO 7
      WRITE (6,33)
      WRITE (6,34)
      WRITE (6,35)
7     IF (IPRINT.NE.KOUNT) GO TO 8
      KOUNT=0
      WRITE (6,36) KK,NOD,NVIOL,U,(X(I),I=1,N)
8     CONTINUE
      KK=KK+1
C     CHECK IF SOLUTION IS FEASIBLE
      IF (NVIOL.EQ.0) GO TO 12
      IF (KK.EQ.1) GO TO 27
9     KB=NOD
C     CHECK IF ALTERNATE CONSTRAINT AT A PARTICULAR NODE HAS BEEN ADDED
10    IF (ICHECK(KB).EQ.0) GO TO 11
      ICHECK(KB)=0
      NCONS=NORG+NOD
      R=0.001
```

```
      GO TO 3
C     CHECK IF ALL NODES HAVE BEEN SEARCHED
11    IPZ(KB)=1
      KB=KB-1
      IF (KB.EQ.0) GO TO 23
      GO TO 10
C     CHECK IF SOLUTION IS INTEGRAL
12    KM=0
      DO 13 I=1,K
C     X IS INCREASED SLIGHTLY TO GET PROPER INTEGER VALUES OF X
      X(I)=X(I)+.01
      IX(I)=IFIX(X(I))
      X(I)=X(I)-.01
13    DIF(I)=X(I)-FLOAT(IX(I))
      DO 14 I=1,K
      IF (ABS(DIF(I)).LT.1.E-2) GO TO 14
      KM=KM+1
      L=I
14    CONTINUE
      IF (KM.EQ.0) GO TO 17
C     IF SOLUTION IS NONINTEGRAL ADD CONSTRAINTS
      NOD=NOD+1
      ICHECK(NOD)=1
      IVAR(NOD)=L
      AX=X(L)
      IF ((AX/ABS(AX)).LT.0.) GO TO 15
      N1(NOD)=IX(L)
      N2(NOD)=IX(L)+1
      GO TO 16
15    N2(NOD)=IX(L)
      N1(NOD)=IX(L)-1
16    CONTINUE
      NCONS=NORG+NOD
      IF (NOD.GT.MAXNOD) GO TO 28
      R=0.001
      GO TO 3
C     IF INTEGRAL SOLUTION IS BEST SO FAR , RECORD IT
17    IF (KK.EQ.1) GO TO 22
      IF (NNK.NE.0) GO TO 19
      NNK=NNK+1
      DO 18 I=1,N
18    XB(I)=X(I)
      BEST=U
      GO TO 9
19    SNU=U
      IF (SNU.LT.BEST) GO TO 20
      GO TO 9
20    DO 21 I=1,N
21    XB(I)=X(I)
      BEST=SNU
      GO TO 9
22    KO=0
      GO TO 32
23    KO=0
      IF (NNK.EQ.0) GO TO 31
      DO 24 I=1,N
24    X(I)=XB(I)
      U=BEST
      NOD=0
      NCONS=NORG
      IF (NCONS.EQ.0) GO TO 25
      CALL CONST (X,NCONS,PHI)
25    IF (NEQUS.EQ.0) GO TO 26
      CALL EQUAL (X,PSI,NEQUS)
26    CONTINUE
      GO TO 32
27    WRITE (6,37)
      CALL EXIT
28    IF (NNK.EQ.0) GO TO 30
      WRITE (6,38)
      DO 29 I=1,N
29    X(I)=XB(I)
      NCONS=NORG
      KO=1
```

```
      NOD=0
      CALL ANSWER (U,X,PHI,PSI,N,NCONS,NEQUS)
      CALL EXIT
30    WRITE (6,39) MAXNOD
      CALL EXIT
31    WRITE (6,40)
      CALL EXIT
32    CONTINUE
      RETURN
C
33    FORMAT (1H1,*INTERMEDIATE OUTPUT FOR NONLINEAR INTEGRAL OPTIMIZATI
     1ON*,/)
34    FORMAT (1HO,*SOLUTION IS FEASIBLE ONLY IF NVIOL IS=0*,/)
35    FORMAT (1HO,*STEPNO.*,3X,*NODE NO.*,3X,*NVIOL*,6X,*U*,10X,*INDEPEN
     1DENT VARIABLES X(I)*,/)
36    FORMAT (I5,6X,I5,4X,I5,2X,6E16.8,/35(43X,5E16.8,/))
37    FORMAT (1HO,*METHOD HAS FAILED TO FIND NON INTEGRAL SOLUTION. TRY
     1 OTHER METHODS FOR NONINTEGRAL SOLUTION*,/1HO,*AND USE THAT SOLUTI
     2ON AS STARTING POINT FOR THIS METHOD*,/)
38    FORMAT (1HO,*SEARCH STOPPED AFTER MAXIMUM ALLOWABLE NUMBER OF NODE
     1S*,/1HO,*BEST INTEGRAL SOLUTION IS PRINTED OUT*,/)
39    FORMAT (1HO,*NO INTEGER SOLUTION HAS BEEN FOUND AFTER*,I5,*NODES*)
40    FORMAT (1HO,*METHOD HAS HUNG UP AND COULD NOT FIND ANY INTEGER SOL
     1UTION, TRY AGAIN BY CHANGING THE ORDER OF THE VARIABLES*)
41    FORMAT (61HOREDUCTION FACTOR FOR (R) AFTER EACH MINIMIZATION.  RED
     1UCE  =,F19.8)
42    FORMAT (61HOPENALTY MULTIPLIER USED IN SEEK3. . . . . .
     1   R  =,F19.8)
43    FORMAT (61HOINPUT DATA IS PRINTED OUT FOR IDATA=1 ONLY. . . .  ID
     1ATA  =,I6)
44    FORMAT (61HOINTERMEDIATE OUTPUT EVERY IPRINT(TH) CYCLE. . . .  IPR
     1INT  =,I7)
45    FORMAT (61HONUMBER OF INDEPENDENT VARIABLES . . . . . .
     1   N  =,I6)
46    FORMAT (61HONUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . .  NC
     1ONS  =,I6)
47    FORMAT (61HOESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . RMAX
     1(I)  =,//(5E16.8))
48    FORMAT (61HOESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . RMIN
     1(I)  =,//(5E16.8))
49    FORMAT (61HOFRACTION OF RANGE USED AS STEP SIZE . . . . . .
     1   F  =,F19.8)
50    FORMAT (61HOMAXIMUM NUMBER OF MOVES PERMITTED . . . . . .  M
     1AXM  =,I7)
51    FORMAT (61HOSTEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
     1   G  =,F19.8)
52    FORMAT (61HONUMBER OF EQUALITY CONSTRAINTS. . . . . .  NE
     1QUS  =,I7)
53    FORMAT (61H-STARTING VALUES OF X(I) . . . . . .XSTRT
     1(I)  =,//(5E16.8))
54    FORMAT (1HO,*INPUT DATA FOR NONLINEAR INTEGER OPTIMIZATION*,/)
55    FORMAT (1HO,*NUMBER OF VARIABLES TO BE MADE INTEGER. . . . . .
     1   K  =*,I6)
56    FORMAT (1HO,*MAXIMUM NUMBER OF NODES TO BE SEARCHED. . . . . MA
     1XNOD  =*,I6)
57    FORMAT (1HO,20X,*OPTIMUM NON-INTEGRAL SOLUTION*/,20X,*----------
     1------------------*,/)
58    FORMAT (20X,*  MINIMUM U=*,E16.8,//)
59    FORMAT (25X,2HX(,I2,3H) =,E16.8)
60    FORMAT (1H-,22HINEQUALITY CONSTRAINTS)
61    FORMAT (23X,4HPHI(,I2,3H) =,E16.8)
62    FORMAT (1H-,*  EQUALITY CONSTRAINTS*)
63    FORMAT (23X,4HPSI(,I2,3H) =,E16.8)
64    FORMAT (1HI)
      END
```

```
      SUBROUTINE SOLVE (N,RMAX,RMIN,NCONS,NEQUS,XSTRT,F,G,R,REDUCE,MAXM,
     1IPRINT,INDEX,U,X,PHI,PSI,NVIOL,WORK1,WORK2,WORK3,WORK4)
      DIMENSION RMAX(1), RMIN(1), XSTRT(1), X(1), PHI(1), PSI(1), WORK1(
     11), WORK2(1), WORK3(1), WORK4(1)
      COMMON KO,NNDEX
C     ZERO WORKING ARRAYS
      DO 1 I=1,N
      X(I)=0.0
      WORK1(I)=0.0
      WORK2(I)=0.0
      WORK3(I)=0.0
1     WORK4(I)=0.0
      KO=0
      ULAST=10.0E+40
C     DEFINE NNDEX=2 SO THAT SEARCH WILL FUNCTION CORRECTLY
      NNDEX=2
2     CALL SEARCH (X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,MAXM,NVIOL,
     1F,G,IPRINT,INDEX,R,WORK1,WORK2,WORK3,WORK4)
      IF (KO.NE.1) GO TO 3
      GO TO 4
3     IF (ABS(U-ULAST).GT.1.E-07*ABS(ULAST)) GO TO 5
C     OPTIMUM HAS BEEN REACHED
      RETURN
4     RETURN
5     IF (R.GT.1.0E-20) GO TO 6
      KO=1
      GO TO 4
6     ULAST=U
      R=R*REDUCE
      DO 7 I=1,N
7     XSTRT(I)=X(I)
      GO TO 2
      END
```

CP TOT 0033

```
      SUBROUTINE ADDL (X,PHI)
      COMMON /SPL/ ICHECK(50),IVAR(50),IPZ(50),N2(50),N1(50),NORG,NOD
C     ADDL RETURNS ADDITIONAL CONSTRAINTS TO MAKE SOLUTION INTEGER
      DIMENSION X(1), PHI(1)
      IF (NOD.EQ.0) GO TO 4
      DO 3 I=1,NOD
      IF (ICHECK(I).EQ.0) GO TO 1
      L=IVAR(I)
      NN=N1(I)
      II=I+NORG
      PHI(II)=-(X(L)-FLOAT(NN))*1000.
      GO TO 3
1     IF (IPZ(I).EQ.1) GO TO 2
      L=IVAR(I)
      NN=N2(I)
      II=I+NORG
      PHI(II)=(X(L)-FLOAT(NN))*1000.
      GO TO 3
2     II=I+NORG
      PHI(II)=0.
3     CONTINUE
4     RETURN
      END
```

CD TOT 0023

```
      SUBROUTINE SEARCH(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,MAXM,N
     1VIOL,F,G,IPRINT,INDEX,P,XO,XB,DXXX,TXXX)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),XO(1),XB(1),
     1DXXX(1),TXXX(1)
      COMMON KO,NNDEX
C
C     DIRECT  SEARCH  PORTION  OF  SEEK1  AND  SEEK3
C
C     THIS IS THE DIRECT SEARCH ALGORITHM OF HOOKE AND JEEVES
C     SEARCH IS USED BY SEEK1 AND SEEK3
C     NNDEX=1  MEANS SEARCH HAS BEEN CALLED BY SEEK1
C     NNDEX=2  MEANS SEARCH HAS BEEN CALLED BY SEEK3
      NVIOL1=1
      KKK=0
      M1 = 0
   20 K1=1
      K2=N
   30 DO 40    I=K1,K2
      DXXX(I)=0.
      TXXX(I)=0.
      XO(I)=0.
   40 XB(I)=0.
      DO 60   I=K1,K2
   60 X(I) = XSTRT(I)
C  SET  FIRST  BASE  POINT
      DO 70 I=K1,K2
   70 XO(I) =X(I)
C     GENERATE  DELX(I)  AND  TEST(I)
      DO 80 I=K1,K2
      DXXX(I) = F*(RMAX(I)-RMIN(I))
   80 TXXX(I)=DXXX(I)*G
      NCALL=1
  100 CONTINUE
      GO TO (101,102)NNDEX
  101 CALL OPTIME1(X,UART,PHI,PSI,NCONS,NEQUS,NVIOL)
      GO TO 110
  102 CALL OPTIME2(X,UART,PHI,PSI,NCONS,NEQUS,NVIOL,P)
  110 IF(NCALL.NE.1)GOTO   120
      UARTO = UART
  120 CONTINUE
      IF(NVIOL.EQ.0)NVIOL1=0
      IF(NNDEX.EQ.1) GO TO 130
C     INDEX=0 INDICATES TO SEARCH THAT IT IS BEING USED BY FEASRL
      IF(INDEX.EQ.1) GO TO 130
C     IF SEARCH IS BEING USED MERELY TO OBTAIN A FEASIBLE STARTING POINT
C     THEN RETURN AS SOON AS SOLUTION GOES FEASIBLE
      IF(NVIOL1.EQ.0)GO TO   385
  130 GO TO (170, 200,  210,  355) NCALL
  170 CONTINUE
C  MAKE  SEARCH
  180 NFAIL=0
      DO 240 I=K1,K2
      X(I)=X(I)+DXXX(I)
      NCALL=2
      GO TO 100
  200 CONTINUE
      IF(UART.LT.UARTO)  GOTO  230
      X(I)=X(I) - 2.0*DXXX(I)
      NCALL=3
      GO TO 100
  210 CONTINUE
      IF(UART.LT.UARTO)  )OTO  230
      NFAIL = NFAIL + 1
      X(I)=X(I)+DXXX(I)
      GOTO  240
  230 UARTO = XART
  240 CONTINUE
  250 IF(NFAIL.EQ.N)GOTO 260
      GOTO  315
  260 DO 280 I=K1,K2
      IF(DXXX(I).GT.TXXX(I)) GO TO  290
  280 CONTINUE
      GO TO 385
  290 DO 310  I=K1,K2
  310 DXXX(I)=DXXX(I)/2.
      GOTO 180
C     ESTABLISH  NEW  BASE  POINT
```

```
315 DO 320  I=K1,K2
320 XB(I) = (I)
    M1 = M1 + 1
    IF (NNDEX.EQ.1) GO TO 330
    GO TO  340
330 KKK=KKK+1
    IF(KKK.NE.IPRINT) GO TO 340
    CALL UREAL(X,ULOW)
    WRITE (6,2)  M1,ULOW , (X(I), I=1,N)
    KKK=0
340 CONTINUE
    IF(M1.GT.MAXM) GO TO 385
C   MAKE A PATTERN MOVE
    DO 350  I=K1,K2
350 X(I) = X(I) + (X(I) - XO(I))
    NCALL=4
    GO TO 100
355 CONTINUE
    IF(UART.LT.UARTO)  GOTO 370
    DO 360 I=K1,K2
    XO(I) = XB(I)
360 X(I) = XB(I)
    GOTO 180
370 DO 380  I=K1,K2
380 XO(I) = XB(I)
    UARTO = UART
    GOTO 180
385 CALL UREAL(X,U)
    GO TO(103,104)NNDEX
103 CALL OPTIME1(X,UART,PHI,PSI,NCONS,NEQUS,NVIOL)
    GO TO 105
104 CALL OPTIME2(X,UART,PHI,PSI,NCONS,NEQUS,NVIOL,R)
105 IF(NVIOL.EQ.0)GOTO387
    IF(M1.GT.MAXM)WRITE(6,4)MAXM
    KO=1
387 RETURN
  2 FORMAT(1HO,I4,3X,5E16.8/(24X,4E16.8))
  4 FORMAT(1HO,60HNO FEASIBLE SOLUTION AFTER ALLOWABLE NUMBER OF MOVES
   1, MA(M =,I6/)
    END
```

CD TOT 0117

```
      SUBROUTINE OPTIME2(X,UART,PHI,PSI,NCONS,NEQUS,NVIOL,R)
      DIMENSION X(1),PHI(1),PSI(1)
      COMMON KO,NNDEX
C     VERY MINOR VIOLATIONS OF INEQUALITY CONSTRAINTS SHOULD NOT MAKE
C     THE ENTIRE SOLUTION INFEASIBLE. THEREFORE TEST FOR PHI(I).GE.ZERO
C     WHERE ZERO=-1.0E-10
      ZERO=-1.E-10
      NVIOL=0
      SUM1=0.0
      SUM2=0.0
      CALL UREAL(X,U)
C     SEEK3  PENALTY  FUNCTIONS -
C
C     THE ARTIFICIAL OBJECTIVE FUNCTION IS OF THE FORM
C        UART=UREAL + R*SUM(1./PHI(I)) + SUM((PSI(J)**2)/SQRT(R))
C
  110 DIV=SQRT(R)
      IF(NCONS.LE.0)GOTO113
      CALL CONST(X,NCONS,PHI)
      DO 112 I=1,NCONS
      IF(PHI(I).GE.ZERO)GOTO111
      NVIOL=NVIOL+1
C     ADD A SEVERE PENALTY TO ANY PHI(I) WHICH IS VIOLATED
      SUM1=SUM1+ABS(PHI(I))*10.0E+20
      GOTO112
C     AVOID DIVIDING BY APPROXIMATELY ZERO. THERE IS NO POINT PENALIZING
C     A VERY SMALL PHI(I) ANYWAY
  111 IF(ABS(PHI(I)).LT.-ZERO)GOTO112
      SUM1=SUM1+R/ABS(PHI(I))
  112 CONTINUE
  113 IF(NEQUS.LE.0)GOTO115
      CALL EQUAL(X,PSI,NEQUS)
      DO 114 J=1,NEQUS
  114 SUM2=SUM2+(ABS(PSI(J))**2)/DIV                                    18.
  115 UART=U+SUM1+SUM2
      RETURN
      END
```

CD TOT 0037

```
      SUBROUTINE ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
      DIMENSION X(1),PHI(1),PSI(1)
      COMMON KO,NNDEX
C     THIS SUBROUTINE IS USED MERELY TO OUTPUT THE FINAL SOLUTION IN A
C     STANDARD FORM. IF AN OPTIMUM IS NOT REACHED(KO=1)THEN THE RESULTS
C     AT THE LAST ITERATION MAY BE PRINTED OUT.
      CALL UREAL(X,U)
      IF(KO.EQ.0)GOTO1
      WRITE(6,18)
      WRITE(6,19)U
      GOTO2
    1 WRITE(6,20)
      WRITE(6,21)U
    2 WRITE(6,22)(I,X(I),I=1,N)
      IF(NCONS.EQ.0)GOTO3
      CALL CONST(X,NCONS,PHI)
      WRITE(6,23)
      WRITE(6,24)(I,PHI(I),I=1,NCONS)
    3 IF(NEQUS.EQ.0)GOTO30
      CALL EQUAL(X,PSI,NEQUS)
      WRITE(6,25)
      WRITE(6,26)(I,PSI(I),I=1,NEQUS)
   18 FORMAT(1H-,16X,25HRESULTS AT LAST ITERATION,/)
   19 FORMAT(20X,3HU =,E16.8//)
   20 FORMAT(1H1,21X,22HOPTIMUM SOLUTION FOUND,/)
   21 FORMAT(20X,12HMINIMUM  U =,E16.8//)
   22 FORMAT(25X,2HX(,I2,2H) =,E16.8)
   23 FORMAT(1H-,22HINEQUALITY CONSTRAINTS)
   24 FORMAT(23X,4HPHI(,I2,2H) =,E16.8)
   25 FORMAT(1H-,22H  EQUALITY CONSTRAINTS)
   26 FORMAT(23X,4HPSI(,I2,2H) =,E16.8)
   30 RETURN
      END
```

CD TOT 0033

REFERENCES

REFERENCES

1. J. N. Siddall, 'OPTISEP' Designers Optimization Subroutines, ME/70/DSN/RP1.
2. J. N. Siddall and others, Designers Optimization Problem Solves, OPTIPAC, Faculty of Engineering, McMaster University, 1969.
3. W. Spendley, G. R. Hext, and F. R. Himsworth, Sequential Application of Simplex Designs in Optimization of Evolutionary Operations, Technometrics, 4 (1962), 441
4. J. A. Nelder and R. Mead, A Simplex Method for Function Minimization, Computer Journal 7(1965), 308
5. J. Kowalik and M. S. Osborne, Methods For Unconstrainted Optimization Problems, Elsevier New York. (1968), p. 82.
6. J. N. Siddall, Analytic Decision Making in Engineering Design.
7. H. A. Spang, A Review of Minimization Technique for Nonlinear Functions, SIAM Review, Vol. 4., No. 4, Oct. 1962, p. 343.
8. C. W. Caroll, The Created Response Technique for Optimizing Nonlinear Restrained Systems, Operation Research 9 (1961), 169.
9. A. V. Fiacco and G. P. McCormick, Non Linear Programming, John Wiley and Sons, 1968, p. 57.

10. R. Fletcher and C. M. Reeves, Function Minimization by Conjugate Gradients, Computer J, 7(1964), 149

11. W. C. Davidon, Variable Metric Method for Minimization A.E.C. Research and Development Report. 1959.

12. R. Fletcher and M. J. D. Powell, A Rapidly Convergent Descent Method for Minimization, Computer Journal 6 (1963), 163

13. A. Miele and J. W. Cantrell, Study on a Memory Gradient Method for Minimization of Functions, J. Optimization Theory and Application, Vol. 3, No. 6, (1969), 459.

14. G. B. Dantiz, D. R. Fulkerson and S. M. Johnson, Solution of Large Scale Travelling Saleman Problem, J. Operation Research Soc. America, 2, p. 393.

15. R. E. Gomory, On the Relation Between Integer and Non-Integer Solutions to Linear Programs. Proc. Nat. Acad. Sci (1965)

16. A. H. Land and A. G. Doig, An Automatic Method for Solving Descrete Programming Problems, Econometrica, Vol. 28, No. 3, (1960), 497

17. Abadie, Non Linear Programming, John Wiley and Sons (1967), p. 243.

18. R. J. Dakin, A Tree Search Algorithm for Mixed Integer Programming, Computer J., 8(1965), 250.

19. R. Hooke and T. A. Jeeves, Direct Search Solution of Numerical and Statistical Problems, J. Assoc. Compu. Math, 8 (1961), 212

20. Bracken and McCormic, Selected Applications of Non Linear Programming. John Wiley and Sons (1968)

21. Conserving Memory Space, Data Processing and Computing Centre, McMaster University, Publication 3.16, Oct. (1970).

22. M. J. D. Powell, An Efficient Method for Finding the Minimization of a Function of Several Variables Without Calculating Derivatives, Computer J., 7 (1964), 155

23. R. E. Gomory, Essential of an algorithm for integer solutions to linear programs. Bull. Amer. Math. Soc. 64 (1958).

24. R. E. Griffith and R. A. Stewart "A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems", Management Science 7(1961), pp. 379-392.