'OPTIPAC' - OPTIMIZATION IN ENGINEERING DESIGN

'OPTIPAC'

A USER-ORIENTED COMPUTER SYSTEM

FOR

OPTIMIZATION IN ENGINEERING DESIGN

By

JOHN FRANKLIN McDONALD, B.ENG.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

(May) 1970

MASTER OF ENGINEERING (1970)
(Mechanical Design)

McMASTER UNIVERSITY
Hamilton, Ontario.

TITLE: OPTIPAC: A User-Oriented Computer System for Optimization in Engineering Design

AUTHOR: John Franklin McDonald, B.Eng. (McGill University)

SUPERVISOR: Professor J.N. Siddall

NUMBER OF PAGES: v, 127

SCOPE AND CONTENTS:

A description is given of the multi-technique nonlinear optimization system called OPTIPAC.

The overall organization of the program is outlined and the significant features of each of the method subroutines are discussed. Considerable emphasis has been placed on the documentation for the system, and the two manuals which have been written are described briefly. The results of three test problems are included to demonstrate the value of having a variety of techniques in the package.

A preliminary evaluation of OPTIPAC's performance is given, with relevant suggestions for further development.

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Professor J.N. Siddall who originally conceived of this project and who was constantly available for advice and encouragement.

This work is based on a group project undertaken during the winter term of 1968-69, in Professor Siddall's graduate course "Design Optimization". The members of that class were: E.M. Alexander, M. Badiuzzaman, L. Bialy, D.J. Bonham, C.K. Cordahi, G. Cocksedge, R.H. Dumala, L. Greenberg, R.L. Hewitt, M. Kratky, J.F. McDonald, H.V. Minh, H. Moussadji, L.D. Newton, S.S. Pereira, and M.M. Win. The contribution of each is gratefully acknowledged.

The project was supported by the National Research Council, Research Grant No. A7105.

TABLE OF CONTENTS

		Page
1.	INTRODUCTION	1
2.	THE COMPUTER PROGRAM PACKAGE "OPTIPAC"	3
	2.1 General Description	3
	2.2 Service Subroutines	5
	2.3 System Subroutines	10
	2.4 Method Subroutines	17
3.	DOCUMENTATION FOR THE SYSTEM	34
	3.1 The Users' Manual	34
	3.2 The Programmers' Manual	36
4.	TEST PROBLEMS	37
5.	DISCUSSION	43
6.	CONCLUSIONS	51
	APPENDIX	
A.	SAMPLE DOCUMENTATION FOR THE SYSTEM	53
B.	RESULTS OF TEST PROBLEMS	64
C.	FORTRAN IV LISTING OF PROGRAM	66
	REFERENCES	126

LIST OF ILLUSTRATIONS

		Page
1.	Typical SEEK1 Strategy	23
2.	SEEK2 Starting Points	26
3.	Histogram of Test Results	39

TEXT

1. INTRODUCTION

The basic criterion for a successful engineering design is that it meet or surpass all restrictions imposed upon it by the design specifications themselves, the laws of physics and chemistry, and the properties of the materials used. A design which satisfies all these requirements is called an acceptable, or feasible solution to the problem. In practice, nearly all design problems have several feasible solutions, and the final configuration must be chosen according to some other criterion such as minimum weight, maximum volume, or minimum cost. This part of the design procedure is known as optimization.

Before the introduction of high speed digital computers, very little systematic optimization was done because of the prohibitive amount of time necessary to determine even a few feasible solutions. Although several computer techniques have now been developed, formal optimization in engineering design is still not widely used and there appear to be two main reasons for this. First of all, few engineers have either the time or computer programming knowledge to write their own optimization algorithms. Programs which are available in computer libraries are usually inflexible and difficult for an inexperienced programmer to use. Secondly, only for purely linear problems,* is there a general method (revised Simplex[1]) which can guarantee that the optimum

---

*In optimization theory the terms "linear" and "nonlinear" refer to the forms of the constraint equations and inequalities, and the optimization function which define the particular problem.

found is the global or absolute optimum. Unfortunately, most real problems are nonlinear and the relative success of any one of the nonlinear techniques is largely dependent upon the form of the functions describing the problem. It is rarely possible to predict which method is best suited to a particular problem. To overcome all these difficulties it was felt that the designer needed a pre-written program package containing several different optimization techniques, with input requirements kept to a minimum. In addition, the program would need thorough documentation written in a straightforward, "how-to-do-it" style.

A system of this type has been developed by the author and others who are credited in the "Acknowledgements". The package is called OPTIPAC and it contains eight nonlinear optimization methods and a code for revised Simplex. Input/output is controlled internally and the user needs only a basic understanding of simple FORTRAN. Step-by-step instructions on how to run a problem are contained in a users' manual,[2] while a second manual[3] provides detailed information about the actual program organization and logic.

This thesis describes the significant features of OPTIPAC and makes suggestions for its further development. The results of some test problems are discussed and a complete FORTRAN listing of the program is included in the Appendix C to provide a permanent record of the version of OPTIPAC which is described here. The users' and programmers' manuals[2,3] are frequently referred to as they contain a thorough description of every facet of the system's design and operation.

2. THE COMPUTER PROGRAM PACKAGE "OPTIPAC"

2.1 General Description

The program is written in FORTRAN IV and is organized into a series of subroutines which fall into three basic categories: service subroutines, system subroutines and method subroutines.

The service subroutines are written by the user to define the objective function and constraints for his problem. These, along with a program MAIN and some data cards, comprise the user's input deck. The rest of the program is stored on magnetic tape.

The system subroutines form the heart of the package. They read in the data, call the appropriate method(s), find a feasible starting point if necessary, print out the results, and perform a sensitivity analysis of the results if requested. OPTIPAC* is the name of the controlling subroutine which provides the overall logic. Access to the package is obtained by calling subroutine OPTIPAC from another program -- often a small "dummy" MAIN. Probably, the most powerful feature of the package is that a problem can be run on several methods at once. This provides both a check on the solution and an indication as to which is the most suitable optimization technique. As stated in the introduction, none

---

*The name "OPTIPAC" is derived from the words OPTImization PACkage. Although it is actually the name of a subroutine, it is used synonymously as the name of the whole package.

of the nonlinear methods is completely general, and several parameters, such as stopping criteria, step sizes, and the allowable number of moves, must be adjusted for each problem. Often it is difficult to choose these values in advance, and consequently the package has been designed to operate at two distinct levels. At the "unsophisticated" level, subroutine OPTIPAC automatically assigns reasonable values to all parameters which require judgment on the part of the user. This reduces the necessary input data considerably and makes it very easy to get an initial, rough solution. At the "sophisticated" level, the user must feed in the extra data cards to define all the program parameters. This enables him to tune methods specifically to his problem, thus obtaining the most accurate solution possible. This two-level facility is an extremely useful feature. It means the package is of equal value to a person who knows nothing about optimization theory and to someone who is familiar with the smallest details of each method.

The method subroutines contain the coding for the various optimization techniques. At present, these include revised Simplex for purely linear problems, and eight methods for nonlinear problems. These methods are: two types of direct search, a sequential direct search, an alternate search-linearization method, successive linear approximation, geometric programming and two different random search strategies. Such a wide variety of methods greatly increases the likelihood of the program finding a solution for any input problem. Obviously, the effectiveness of the package will increase as more methods are added, and the program has been set up with this in mind. Only a few modifications are necessary to incorporate an entirely new method. (The actual procedure involved is given in section 5 of the programmers' documentation[3]).

2.2 Service Subroutines

The description of the problem to be optimized is supplied to the package via the three service subroutines for all methods except revised Simplex and geometric programming. (These are highly specialized techniques for which the constraints and objective function must be fed in as data in a specified pattern). The objective function, equality constraints, and inequality constaints are evaluated in subroutines UREAL, EQUAL, and CONST respectively. In order to standardize the input to some extent, the following convention is used for stating the problem:

Minimize the objective function* defining the optimization criterion:

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

subject to equality constraints defining feasibility:

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

and inequality constraints defining feasibility:

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

where x_i are the 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 objective function is also known as the optimization, cost, or criterion function.

The user must abide by this convention, but it in no way detracts from the generality of the program. Maximization can easily be achieved by minimizing the negative of the true objective function. Also, inequalities of the form $\phi_k \leq 0$ can be readily converted to $\phi_k \geq 0$ by multiplying through by -1. If the constraints have non-zero terms on the right hand side, then these terms must be transposed to the left hand side. Problems with only one type of constraint (m=0 or p=0), or with no constraints at all (m=0 and p=0) are perfectly acceptable.

The input to the service subroutines is the X(I) array containing the current values of the design variables. The corresponding values of U, ψ_j and ϕ_k are calculated and returned to OPTIPAC. In the simplest case, the objective function and the constraints can be expressed directly as FORTRAN arithmetic statements such as,

```
      U=X(1)*X(3)
or    PSI(1)=X(1)-SIN(X(2))*3.0
or    PHI(3)=X(2)-16.0
```

Often, however, a more complicated analysis is involved. It may, for instance, require the solution to a set of eigen value equations in order to put a constraint on the eigen value itself.

e.g. `PHI(2)=EIGEN-2.3`

This is quite straightforward to do, since the user actually punches up the service subroutines and can therefore include as much coding as necessary. He may dimension his own working arrays, and call any subroutines he wishes from the computer library. If extra data such as physical constants or material properties is needed, it can be read in by the MAIN program and transferred to the service subroutines through labelled COMMON. When a complicated analysis is required, the user should include conditional STOP's after sections of coding which could possibly produce meaningless results. If, for instance, a matrix inversion fails, then the program should be stopped rather than have OPTIPAC continue, acting on misleading or even absurd information. It is extremely important that the service subroutines be written efficiently -- especially if they are complicated. They are called almost continually by the method subroutines and their execution time constitutes a large portion of the total execution time for the job.

Although the three service subroutines are very similar to each other from the programming point of view, they perform separate roles in specifying the optimization problem.

Objective Function: Subroutine UREAL

UREAL contains the coding to evaluate the objective function U at a point. Most frequently, this is the cost of the product. Other typical objective functions are weight, volume, strength, output power, aerodynamic drag, and fluid and thermal flow rates. The objective function must be dependent on at least one of the design variables, although it need not necessarily represent any physical characteristic of the design. For example, a specific value of horsepower could be obtained by minimizing,

```
U = (HPTEST-HPGOAL)**2
```

It is often difficult to choose a single objective function. For instance, the designer may want to minimize the cost and the volume at the same time. This is possible by writing,

```
U = WATE1*COST + WATE2*VOL
```

The weighting factors WATE1 and WATE2 are needed to compensate for large differences in the orders of magnitude of COST and VOL, and also to place emphasis on the more important of the two. Several trial runs would probably be necessary to determine reasonable values for these factors.

Equality Constraints: Subroutine EQUAL

EQUAL calculates the equality constraints ψ_j which are usually equations based on physical or chemical laws. They may also be design objectives such as,

PSI(1)=X(1)-X(2)

which could stipulate a beam of square cross-section for instance. Since all the nonlinear methods in OPTIPAC are basically exploratory strategies, the equality constraints are very rarely exactly equal to zero. This creates some technical difficulties which are later discussed for each method subroutine. For this reason, it is desirable to use as few ψ's as possible. If some tolerance is acceptable on either side of the equality, then quite often, two inequality constraints can be used instead.

$$\text{e.g.} \quad \left.\begin{array}{l} \text{PHI(1)=X(1)-X(2)+.01} \\ \text{PHI(2)=X(2)-X(1)+.01} \end{array}\right\} \text{ could replace PSI(1)=X(1)-X(2)}$$

Another problem with equality constraints is introduced if the independent variables are of different orders of magnitude. Typically, one constraint could be defining a buckling load of millions of pounds, while another specifies a flange thickness of a few inches. Weighting factors would be needed to prevent the buckling constraint from completely dominating the others. Alternate search (subroutine ALTS) is the only method which adds weighting factors internally. For the rest of the techniques, these weighting factors can be added directly in subroutine EQUAL as shown below:

```
PSI(1)=1.0*(X(1)-X(2))
PSI(2)=1.0E-06*(C1*X(3)**2-C2*X(3)-C3)
```

where X(3) is the critical buckling load and C1,C2 and C3 are functions of the other independent variables. The factors 1.0 and 1.0E-06 would probably have to be adjusted after a few trial runs.

Inequality Constraints: Subroutine CONST

CONST evaluates the inequality constraints ϕ_k, where $\phi_k \geq 0$ at a feasible point. They are used to place bounds on the independent variables themselves or on functions of them. Sometimes it can be quite difficult for the designer to know if he has put enough constraints on his problem. The best way for him to find out is by making a trial run and checking if the results are reasonable or not. Often, seemingly trivial restrictions must be included. For example, it may be necessary to have a constraint stating that the overall height of an I-beam is at least as great as two flange thicknesses. This fact is self-evident to the designer, but not to the purely mathematical optimization techniques. Geometric programming (subroutine GEOM) is the only method which assumes that all the design variables are positive. Any of the other methods will readily accept negative physical dimensions or even negative cost if specific constraints are not imposed.

Like the equality constraints, some of the inequality constraints may need weighting to allow for differences in magnitude or relative importance. These weighting factors have to be included in subroutine CONST since none of the methods is set up to add them internally.

The effect of weighting factors in the three service subroutines can be quite significant -- especially when using methods which minimize an unconstrained objective function with penalty terms added for violated constraints. This is discussed fully in section 2.4.

2.3 System Subroutines

The system subroutines make program OPTIPAC a coherent package rather than just a collection of different optimization techniques. They read in and screen the data, find a feasible starting point if necessary, print out the results and perform a sensitivity analysis upon request. Most important of all, they can process any number of data decks, permitting the user to try different methods and different program parameters all in one run. The purpose and operation of each of the system subroutines is explained below.

Central Control: Subroutine OPTIPAC

Subroutine OPTIPAC coordinates the operation of the entire package. It acts essentially like a main program, but is written in the form of a subroutine for two reasons. First of all, the initial DIMENSION statement presents a technical difficulty. Several arrays must be sized specifically for each problem to use the computer memory efficiently. This can be done only by inserting actual numbers into the arguments of array names in the DIMENSION statement of the main program. Since the whole package is on tape, this would be quite impractical. It would eliminate one of the system's major advantages -- a small input deck. In a subroutine, however, arrays may be given variable dimensioning which means that they expand to the size allotted to them in the calling program (see reference 3, page 5-2). Thus, by writing OPTIPAC as a subroutine, the package can still be stored on tape, and can be called by a very simple, or "dummy", program MAIN consisting basically of a DIMENSION statement and a CALL to OPTIPAC.

Making OPTIPAC a subroutine also permits any program to have access to it. For example, optimization of some intermediate results may be needed during the execution of a large analytical program. This could not be run as a continuous job if OPTIPAC was itself a main program. At McMaster, the package is kept semi-permanently* on a COMMON file "OPTAPE". This makes it available to any program having a control card COMMON(OPTAPE) and a CALL statement to subroutine OPTIPAC.

Since the user has to keypunch the MAIN program himself, the arrays in its DIMENSION statement, (and therefore the names in the CALL OPTIPAC argument list) are kept to a minimum. Only data arrays and large, doubly-subscripted working arrays are included. All other working space required by the package is declared in subroutine OPTIPAC as labelled COMMON blocks which are allotted to the other subroutines as shown on page 5-13 in reference 3. The blocks consist of from one to four arrays, each dimensioned (100). This scheme allows several subroutines to share storage space, although for small problems, the memory set aside for working arrays is larger than necessary. (This inefficiency could only be corrected by further complicating program MAIN). Another result of using working arrays of fixed size (100) is that input problems

---

*The COMMON file is re-created from a binary tape immediately after every "dead-start" of the computer. True permanent files are not yet available at McMaster.

are arbitrarily limited to having 100 independent variables, 100 equality constraints, and 100 inequality constraints.

After subroutine OPTIPAC has set up the labelled COMMON blocks, it clears all the working arrays and initializes the error flag, KO=0. (All subroutines in the package use KO=1 to indicate a failure of any kind). OPTIPAC then calls subroutine DATA to read in the data for the method being run. If KO=1 after DATA, the job is terminated because READ statements will have been omitted putting the remaining data cards out of phase. The values of INDEX, LEVEL and NSENSE which are returned from DATA, determine the flow of logic through the rest of the package.

INDEX identifies the method to be used, or signals the end of the data deck if set = 99. LEVEL indicates whether the package is to be run in the unsophisticated mode (LEVEL=0) or the sophisticated mode (LEVEL=1). If a sensitivity analysis has been requested, then subroutine DATA returns NSENSE=1 (otherwise NSENSE=0). Subroutine OPTIPAC first checks the value of INDEX to see if control must be returned to program MAIN (i.e. INDEX=99). If not, then a new set of data is ready and the level of sophistication is checked. Before calling the method subroutine, the computer's internal clock (subroutine SECOND) is referenced to obtain the time at the start of execution. If LEVEL=0, OPTIPAC presets the necessary program parameters and then calls the method subroutine stipulated by INDEX. At LEVEL=1, the method subroutine is called immediately after the return from DATA because all the program parameters are read in from data cards. The method subroutine performs the optimization procedure, prints out the results, and returns control to subroutine OPTIPAC. Subroutine SECOND is called again, and the net execution time for the method is calculated and printed out. Then, if the flag NSENSE=1, subroutine SENSE is called to do a sensitivity analysis of the results. Finally, control is returned to the beginning of subroutine OPTIPAC and the sequence is repeated for the next set of data. To summarize, subroutine OPTIPAC performs the following functions:

a) provides entry to the package from any other program
b) allocates storage space for all internal working arrays
c) clears these working arrays and sets KO=0
d) calls subroutine DATA to read user's input data deck
e) presets parameters for method subroutines at LEVEL=0
f) calls the appropriate method subroutine
g) calculates and prints out the net execution time for the method
h) calls subroutine SENSE if sensitivity analysis requested
i) repeats this sequence for many data decks until INDEX=99 is encountered.

System Input: Subroutine DATA

The purpose of subroutine DATA is to read in all the data for each method, check key parameters to see if they are acceptable, and list the input data (upon request) for the user's scrutiny.

Basically, subroutine DATA is a series of READ statements, one for every possible input parameter to the package. The first card of every method's data deck contains three parameters, INDEX, LEVEL and IDATA, which control the flow through the remainder of subroutine DATA.

Since the set up of the input deck for each method is completely specified in the users' manual,[2] the values of INDEX and LEVEL together determine which parameters are to be read in. Therefore, simple logical statements are placed before each READ so that unwanted parameters are bypassed. All arrays are cleared before data is read into them. Immediately following each READ, the parameter IDATA is tested and if IDATA=1, the value of the parameter(s) just read is printed out. This allows the user to check his input. On later runs he may suppress the listing by setting IDATA=0.

LEVEL and IDATA must be 0 or 1 while INDEX must be between 0 and 8 inclusive or be equal 99 to signal the end of the data decks. Subroutine DATA checks these values, and if any is unacceptable, the error flag KO is set equal to 1 and control is returned to OPTIPAC which returns to MAIN.

Subroutine DATA is designed to read in only the special OPTIPAC parameters described in reference 2. If the user has auxiliary data, (such as physical constants), which is needed by the service subroutines, then he must insert his own READ statements in program MAIN and transfer the information via labelled COMMON blocks.

<u>Feasible Starting Point:</u> Subroutine FEASBL

Several of the nonlinear optimization techniques require a feasible starting point, i.e., a point which satisfies all the contraints. In many cases however, the user does not know and cannot calculate a feasible point for his problem. To overcome this difficulty, subroutine FEASBL is included in the package.

FEASBL consists of two phases since there are two types of constraints. First of all, method subroutine SEEK3 is called to find a point which satisfies all the inequality constraints.* If such a point is obtained, then FEASBL uses a direct search in the feasible region to drive the equality constraints to zero. In this search, the objective function is the sum of the absolute values of the equality constraints, and ideally, the minimum is at zero. No acceleration or pattern move is used since the equalities are already reduced to reasonably small values in SEEK3. The actual final magnitude of the equalities can be controlled by the user at LEVEL=1 by his choice of the parameter "F" (see reference 3, page 5-76). If SEEK3 fails to find a point which satisfies the inequality constraints, then FEASBL cannot proceed because the direct search minimization of the ψ's can only operate in the feasible region. When this happens, an error message is printed out and the user must try another (still infeasible) input starting point.

In the current version of OPTIPAC, FEASBL is used by alternate search (ALTS) and successive linear approximation (APPROX). Neither of these methods can get started if any equalities are violated. Adaptive random search (ADRANS) does not require a feasible start, but calls subroutine FEASBL to speed up the method. These three methods call FEASBL automatically -- it is not an option controlled by the user.

---

*When called by FEASBL, SEEK3 cuts out as soon as a feasible point is found. It does not complete the optimization of the problem unless INDEX=3.

System Output: Subroutine ANSWER

Subroutine ANSWER is a convenient means of printing out the results of the methods in a neat, standardized form. As a safety feature, ANSWER evaluates U, PHI(I) and PSI(I) directly from subroutines UREAL, EQUAL and CONST respectively. This is necessary because the final values at the end of a method do not always correspond to the optimum point defined by X(I). For example, in a direct search, the method stops when no improvement can be found. In this case, the final values of PHI(I) and PSI(I) usually refer to the last unsuccessful (often infeasible) point tried. Also, the final value of U may actually be U plus some small penalty terms if equality constraints are involved. Subroutine ANSWER is used to print out either the optimum found or the results of the last iteration if the method stops prematurely. Intermediate results are printed out by the method subroutines according to the parameter IPRINT.[2]

Sensitivity Analysis: Subroutine SENSE

The designer is often interested in how the optimum would be affected by a small change in any of the independent variables. To provide him with this information, subroutine SENSE has been included in the package. Since it entails a large amount of output, the sensitivity analysis is only performed if specifically asked for (see reference 2, page 2-6). The procedure is quite straightforward. The first variable X(1) is decreased fractionally from its optimum value and U, PHI(I) and PSI(I) are calculated and printed out. The same is done for an increased value of X(1). Then X(1) is returned to the optimum and the next variable is changed, and so on. The fraction which is added and subtracted to each variable is FSENSE, a parameter input as data by the user. The print-out from SENSE allows the user to see which variables have a strong influence on U, and which constraints are sensitive to small changes in the variables, i.e. which are the critical constraints. Another useful type of sensitivity analysis, is to show the effect on the optimum of changes in the inequality constraints themselves. This can be achieved with OPTIPAC by running a problem several times, varying the PHI(I) statements in subroutine CONST. Typically, a "DO-loop" would be placed around CALL OPTIPAC in the program MAIN, and the constants to be changed in the inequalities would be stored in a labelled COMMON block.

Method Execution Time: Subroutine SECOND

To compare efficiencies of the various methods, the execution times must be considered as well as the optima obtained. On the C.D.C. 6400, subroutine SECOND provides access to the computer's internal clock. Therefore, SECOND is called immediately before and after the CALL to a method subroutine and the net execution time is simply the difference between the two readings. All computers have similar internal clocks, and only a minor modification is required to run on another machine (see reference 3, page 5-88).

2.4 Method Subroutines

The method subroutines contain the coding for the various optimization procedures. Every method can be run at LEVEL=0 (unsophisticated user) or at LEVEL=1 (sophisticated user). However, this only affects the values of the input parameters and the actual strategy used is identical for both values of LEVEL. The current version of OPTIPAC includes linear programming and eight nonlinear methods.

<u>Linear Programming</u>: Subroutine SIMPLE

Linear programming minimizes a linear objective function subject to linear constraints. It is included in the package for two reasons. First of all, two of the nonlinear methods, alternate search and successive linear approximation, require the minimization of a linearized system to determine optimum gradients. These methods could call the computer's own library subroutine directly, but that would introduce another machine-dependent feature. Also, the variable dimensioning scheme used elsewhere in the package could not be applied. This would mean that more array names would have to be added to subroutine OPTIPAC'S argument list and to program MAIN's DIMENSION statement. The second reason for including linear programming is to make OPTIPAC more general. It is written in the form of a separate method subroutine to allow the user to run a linear problem easily by following the straightforward instructions in the users' manual.[2]

The algorithm chosen is the I.B.M. subroutine "SIMPLE" which uses Revised Simplex, a computationally more efficient version of Dantzig's original Simplex method.[1,17] Slight modifications have been made to make the subroutine conform with the rest of the package, but the basic algorithm is unchanged. It performs Phase I and Phase II[2] so that an initial feasible basis is not required. It is important to note that SIMPLE assumes it is dealing with equations and the user must add slack variables to convert inequalities to equations. The number of slack variables plus the number of independent design variables gives the total number of Simplex variables, or columns in the Simplex tableau. Another restriction is that SIMPLE can handle only positive values of the Simplex variables. If any of the design variables is expected to be negative (a voltage or beam deflection for example), then the user can employ the substitution $x_i = (x_i^+ - x_i^-)$, where both x_i^+ and x_i^- are positive valued but x_i itself may be negative. Consider the constraint

3.*X(1) + 2.*X(2) = 4.0

If the user knows X(2) is negative, he must rewrite the constraint as,

3.*X(1) + 2.*X(2) -2.*X(3) = 4.0

The Simplex method calculates the optimum values of X(2) and X(3) and their difference gives the optimum value of the second design variable. This substitution is very useful, although it does increase the number of Simplex variables.

The only input parameter which the user can control (at LEVEL=1) is NSTOP, the maximum number of iterations* allowed without reaching an optimum. At LEVEL=0, this is set arbitrarily at four times the number of Simplex variables plus ten. If the program stops because NSTOP iterations have been exceeded, a message is printed out to tell the user whether or not the solution is still feasible. If it is, then the problem should run successfully with a larger input value of NSTOP. If the solution is not feasible after NSTOP iterations, it is unlikely that SIMPLE can find an optimum at all. This is usually due to an input

---

*One Simplex iteration consists of selecting the variable to be removed from the basis and the variable to be added to the basis, and performing the interchange.

error in the coefficients of the objective function or constraint equations. If the user omits a necessary constraint entirely, a message is printed out stating that the optimum is unbounded. The results at the optimum are printed out only when SIMPLE is being used as a method subroutine (INDEX=0). When it is called by ALTS or APPROX, there is no printed output except for error messages.

Nonlinear Programming

Five of the eight nonlinear methods contained in OPTIPAC are direct or random search techniques. They differ in their strategy for determining the direction and magnitude of trial moves and in their criteria for ending the search. These differences are significant and usually one method is considerably more efficient than the others for a particular problem. The direct searches are relatively fast but not always accurate, while the random searches are slow but can avoid or at least detect local optima. Two other techniques in OPTIPAC rely on a linear approximation of the nonlinear problem. One is the Method of Successive Linear Approximation (MAP) developed by Griffith and Stewart,[5] and the other is a combination of accelerated direct search and MAP developed by Gurunathan.[6] They both use a Simplex solution to determine the optimum gradient -- the direction which gives the largest improvement in the objective function. The remaining nonlinear method is geometric programming[4] which solves the special problem where all terms in the objective function and constraints are products of the design variables. In some limited cases, geometric programming yields the global optimum directly, but in general, a direct search is required to optimize the associated dual problem.

Direct Search: Subroutine SEEK1

SEEK1 uses the direct search strategy of Hooke and Jeeves[7] followed by a random search to check if a true optimum has been found.

All the direct search methods in OPTIPAC are based on the same principle. That is, to incorporate the constraints into an artificial objective function which can be minimized by systematically calculating its value at selected points in the search region, and taking the smallest value as the minimum. To account for the constraints, penalty terms are added to the real objective function whenever constraints are violated. By making these penalty terms proportional to the magnitude of the violation, it is possible to compare the values of the artificial objective function at different points and to move in the direction of the apparent optimum. For SEEK1, the penalty terms are simply the absolute value of each violated constraint multiplied by a large constant.[2]

In the "exploratory search", each variable is never changed by more than one basic step length and the results of the exploratory search determine the direction for making the larger, pattern moves. This means that the search is only accelerated on the basis of feedback from changes in all the variables. This is a major difference between SEEK1 and SEEK2. SEEK2 uses acceleration in the exploratory search itself to change each variable as much as possible before starting the pattern moves. The relative success of the two approaches depends entirely on the form of the problem and the starting point used.

Like most direct search methods, SEEK1 tends to stall on constraints. This occurs when no small change (equal to the specified minimum step size) in a single variable can improve the artificial objective function. Usually, an improvement could be found using a pattern move, but pattern moves are only possible after a successful exploratory search. To overcome this difficulty, SEEK1 employs a simple random search after the direct search has hung up. Every variable is increased (or decreased) by a random fraction of ten times* the original step length and the result is a composite move of random length and direction. At LEVEL=0, up to one hundred such moves are tried to find an improved value of the artificial objective function. At LEVEL=1, the number is specified by the input parameter NTEST. If an improvement is found, then the direct search is resumed. If not, the method assumes it has reached the optimum. Figure 1 shows how this random search gets the method started again after it has stalled on a constraint.

The input starting point and the weighting factors for the constraints can greatly influence the results of SEEK1. The starting point does not have to be feasible, but its position in relation to the constraints largely determines whether or not the method will hang up. Since it is often impossible for the user to visualize his problem in space, the safest approach is to run the problem with several different starting points.

The penalty terms added to the artificial objective function are proportional to the magnitude of the violation of each constraint. This

---

*Relatively large moves are made because the object is to get as far away from the constraints as possible so that the direct search can be started again.

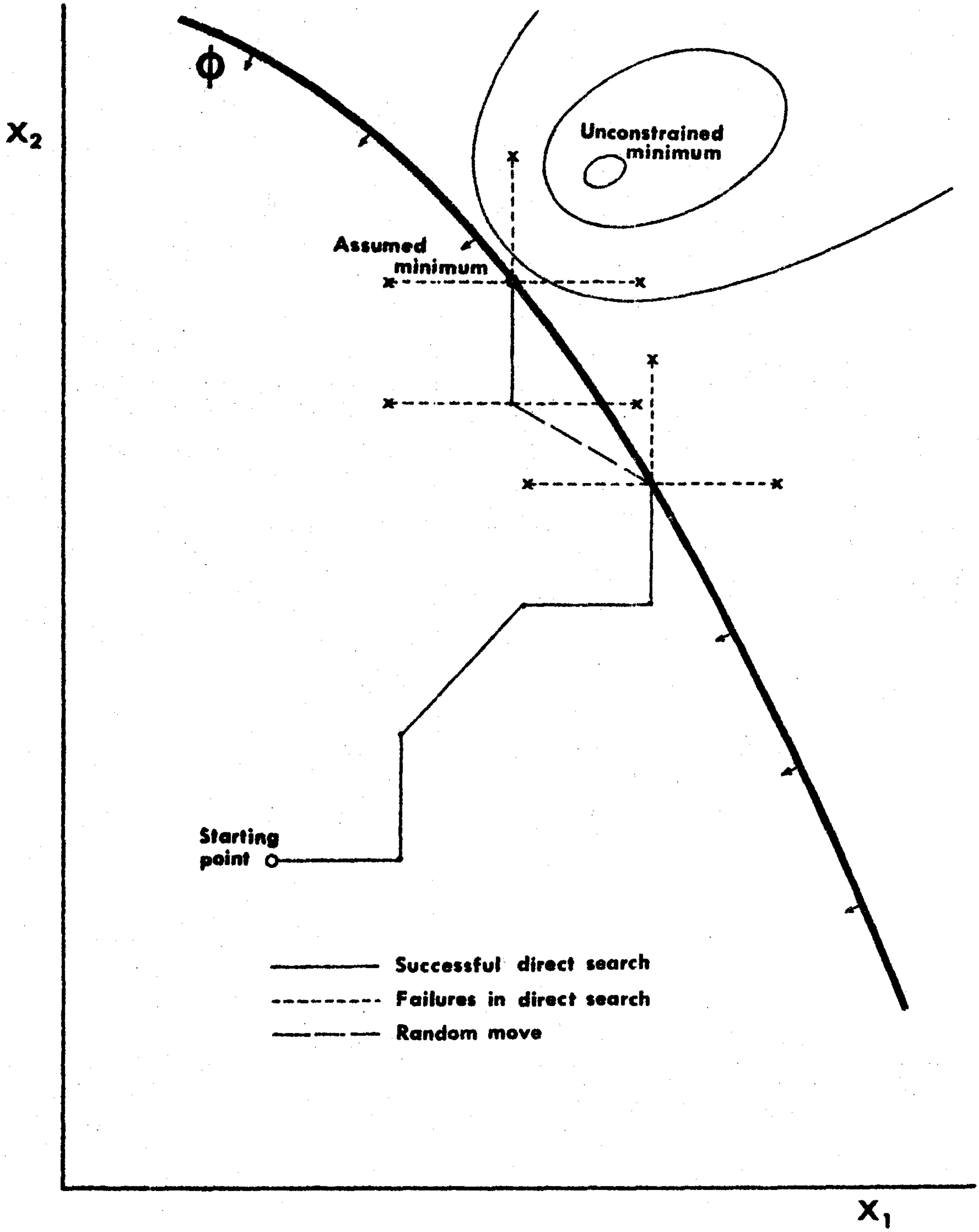


Figure 1. Restarting SEEK1 with a random move

causes difficulties when certain constraints are very sensitive to changes in a particular variable -- especially a change in sign. For example, a problem may have a simple inequality constraint to keep a small physical dimension, X(3) positive. There may also be a complicated equality constraint where X(3) appears in several terms multiplied by large factors. Then it is quite possible that, in moving from a positive to a negative value of X(3) the equality constraint is drastically reduced, while the inequality becomes slightly violated. The overall effect is a large improvement in the artificial objective function. After this type of jump has occurred, it is very difficult to drive X(3) positive again because the equality constraint has such a low value that almost any increase in X(3) increases the artificial objective function. In some cases, this prevents SEEK1 from obtaining a feasible solution at all. This trouble can be avoided by adding a large weighting factor to X(3) in the inequality constraint. That is, constrain 10000.*X(3) to be positive, rather than just X(3). Then a negative value of X(3) causes an overall increase in the artificial objective function as it should. To choose appropriate values, the user can run his problem at LEVEL=0 without any weighting factors and use the results to decide which (if any) constraints need to be weighted.

<u>Direct Search</u>: Subroutine SEEK2

SEEK2 uses the direct search strategy developed by Flood and Leon.[8,9] As mentioned above, the distinctive feature of this technique is that an acceleration procedure is used to advance each variable as far as possible before any pattern move is attempted. This approach is suitable for some problems, but in general, SEEK2 tends to be extremely sensitive to the input starting point and to the order in which the design variables are assigned to X(1) through X(N). The starting point is important because, by making large moves in a single direction, the method can hang up on constraints before all the variables have been changed. Then the final value of the objective function depends on the location of the starting point, as shown in Figure 2.

The user arbitrarily names the design variables X(1), X(2),...X(N) when he is formulating his problem. However, his choice fixes the order in which the design variables are moved, since SEEK2 always changes the X's in sequence, starting with X(1). The effect of the design variable assignments can be seen by studying Figure 2. Starting points B and C would have been quite acceptable if the variable X(2) had been moved first, that is; if the user had reversed the names of the design variables. Unfortunately, in most cases it is impossible to predict the best order -- especially since it may change as the solution proceeds. Flood and Leon[9] suggest randomly changing the order after every search iteration. This modification could easily be added as a small subroutine, and it would probably greatly improve the efficiency of the method. At present, SEEK2 does not have this feature, and the user must reformulate the problem to change the search sequence.

The penalty terms for SEEK2 are the same as for SEEK1, and weighting factors should be applied to the constraints in the same manner. The method stops when, moving with the minimum step size, the relative change in the artificial objective function is less than the specified tolerance EPS.

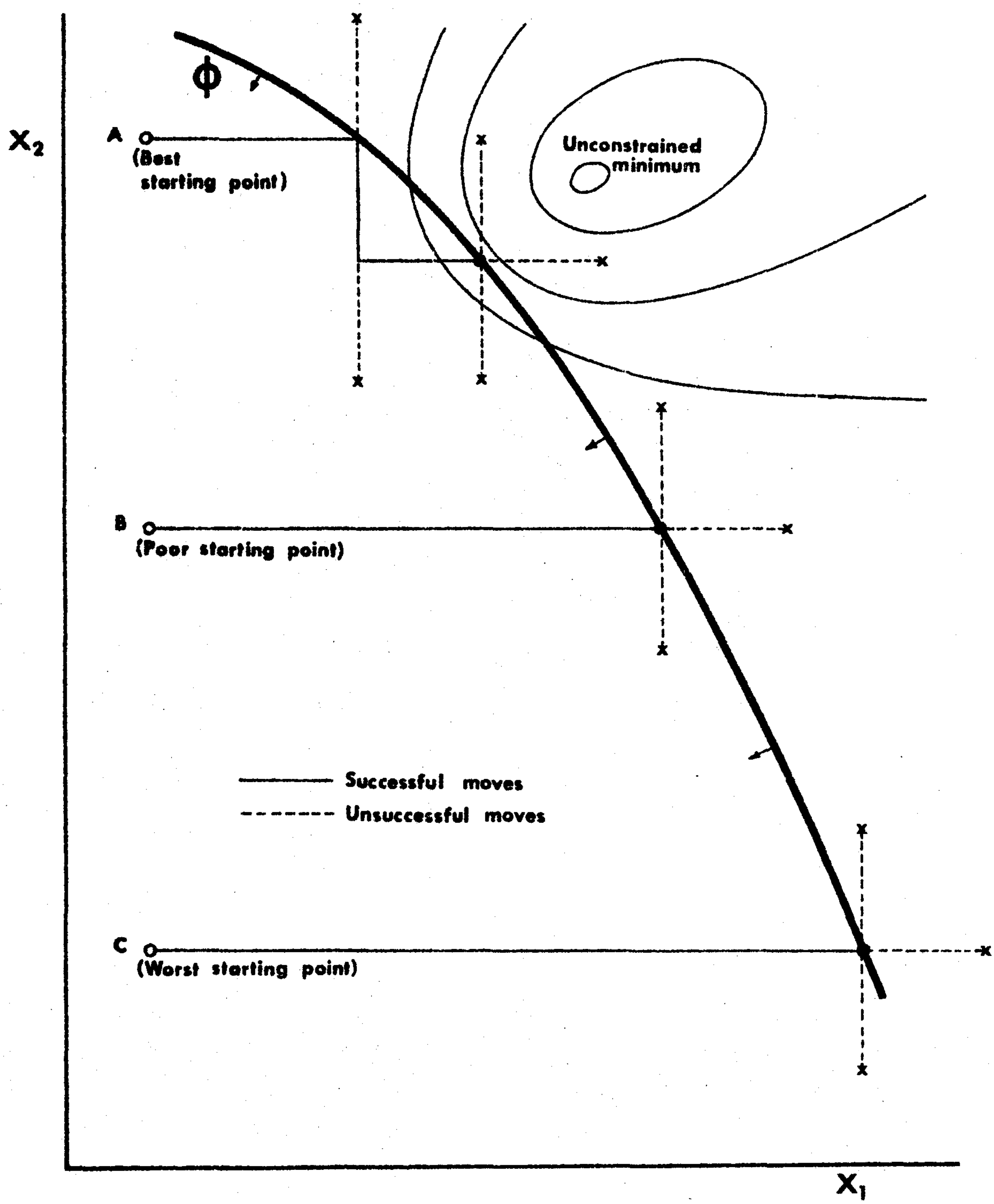


Figure 2. The importance of the starting point in SEEK2

Sequential Direct Search: Subroutine SEEK3

SEEK3 is based on a method by Fiacco and McCormick[10,11] which they call the Sequential Unconstrained Minimization Technique (SUMT).

The method consists of a series of direct search minimizations using the strategy of SEEK1. The artificial objective function uses special penalty terms[2] which are designed to prevent the solution from leaving the feasible region (all inequalities satisfied) while driving the equality constraints to zero. This assumes that the input starting point is feasible. To permit infeasible starting points, alternate penalty terms, like those used in SEEK1, are substituted for all unsatisfied inequality constraints. These alternate penalties are relatively large and the solution tends to the feasible region rapidly. Fiacco and McCormick have proposed another procedure for handling infeasible starting points which uses SUMT itself to drive the inequalities positive. Experience with OPTIPAC however, has indicated that the former approach is quite adequate.

Some effort has been made to find criteria for choosing the penalty term parameter R and its reduction factor REDUCE. No satisfactory answer has been found, and it appears that these parameters are problem-independent. Their values can affect the rate of convergence, but they do not influence the optimum obtained. The LEVEL=0 values of R=1.0 and REDUCE=.04 have proved effective for many test problems.

Each iteration of SEEK3 constitutes a complete minimization problem in itself. To reduce the number of calculations (and therefore computer time), some techniques have been developed[10] for extrapolating the results of successive iterations to speed up convergence. This is a feature which should definitely be added to SEEK3 in the future.

SEEK3 is not as prone to stalling on constraints as are SEEK1 and SEEK2, although some weighting factors (especially on equalities) are usually necessary. The method stops when the relative change in the objective function is less than 10^{-8} or when R has been reduced below 10^{-21}.

<u>Adaptive Random Search:</u> Subroutine ADRANS

ADRANS uses the pseudo-random search strategy originated by Gall.[12] The basic approach is to determine the optimum search direction by taking the mean path through five randomly generated improved points. The artificial objective function uses the same penalty terms as SEEK1,[2] and the method can handle infeasible starting points. An attractive feature of ADRANS is that every trial move involves changes in all the variables, making the method less likely to stall on constraints. Generating the trial random points is a cumbersome process, but the directions obtained are reliable and accelerated pattern moves help to improve the overall efficiency. At present, subroutine FEASBL is called to speed up the method by providing a reasonable starting point -- even though ADRANS does not require a <u>feasible</u> starting point.

ADRANS is assumed to have reached the optimum when no improvement in the artificial objective function can be found after generating a user-specified number (NSMAX) of random trial moves.

<u>Random Search:</u> Subroutine RANDOM

RANDOM is probably the best method in OPTIPAC for handling problems with local optima. The strategy used was developed by Dickinson and

Gallagher[13] although similar techniques have been devised by other authors.[14] The method evaluates the objective function at NUMR randomly chosen test points within the initial search region specified by the user. Points which violate any inequality constraints* are discarded, and the remainder are sorted according to their value of the objective function. Then the search area is shrunken to include only the NRET best points and the procedure is repeated until the range of each variable is acceptably small. The important feature here is that, if local optima exist in the original search region, they will prevent RANDOM from shrinking that region to any great extent. The user could then investigate his original area in smaller segments to locate the true optimum.

The number of random points generated and the shrinkage factor used can affect RANDOM's efficiency and so both parameters are controlled by the user at LEVEL=1. Since the whole object is to shrink the original search region, it follows that if the user excludes the optimum in his initial estimates of the design variable ranges, then it is impossible for RANDOM to reach that optimum.

<u>Successive Linear Approximation:</u> Subroutine APPROX

Griffith and Stewart[5] have developed a technique for conducting an extremely efficient search. The method converts the nonlinear problem into a linear problem by using a first order Taylor series expansion to approximate the objective function and the constraint equations about a

---

*RANDOM at present does not accept equality constraints.

point. This produces a system of linear equations and inequalities in which the variables are the steps to be taken in each search direction and the linearized objective function is the improvement in the objective function at the new point. After adding constraints to limit the step lengths,* this system is solved as a linear programming problem (subroutine SIMPLE) to find the optimum search vector. Every move is determined in this manner, and the process stops when SIMPLE cannot find a significant improvement in the objective function.

In practice, there appear to be two main difficulties with the method. First of all, the partial derivatives which form the Simplex coefficients are evaluated numerically[2] and they can be quite inaccurate. This is a serious problem when equality constraints are linearized because no compensating slack variables are added as they are to inequalities. The second problem is in determining the limits to be placed on the individual step lengths. Their maximum size has been arbitrarily set at ten percent of the range of each variable to satisfy the approximate Taylor series expansion. As the solution proceeds, it is necessary to decrease the allowable step lengths in order to force convergence. The logic which controls this step length regulation is purely intuitive on the part of the author[3] and it may prove to be too crude for larger problems.

APPROX has been very successful on the test problems tried and usually the difficulties mentioned above can be avoided by careful

---

*The step lengths are restricted to small values because the Taylor series expansion is only valid near the base point.

selection of the input parameters at the sophisticated level (LEVEL=1).

Alternate Search: Subroutine ALTS

A logical extention of the method of successive linear approximation is to combine it with a direct search in order to take better advantage of the optimum search direction, thus reducing the necessary number of Simplex solutions. Gurunathan's work[6] has been used as the basis for subroutine ALTS.

An accelerated direct search is carried out in the feasible region (all inequalities satisfied) with an artificial objective function composed of the true objective function plus the values of the equality constraints multiplied by weighting factors. Whenever the direct search stalls, a linearization is performed to find a new search direction. The process stops when no significant improvement can be obtained by either method. One disadvantage of ALTS is that a feasible starting point is required, but in most cases subroutine FEASBL[3] is able to locate one.

The major difficulty with the method is in choosing step length limitations for the linearizations. The problem is more pronounced than for APPROX because the linearizations are separated by portions of direct search and therefore the Simplex search directions do not develop in a recognizable pattern. At present, the step lengths are not adjusted at all, and oscillation or overstepping of the optimum can occur. Since convergence is not guaranteed, the method keeps track of the "best point so far" which is taken as the optimum if the method does not converge.

At LEVEL=1, the user has control over all important parameters[2] (including maximum step length) and he should be able to tune the method to his problem. The direct search portion of ALTS is particularly efficient for handling equality constraints. The linearizations will be more successful when a method of forcing convergence is perfected.

Geometric Programming: Subroutine GEOM

Geometric programming is the only special purpose nonlinear method in OPTIPAC. It was invented by Zener[4] to solve the problem where the objective function and inequality constraints are "posynomials", i.e. polynomials with positive coefficients. Also, the independent variables are restricted to having positive values.

The method involves a mathematical transformation to the dual problem, the maximization of the dual problem, and then a transformation back to the input or primal problem.[2] In certain cases,* the dual maximization is not needed as the mathematical transformations yield the global optimum directly. For most problems however, SEEK1 is required to maximize the dual objective function.

The most attractive feature of geometric programming is that the relative values of the primal and dual objective functions indicate whether or not the solution is optimal. They are equal at the global optimum, and represent upper and lower bounds on the global optimum if they are not equal. One major disadvantage of the method is that the transformation back to the primal problem is not always possible. Then the value of the dual function gives a lower bound on the optimum, but no information is gained about the values of the design variables.

In its present form, GEOM has very limited applications. It needs

---

*The global optimum is obtained directly when the "degree of difficulty" equals zero (see reference 2, page 4-50).

to be modified to permit negative polynomial coefficients, (and therefore greater-than-equal type inequality constraints), and negative independent variables. It has been used successfully to design electrical transformers[15] and journal bearings,[16] but problems with large "degrees-of-difficulty" have not been tested.

3. DOCUMENTATION FOR THE SYSTEM

The main object of OPTIPAC is to encourage the use of formal optimization procedures in engineering design. It is aimed largely at people unfamiliar with optimization theory and therefore the documentation for the system is extremely important. Separate manuals have been written for the user[2] and the programmer,[3] and a third manual is being compiled* to illustrate typical applications and sample input for some test problems.

3.1 The Users' Manual

The first section, "Quick Information", provides a very brief description of the whole system. The generalized form of the optimization problem which is solved by OPTIPAC is given, with an explanation of how to convert any problem to the standard form. The three categories of user, unsophisticated, sophisticated and programmer, are clearly defined so that the user can decide which parts of the documentation concern him. "Procedural Instructions" outline a systematic, check-list approach to running a problem, referring the user to the relevant documentation at every step. Finally, there is a list of the nine techniques currently included in the package and a simplified flow chart showing the program organization.

---

*The third manual is intended for commercial users and has not yet been completed. It is not described further in this thesis.

The second section explains how to set up the input deck, and describes the arrangement of the MAIN program, service subroutines and data deck. A diagram is used to show the complete input deck with all the control cards necessary to gain access to OPTIPAC which is stored on magnetic tape. Instructions are also given for running more than one method at a time and a second diagram illustrates this case. The sensitivity analysis which is contained in subroutine SENSE is described fully, and instructions for requesting it are given.

The third section of the users' manual contains the documentation for each of the method subroutines at the unsophisticated level. After a short introduction, there is a simplified flow chart to help the user choose methods for running his problem. This method selection chart is intended only as a rough guide however, and at the unsophisticated level, best results are obtained by trying as many methods as possible. The descriptions of the methods are written in a standard format and are very brief. A statement is given of the type of problem which can be handled, and the basic instructions necessary to run a job are provided. Virtually no background theory is included in this section. The data decks required by each method at this level are almost identical, which makes it very easy for the user to try several different techniques.

The fourth and last section of the users' manual contains the documentation for a sophisticated user. The layout is similar to that in the previous section, but considerably more detail is included. The basic theory behind each technique is outlined and useful references are given. A sub-section on special features helps the user choose values for all the input program parameters, and the default values of these parameters used at LEVEL=0 are listed. As an aid in de-bugging, a flow chart is provided to show which subroutines are called. Two excerpts from the users' manual are contained in Appendix A to illustrate typical documentation at both the unsophisticated and sophisticated levels.

3.2 The Programmers' Manual

The second manual contains all the information concerning the operation and organization of the FORTRAN program itself. It is divided into two parts: a description of the program, and an actual listing of the source deck.

The first section begins with a general description of the system, including its subroutine structure, the variable dimensioning scheme and the use of COMMON blocks. A "Thesaurus of Program Parameters" gives a complete alphabetical list of all user-input parameters together with their definitions. The details of each subroutine are discussed in a standard format. The internal variables are defined, and a flow chart of the program logic is given. A second, simplified flow chart shows how the particular subroutine is related to the rest of the package. Additional notes are used to elaborate on unusual or subtle aspects of the coding. The programmers' documentation for subroutine RANDOM is included in Appendix A as a typical example. Two other important topics which are covered in this manual are the incorporation of new method subroutines and features of the program which are machine-dependent.

The second half of the programmers' manual is taken up by the FORTRAN IV listing of OPTIPAC. Comment cards have been used liberally to help clarify the logic involved.

4. TEST PROBLEMS

The test problems discussed below represent real design problems chosen to give a good comparison of all the methods. They demonstrate clearly how difficult it is to predict which method will find the best solution. Several other problems were used in developing the individual methods and larger design problems have been run on the package by both undergraduate and graduate students at McMaster University.

The first example is the design of a three phase shell type electrical transformer. This was used as the main test problem for the geometric programming subroutine GEOM and it is fully described in Frank's paper.[15] The object is to minimize the volume of material while satisfying two geometrical constraints. GEOM assumes that all the variables are positive, but for the other methods, extra constraints are needed. Each of the independent variables is a physical dimension of the transformer, and the problem can be stated mathematically as follows:

Minimize,

$$U = .2007\ x_3x_4x_5 + .2697\ x_1x_2x_6 + \frac{3.69 \times 10^9\ x_6}{x_1x_2x_3^2x_4^2}$$

Subject to the inequality constraints,

$$\phi_1 = -4.0x_1/x_5 - 6.0x_2/x_5 - 4.0x_3/x_5 + 1.0 \geq 0$$
$$\phi_2 = -6.0x_3/x_6 - 6.0x_4/x_6 - 9.424x_1/x_6 + 1.0 \geq 0$$
$$\phi_3 = x_1 \geq 0$$
$$\phi_4 = x_2 \geq 0$$
$$\phi_5 = x_3 \geq 0$$
$$\phi_6 = x_4 \geq 0$$
$$\phi_7 = x_5 \geq 0$$
$$\phi_8 = x_6 \geq 0$$

The problem was run on eight methods at the unsophisticated level, and the results and execution times are tabulated in Appendix B. All methods used the same starting point. GEOM's solution agrees exactly with that of Frank.[15] It is particularly interesting to note how well some of the other methods work on this specialized problem. Sequential search, SEEK3, is especially good and the direct searches are considerably faster than GEOM itself. At the sophisticated level it would definitely be possible to adjust parameters in SEEK3 to obtain the global optimum. The histogram in Figure 3 gives a visual comparison of the minima obtained and the execution times required by each method.

The second test problem is a simple structural optimization, described by Siddall.[17] A three member indeterminant truss is to be designed for minimum weight. The lengths of the members are fixed and the structure must be able to support a one thousand pound load. Initially, eight independent variables were chosen: the cross-sectional areas and tensile stresses of each member, and the horizontal and vertical displacements of the point of application of the load. The problem could then be specified by two force-equilibrium equations, three displacement equations and nine inequality constraints restricting stresses, minimum areas and buckling loads. This formulation was run on OPTIPAC without much success. All of the methods had difficulty handling the five equations (equality constraints). After careful examination it was realized that only three of the variables were truly independent. Having chosen values for the three cross-sectional areas, the five equations become linear and can be solved by Gauss elimination for the

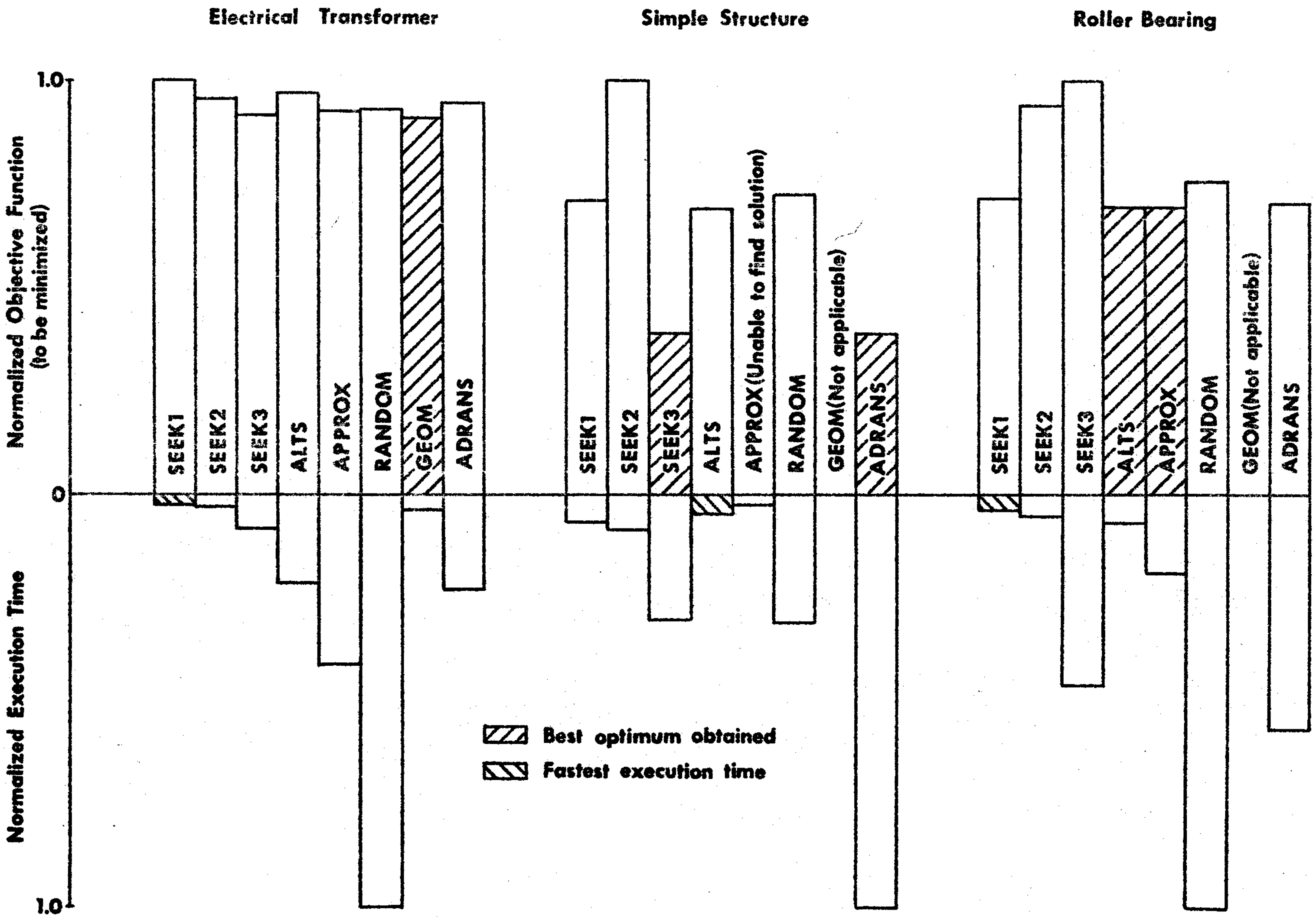


Figure 3. The relative performance of the nonlinear methods in OPTIPAC

remaining five intermediate variables. (These are sometimes called "state" variables). Inequality constraints are still imposed upon the intermediate variables, but the formal equality constraints are no longer necessary. This revised problem with three independent variables and nine inequality constraints was run on OPTIPAC at the unsophisticated level using seven methods, and the results are tabulated in Appendix B. (All methods used the same starting point). The mathematical statement of the problem is given below.

Minimize,

$$U = .283(50.9(x_1+x_3) + 36x_2)$$

Subject to the inequality constraints,

$$\begin{aligned}
\phi_1 &= 20000. - |x_4| \geq 0 \\
\phi_2 &= 20000. - |x_5| \geq 0 \\
\phi_3 &= 20000. - |x_6| \geq 0 \\
\phi_4 &= 10^{20}x_1 \geq 0 \\
\phi_5 &= 10^{20}x_2 \geq 0 \\
\phi_6 &= 10^{20}x_3 \geq 0 \\
\phi_7 &= \pi 7.5\ 10^6\ x_1^2 / 50.9^2 - |x_1x_4| \geq 0 \\
\phi_8 &= \pi 7.5\ 10^6\ x_2^2 / 36.0^2 - |x_2x_5| \geq 0 \\
\phi_9 &= \pi 7.5\ 10^6\ x_3^2 / 50.9^2 - |x_3x_6| \geq 0
\end{aligned}$$

It should be noted that the fourth, fifth and sixth constraints are heavily weighted to prevent the cross-sectional areas becoming negative. The variables x_4, x_5, x_6 in the above inequalities are obtained by solving the following set of linear equations for specified values of x_1, x_2 and x_3.

$$
\begin{aligned}
-.707x_1\,x_4 \qquad\qquad + .707x_3\,x_6 \qquad\qquad\qquad\qquad\qquad\qquad\qquad &= -866.\\
.707x_1\,x_4 + x_2\,x_5 + .707x_3\,x_6 \qquad\qquad\qquad\qquad\qquad\qquad\qquad &= 500.\\
50.9\,x_4 \qquad\qquad\qquad\qquad -21.21\;10^6\,x_7 + 21.21\;10^6 x_8 &= 0.\\
36.0\,x_5 \qquad\qquad\qquad\qquad\qquad\qquad + 30.00\;10^6 x_8 &= 0.\\
50.9\,x_6 + 21.21\;10^6\,x_7 + 21.21\;10^6 x_8 &= 0.
\end{aligned}
$$

Once again, the value of having several different techniques in a package is demonstrated. Adaptive random search, ADRANS, finds as low a minimum as sequential search (SEEK3) but it is almost four times slower. The fact that these two entirely different methods obtain identical solutions, gives the user some confidence that the global optimum has been achieved. Both alternate search and successive linear approximation have difficulty linearizing the constraints, and this could be due to the absolute terms in the inequalities. Figure 3 compares the relative performance of the seven methods tried. As this example shows, it is often possible to eliminate or at least reduce the number of equality constraints. The user should always have this aim in mind when formulating his problem.

The third test problem is based on the design of a simple roller bearing in which the total volume of material is the objective function to be minimized. Due to a slight error in one of the constraints,* the solutions obtained are not realistic. However, the example is still a perfectly valid optimization problem in the mathematical sense. It is included here because OPTIPAC's performance contrasts markedly with the two other test problems. The five independent variables selected are the

---

*The variable x_4 should appear in the denominator of the first term in ϕ_1.

thicknesses of the inner and outer races, the overall length of the bearing, the roller diameters, and a factor to control the spacing between rollers. Each of the four dimensions is limited by an inequality constraint, and the bearing must be able to support a radial load of ten thousand pounds. The spacing factor indirectly determines the number of rollers and an additional constraint stipulates that at least three rollers must be used. The problem is formulated as follows:

Minimize,

$$U = \pi x_5 \left[(x_1+x_2+x_3)^2 - (x_2+x_3+1)^2 + (x_3+1)^2 - 1 + \frac{\pi x_2 (x_2+2x_3+2)}{4x_4} \right]$$

Subject to the inequality constraints,

$$\begin{aligned}
\phi_1 &= 2735.\ x_5(x_3+1) - 10000. \geq 0 \\
\phi_2 &= x_1 - x_2 \geq 0 \\
\phi_3 &= x_3 - 0.6 \geq 0 \\
\phi_4 &= x_4 - 1.1 \geq 0 \\
\phi_5 &= \pi(x_2+2x_3+2)/x_2 x_4 - 3. \geq 0 \\
\phi_6 &= -x_5 + 8x_2 \geq 0
\end{aligned}$$

Appendix B shows the results from the seven methods run at the sophisticated level. (All methods use the same starting point). Geometric programming is not applicable because the objective function contains negative coefficients. The histogram in Figure 3 emphasizes again that the relative success of each method in the package is strongly problem-dependent. Sequential search, SEEK3, which is the best method in the structural example, is by far the worst method for this problem. APPROX and ALTS obtain the lowest value of the objective function here, but in the structural problem, ALTS is only mediocre and APPROX fails altogether. Direct search, SEEK1, which is consistently one of the fastest but least accurate methods, manages to find one of the best solutions.

5. DISCUSSION

A multi-technique package has proven to be a valid approach to the general problem of nonlinear optimization. The results of the test problems indicate clearly that a variety of methods is much more effective than any single method.

Direct search SEEK1 is usually the fastest method. It rarely finds the best optimum, although the simple random search at the end of the direct search prevents it from hanging up too badly. SEEK2 is almost as fast as SEEK1 but more prone to stalling on constraints. As mentioned in Section 2.4 of this thesis, SEEK2 needs to be modified so that the order in which the variables are moved is changed after every step. (This would probably be a worthwhile addition to SEEK1 as well). SEEK2 would also benefit from a random check on the optimum obtained and subroutine SHOT of SEEK1 could easily be incorporated for this purpose.

Sequential search, SEEK3 is considerably more accurate than either of the direct searches. This emphasizes the importance of the form of the penalty terms in the artificial objective function, since the actual search strategy is the same as that used in SEEK1. SEEK3's execution time could be reduced by adding the extrapolation feature described in Section 2.4.

Adaptive random search, ADRANS, is a reasonably accurate method, but it is slowed down severely by the cumbersome process of generating trial random points. It seems that there should be some means of progressively modifying the search area to speed up the process. For example, after one improved point is located, the remainder of the search could be concentrated in that area rather than continuing to search the full ranges of each variable. If this segment of ADRANS could be made more efficient, it would not be necessary to call subroutine FEASBL to start the method. (Calling subroutine FEASBL is undesirable because it introduces the difficulties associated with SEEK3 and SEEK1).

Random search, RANDOM, is slower than ADRANS, but it is the only method in OPTIPAC capable of detecting local optima. A useful modification would be to print out all the current "best" points when the method stops before convergence. The user could then use the local optima as starting points for other techniques to determine the true optimum. At present, only the lowest relative minimum is printed out when the method fails to converge. As explained previously, the initial search region specified by the user cannot be increased in RANDOM. This means that the input values of RMIN(I) and RMAX(I) act like strict limit equations on the variables. If the user excludes the optimum by specifying too small a range for any of the variables, it will show up in the solution because that variable will be approximately equal to one of its original bounds. The problem could then be rerun with an expanded initial search region. This difficulty does not occur frequently enough to warrant building in automatic expansion of the search area.

Successive linear approximation (APPROX) is potentially the most effective nonlinear technique in OPTIPAC. It is probably the only method which can be expected to work efficiently on very large problems. At the unsophisticated level, the linearization often fails because the numerical partial derivatives which make up the Simplex coefficients are too roughly approximated. At the sophisticated level, however, the user should be able to obtain good results for most problems. The method can handle equality constraints, provided that the starting point itself satisfies all the equalities. If the user cannot provide such a point, then subroutine FEASBL is called automatically to find one.

Alternate search (ALTS) attempts to combine the speed of direct search with the accuracy of successive linear approximation. The original idea was to use the linearization only to restart the direct search after it had hung up. In practice, the search seldom regains any momentum after its first failure. This is due to the fact that the search usually stalls close to the optimum or on a constraint boundary which permits only composite moves. This leads to a series of successive linearizations, but without the extra logic of APPROX to force convergence. The result may be oscillation or even divergence. The method stops when oscillation is detected, and stores the "best point so far" in case of divergence. The method is still not quite satisfactory however, and the entire step length regulation logic of APPROX should be incorporated. There appears to be a flaw in the basic concept of alternate search: it has combined two complete methods rather than just the best features of these methods. A more logical approach would seem to be to choose all search directions exclusively by linearizing the problem and to determine the correct step lengths by a direct search in the direction obtained. In this way, ALTS would truly utilize only the best features of the two different techniques.

Geometric programming (GEOM) is the only special purpose nonlinear method in the package. It has performed well on very restricted problems, but still needs several modifications which are outlined in Section 2.4 of the thesis.

No difficulties have been encountered with the revised Simplex algorithm SIMPLE. A useful addition would be to automatically make the standard substitution which allows negative Simplex variables. This is already a feature of alternate search and successive linear approximation.

All of the methods have difficulty compensating for constraints of vastly different magnitudes, since the largest constraints tend to dominate the others. Ideally, the program should put equal emphasis on all the constraints unless the user specifically includes weighting factors in the service subroutines. One approach[17] is to normalize all the independent variables by dividing each one by its estimated range. This scaling of the independent variables is useful in unconstrained problems to make step lengths and gradients more uniform. (It would definitely be an asset in the linearizations performed in ALTS and APPROX). It does not, however, make a significant improvement in the constrained case. A better solution seems to be to normalize the magnitudes of the constraints themselves in some fashion. One crude range approximation could be obtained by evaluating each constraint at the upper bounds and then the lower bounds of all the independent variables. The differences could then be used as the scaling factors for subsequent values of the constraints. In certain problems, the user may be able to actually input accurate estimates of the expected ranges. It should be pointed out that the existing method of entering weighting factors is quite satisfactory from the analytical viewpoint, but it requires too much judgment and experience on the part of the user. In a system such as OPTIPAC, the user should not need to get involved with the technicalities of the program.

Very few problems with equality constraints have been run successfully on the package. SEEK3, ALTS, and APPROX are best equipped to handle them, but even these methods have considerable difficulty if the starting point is badly infeasible. Equality constraints are extremely restrictive because they force the solution to move right along a boundary, which is much more demanding than merely staying on one side of the boundary (inequality constraints). The direct searches (SEEK1 and SEEK2) hang up frequently because once they reach a point on or very close to an equality, they cannot find a better point. Their exploratory search does not allow for the necessary move along the constraint. Sequential search, SEEK3, is more successful because of the special form of the penalty terms in the artificial objective function. For the first minimization, the equalities are virtually ignored due to very small weighting factors. The method first concentrates on finding a point which satisfies all the inequalities. On subsequent minimizations, the equalities are gradually emphasized more until they are finally forced to zero. The direct search portion of alternate search (ALTS) uses a somewhat similar strategy, although it requires that the starting point satisfy all inequalities. The search is conducted in the feasible region, with user-specified weighting factors (WATE(I)) to drive the equality constraints to zero. The linearization technique of ALTS and APPROX is ideal for following the constraint boundaries, and APPROX appears to be the best method for handling problems with a large number of equality constraints. RANDOM and GEOM do not accept equalities at all. ADRANS is very inefficient since so many random points must be generated to obtain another point on the constraint boundary. (Execution times soon become prohibitive).

The whole question of equality constraints is completely ignored by many authors. They apparently feel that optimization pertains mainly to the solution of inequalities, while systems of equations are best handled by the methods of numerical analysis and classical mathematics. This is a valid argument in some cases and the structural test problem in this thesis shows how equality constraints can often be eliminated. When they cannot be avoided by reformulating the problem, it is always possible to replace an equality by two inequalities. This implies that some tolerance is acceptable, but the tolerance can be reduced on successive runs until the equality constraint is satisfied exactly.

As a computer system, OPTIPAC has performed well. Problems have been run by a variety of users, many of them unfamiliar with optimization and inexperienced in programming. Most have preferred the unsophisticated mode of operation because the input is very simple and all applicable methods use virtually identical data decks. The users' documentation has proven to be more than adequate, and it is constantly being revised as minor mistakes are discovered. At present, the programmers' manual[3] is referred to mainly by users interested in the FORTRAN listing of OPTIPAC. When major changes to the system are being made, the rest of this manual will be indispensable.

Now that some operational experience with the package has been gained, it is possible to suggest where changes and additions might be made to improve OPTIPAC.

One of the weakest features of the system is the method selection chart. Presently, the most reliable way of choosing a method is to run the problem on all the methods at the unsophisticated level to see which one gives the best results. This would obviously be impractical with very large problems. As more test problems are run, it should be possible to establish a statistical basis for method selection. That is, the efficiency of each method will be functionally related to the key parameters defining the input problem. Typically these would include the number of variables, the number of equality and inequality constraints, and a parameter to indicate the degree of nonlinearity. With this sort of information, the program could choose the most efficient method completely automatically. Before incorporating this feature, some changes to the method of data input would be necessary.

Since the user does not know in advance which method will be run, then he must supply sufficient data to run every method in a single data deck. At the unsophisticated level this is simple, but at the sophisticated level it may mean specifying values for over twenty parameters. To reduce this number, it will be necessary to further standardize several parameters, such as stopping criteria, so that they apply to all methods. Limits on the number of moves or complete iterations can probably be related to the number of variables and thus eliminated from the input deck.

In the present system, all data cards are always read in by the system subroutine DATA. It is now apparent that the user should have the option of bypassing subroutine DATA in order to transfer data directly to OPTIPAC through its argument list and through blank COMMON. This option is essential if the package is to be available as a standard subroutine to other programs when optimization input data is internally generated. Only a very simple modification is needed to add this feature. For example, a value of IPRINT exceeding 500 could be used as the flag for bypassing subroutine DATA. The true value of IPRINT would then be obtained by subtracting 500 from its input value. The overall operation of the system would be unchanged, and runs could still be made at either level of sophistication.

The modifications discussed here represent only some of the more significant improvements which could be made to the system. Necessary changes to the FORTRAN coding itself may become apparent with further usage.

6. CONCLUSIONS

OPTIPAC has been developed to encourage the use of formal optimization techniques in engineering design. Its aim is to provide a system which is easy to use, and yet capable of handling a wide variety of both linear and nonlinear problems. The project consisted of two phases: developing the FORTRAN program itself; and writing detailed documentation for three separate types of user.

Since there is no generally applicable nonlinear optimization technique, several different methods have been incorporated into a single package. Input/output is controlled internally and the system may be operated at two distinct levels, depending on the user's familiarity with optimization and programming. Many test problems have been run and they have shown that a multi-technique approach is well justified. Although the performance of individual methods is unpredictable, at least one of the eight methods can usually obtain a reasonable solution.

It was realized at the beginning of the project, that designers would not use the package unless it was accompanied by thorough documentation. Therefore, a considerable amount of time was spent in compiling a manual for the user[2] and a second manual[3] describing the programming aspects of the system. The users' manual contains explicit, step-by-step instructions for running a job and these have proven to be more than adequate. Students at the undergraduate and graduate level in the Design program at McMaster, have been able to run problems without difficulty. Considerable interest in the system has also been shown by people outside the university. Those who have already used or are intending to use OPTIPAC are: the University of Texas; Sheffield University, England; the National Research Council (Ottawa); STELCO Research Division; DOFASCO; and the Butler Manufacturing Company. The latter three companies are all located in Hamilton.

OPTIPAC's problem-solving ability is limited only by the number of techniques included, and the program has been designed to make the addition of new methods straightforward. As a system, OPTIPAC is still relatively unsophisticated. Its ultimate configuration will probably be as a "conversational" program, with the user interracting through a time-shared terminal.

While it is far from being in its final form, OPTIPAC does appear to have succeeded in its two main objectives. It does handle a wide range of problems, and the system is easy to use.

APPENDIX

A) SAMPLE DOCUMENTATION FOR 'OPTIPAC'[2,3]

(Unsophisticated User)

3-22

RANDOM SEARCH

Name

RANDOM

Purpose

To solve a nonlinear optimization function with nonlinear inequality constraints.

The function to be minimized will be of the form $U = U(x_1, x_2, \ldots x_n)$

and constraints of the form $\phi_k(x_1, x_2, \ldots x_n) \geq 0 \qquad k=1,p$

Method

The method consists of a random search for the minimum or simply a shotgun technique with iterative shrinkage.

List of Input Variables

INDEX	index number of subroutine, = 6
LEVEL	level of sophistication, = 0
IPRINT	prints intermediate results every IPRINT cycle, set at zero for no intermediate data
IDATA	if IDATA = 1 the input data will be printed out, otherwise set at zero
N	number of variables (specified in MAIN)
NCONS	number of inequality constraints
RMAX(I)	estimated upper bound for variable X(I)
RMIN(I)	estimated lower bound for variable X(I)

List of Output Variables

U	minimum value of the optimization function
X(I)	values of independent variables at the optimum

(Unsophisticated User)

How to Set Up MAIN Program

```
 DIMENSION X(N),PHI(NCONS),RMAX(N),RMIN(N),Z(J,N),UU(J)
 N=numerical value
 J=numerical value
 M=1
 NN=1
 NTOTER=1
 CALL OPTIPAC(X,PHI,PSI,A,B,C,WORKA,DELX,STEP,XSTRT,RMAX,RMIN,DSTAR
1,NTERMS,GS,WATE,TEST,Z,UU,EX,CONST,AA,BBB,CC,NCONS,NEQUS,M,N,NN,NT
2OTER,J,XX)
 STOP
 END
```

Note: The numerical values of N, NCONS, J (J = 3*N) must be inserted in the DIMENSION statement. If NCONS is zero then put PHI (1) in the DIMENSION statement.

How to Make Up Data Deck

Variable Name	No. of Cards	Format
INDEX, LEVEL, IPRINT, IDATA	1	4I3
NCONS	1	I5
RMAX(I)	as many as required	5E16.8
RMIN(I)	as many as required	5E16.8

Setting Up Service Subroutines

UREAL, see page 3-30

CONST, see page 3-34

Miscellaneous

The values of RMIN(I), RMAX(I) put in by the user establish absolute bounds on the variables which can only shrink. If the user is unsure, it is safest to make RMAX(I) too large and RMIN(I) too small.

(Sophisticated User)

RANDOM SEARCH

Name

RANDOM

Purpose

To solve a nonlinear optimization function with nonlinear inequality constraints. The function to be minimized will be of the form $U = U(x_1, x_2, \ldots x_n)$ and constraints of the form $\phi_k(x_1, x_2, \ldots x_n) \geq 0$ $k=1,p$

Method

The method consists of a random search for the minimum, or simply a shotgun technique, with iterative shrinkage. Random points for each variable x_1 to x_n are generated from the expression $x_i = \ell_i + r_i(u_i - \ell_i)$

where ℓ_i is the estimated lower limit for x_i

u_i is the estimated upper limit for x_i

r_i is a random number uniformly distributed between zero and one.

Any generated point that violates an inequality constraint is discarded. If the constraints are violated NSMAX times consecutively the process will stop. Problems having more than a few constraints are liable to bog down in violations, particularly if the initial limits overlap appreciably infeasible areas.

The search is begun by evaluating NUMR random points by use of the above equation, NUMR being a multiple of the number of variables. From these the best NRET are selected and used as the basis for a new and shrunken range for each variable. NRET is defined by NUMR/NSHRIN where NSHRIN is a shrinkage factor. Within this new space NUMR new random points are evaluated. These, plus the previous NRET best, are sorted to yield a new NRET best and a new shrunken space. The process is repeated until the range of each variable is acceptably small, or until the range has been shrunken MAXM times.

(Sophisticated User)

4-46

References

1. McArthur, D.S., "Strategy in Research - Alternative Methods for Design of Experiments", IRE Trans. on Engrg. Management, Vol EM-8, March 1961, pp. 34-40.

2. Gallagher, P.J., "MOP-1, An Optimizing Routine for the IBM 650", Can. GE Civilian Atomic Power Dept. Report No. R60cAP35, 1960.

Special Features

MSTART is an integer used to initialize the random number generator subroutine FRANDN. If a large number of random points is generated (MAXM and/or NSMAX very large), several values of MSTART should be tried to insure that the random numbers are being uniformly distributed.

It should be noted that the user's input values for RMAX(I) and RMIN(I) establish absolute extremes for the variables which can only shrink. If there is any uncertainty, RMAX(I) should be made higher than expected and RMIN(I) lower than expected. At LEVEL = 0, parameters set internally for RANDOM are:

F = .001

NSMAX = 300

MAXM = 400

NSHRIN = 4

MSTART = 128

NUMR is set internally in RANDOM as NUMR=J*NSHRIN, where J is set in MAIN and is equivalent to NRET. The user can set NRET and NUMR independently since he inputs J and NSHRIN. A reasonable value of J is the integer result of 10*N/NSHRIN.

(Sophisticated User)

4-47

Input Variables

INDEX	index number of subroutine, = 6
LEVEL	level of sophistication, = 1
IPRINT	prints results every IPRINT cycles, set at zero for no intermediate output
IDATA	= 1, all input data is printed out = 1, input data is not printed out
N	the number of independent variables X(I) (specified in MAIN)
NCONS	the number of inequality constraints
F	fraction of original input range used as a convergence criterion
NSHRIN	shrinkage factor
MSTART	any positive integer, used as starting value for generating random numbers
MAXM	maximum number of cycles allowed if process does not converge
NSMAX	maximum number of times constraints can be violated consecutively before abandoning the search
RMAX(I)	upper bound for variable X(I)
RMIN(I)	lower bound for variable X(I)

Output Variables

U	minimum value for the function
X(I)	value of x_i where minimum occurs

(Sophisticated User)

4-48

How to Set Up MAIN Program

```
 DIMENSION X(N),PHI(NCONS),RMAX(N),RMIN(N),Z(J,N),UU(J)
 N=numerical value
 J=numerical value
 M=1
 NN=1
 NTOTER=1
 CALL OPTIPAC(X,PHI,PSI,A,B,C,WORKA,DELX,STEP,XSTRT,RMAX,RMIN,DSTAR
1,NTERMS,GS,WATE,TEST,Z,UU,EX,CONST,AA,BBB,CC,NCONS,NEQUS,M,N,NN,NT
2OTER,J,XX)
 STOP
 END
```

Note: The numerical values of N, NCONS, J (J = NRET) must be inserted in the DIMENSION statement. If NCONS is zero, then put PHI(1) in the DIMENSION statement.

How to Set Up Data Deck

Name	No. of Cards	Format
INDEX, LEVEL, IPRINT, IDATA	1	4I3
NCONS	1	I5
F	1	E16.8
MAXM	1	I6
MSTART	1	I6
NSHRIN	1	I6
NSMAX	1	I6
RMAX(I)	as many as required	5E16.8
RMIN(I)	as many as required	5E16.8

Setting Up Service Subroutines

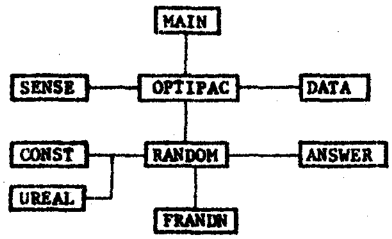
UREAL, see page 4-63

CONST, see page 4-67

(Sophisticated User)

4-49

Subroutines Called



Miscellaneous

RANDOM is a relatively slow method, but it does not hang up on local optima. For this reason, it is a good method for checking the results of other methods.

An improved optimum may be obtained, at the expense of time, by using a larger value of NSHRIN. RANDOM will not run efficiently with small values of NSHRIN, say less than 3.

(Programmer)

5-57

SUBROUTINE RANDOM

General

Subroutine RANDOM is used only as a method subroutine and is called only by OPTIPAC.

Internal Variables

Variables not included in the list below, can be found in the Thesaurus of Program Parameters.

Name	Definition
AA(I)	Lower bounds on X(I), set=RMIN(I) initially
CC(I)	Upper bounds on X(I), set=RMAX(I) initially
F	Fraction of initial range used as the maximum acceptable range for convergence
KK	Temporary counter to compare with IPRINT for printout
KO	Flag, set=1 after abnormal exit, otherwise KO=0
L	Temporary counter of consecutive constraint violations
L1,L2	Temporary counters used for printing out results
LARGE	Temporary variable used for sorting the UU array
MAXM	Maximum number of cycles permitted if no convergence
MM	An integer constant required by subroutine FRANDN, set=0 after initial CALL FRANDN
MSTART	Any positive integer to be used as the initial value of MM
N	Number of independent variables X(I)
NCONS	Number of inequality constraints PHI(I)
NCYCLE	Counter of the number of complete cycles

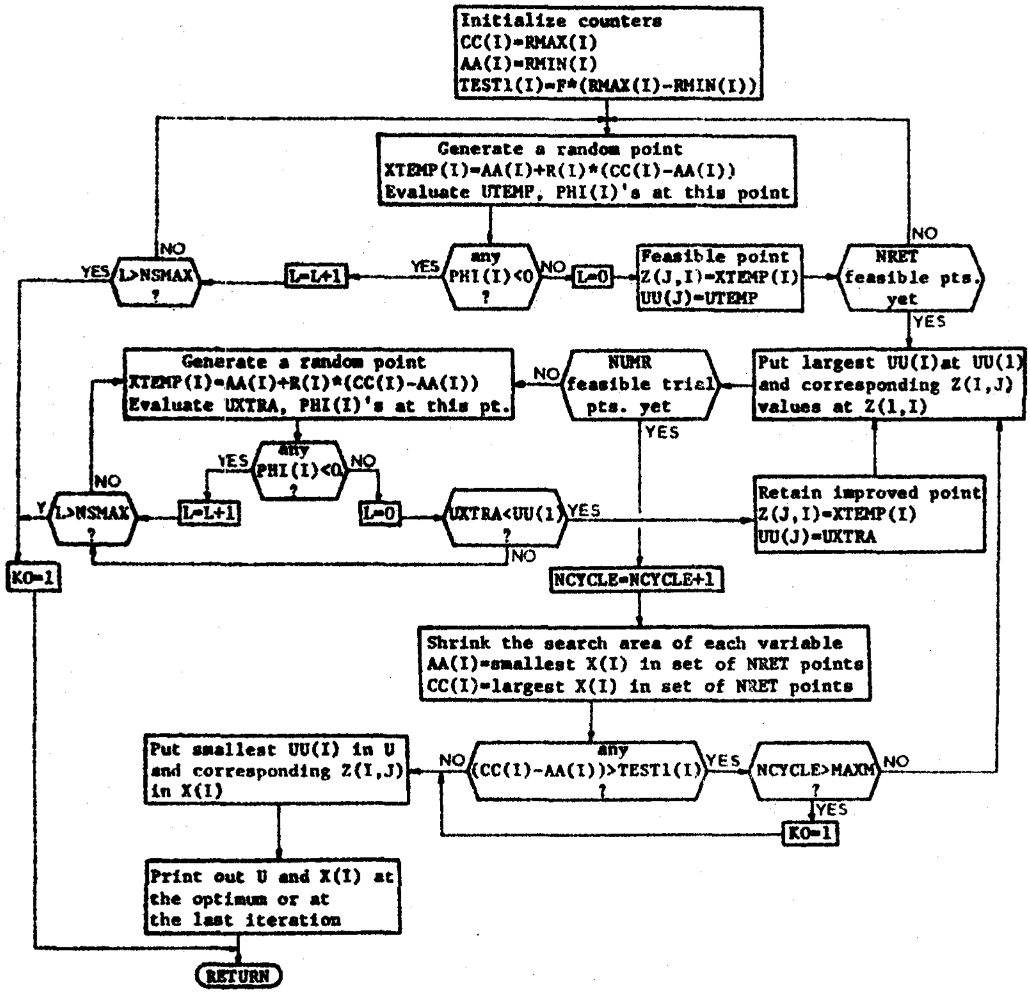
(Programmer)

NRET	Number of "best" random feasible points retained in each cycle, called J in MAIN program, and used to dimension the Z array
NSHRIN	Shrinkage factor where NRET=NUMR/NSHRIN
NSMAX	Maximum number of consecutive infeasible random points permitted
NUMR	Number of random feasible points generated each cycle=NRET*NSHRIN
NVIOL	Counts the number of constraints violated at a point
PHI(I)	Values of the inequality constraints
R(I)	A string of N random numbers associated with X(I)
U	Value of the optimization function at the optimum
UTEMP	Value of the optimization function at a trial point
UU(I)	Values of the optimization function at each of the NRET feasible points, UU(1) contains the largest value
UXTRA	Temporary storage for trial values of U
TEST1(I)	The maximum acceptable range of X(I) at convergence
X(I)	Values of the independent variables at the optimum
XTEMP(I)	Values of the independent variables at trial points
Z(I,J)	The NRET best random feasible points, stored in rows

(Programmer)

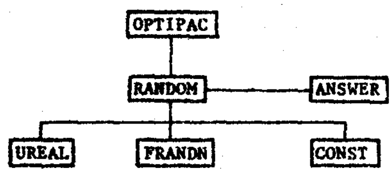
5-59

Program Logic



(Programmer)

Calls To and From Subroutine RANDOM



Notes

RANDOM cannot handle inequality constraints, and NEQUS is therefore not an input parameter. To avoid getting an indefinite error message in subroutine ANSWER, NEQUS is set=0 in the body of RANDOM.

If MAXM cycles are exceeded, it is still necessary to sort the UU(I) array so that the best point so far can be output.

B) RESULTS OF TEST PROBLEMS

Design of a Three Phase Electrical Transformer

Number of independent variables	6
Number of inequality constraints	8
Number of equality constraints	0
User's level of sophistication	0
Number of methods tried	8

Method Name	Time (Secs)	U (cu.ins.)	Independent Variables (ins)					
			x_1	x_2	x_3	x_4	x_5	x_6
SEEK1	0.68	73042.	11.27	14.42	11.78	57.99	178.69	524.81
SEEK2	0.69	70017.	7.90	15.75	16.92	53.87	193.76	499.21
SEEK3	2.39	66723.	8.66	12.91	18.86	40.77	187.56	439.45
ALTS	6.44	70704.	10.26	11.22	16.25	62.50	173.39	569.23
APPROX	12.24	67572.	10.13	10.00	18.00	50.00	172.54	503.55
RANDOM	29.69	68007.	8.67	11.49	16.01	58.26	167.66	527.40
GEOM	1.09	66704.	8.41	13.09	18.75	40.81	187.15	436.56
ADRANS	6.61	69077.	9.41	8.75	15.57	67.82	152.44	589.25

Design of a Three Member, 2-Dimensional Structure

Number of independent variables	3
Number of inequality constraints	9
Number of equality constraints	0
User's level of sophistication	0
Number of methods tried	7

As described in the text, equality constraints have been avoided by careful formulation of the problem. Only seven methods were run because the problem is not of a form acceptable to geometric programming.

Method Name	Time (Secs)	U (lbs)	Independent Variables (sq.ins)		
			x_1	x_2	x_3
SEEK1	0.71	5.659	.0095	.0649	.3375
SEEK2	0.94	7.995	.0000	.0778	.5000
SEEK3	3.50	3.127	.0483	.0000	.1688
ALTS**	0.47	5.545	.0247	.0375	.0334
APPROX*	0.25	-	-	-	-
RANDOM	3.52	5.777	.0215	.0427	.3493
ADRANS	11.52	3.127	.0483	.0000	.1688

** Subroutine ALTS could not make a linearized step after the direct search had hung up. The values shown are simply the results at the last iteration of the direct search.

* Subroutine APPROX could not perform the second linearization and therefore could not get started.

<u>Design of a Simple Roller Bearing</u>

Number of independent variables	5
Number of inequality constraints	6
Number of equality constraints	0
User's level of sophistication	1
Number of methods tried	7

Method Name	Time (Secs)	U (cu.ins)	Independent Variables				
			x_1	x_2	x_3	x_4	x_5
SEEK1	0.54	20.350	.280	.280	.637	13.29	2.240
SEEK2	0.71	26.976	.198	.198	1.314	25.50	1.585
SEEK3	6.65	28.695	.185	.185	1.472	28.96	1.484
ALTS	0.90	20.053	.287	.287	.600	12.74	2.292
APPROX	2.74	20.053	.287	.287	.600	12.74	2.292
RANDOM	14.37	21.708	.311	.287	.648	9.09	2.225
ADRANS	8.14	20.077	.287	.287	.600	12.65	2.293

C) FORTRAN IV LISTING OF PROGRAM OPTIPAC

```
      SUBROUTINE OPTIPAC(X,PHI,PSI,A,B,C,WORKA,DELX,STEP,XSTRT,RMAX,RMIN
     1,DSTAR,NTERMS,GS,WATE,TEST,Z,UU,EX,CONST,AA,BBB,CC,NCONS,NEQUS,M,N
     2,NN,NTOTER,NRET,XX)
      DIMENSION X(1),PHI(1),PSI(1),Z(NRET,1),A(M,1),B(1),C(1),WORKA(1),
     1CC(NTOTER,1),XX(1),DELX(1),STEP(1),XSTRT(1),RMAX(1),RMIN(1),DSTAR(
     2NTOTER,1),NTERMS(1),GS(1),WATE(1),TEST(1),UU(1),EX(NTOTER,1),CONST
     3(1),AA(NTOTER,1),BBB(NTOTER,1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A1/WORK1(100),WORK2(100),WORK3(100),WORK4(100)
      COMMON /A2/WORK5(100),WORK6(100)
      COMMON /A3/WORK9(100),WORK10(100),WORK11(100)
      COMMON /A4/WORK12(100),WORK13(100),WORK14(100),WORK15(100)
      COMMON /A5/WORK16(100)
      COMMON /A7/WORK18(100),WORK19(100)
      COMMON /A8/IWORK1(100)
      COMMON /NAME/METHOD(9)
C     STORE THE NAMES OF THE METHODS FOR HEADINGS IN SENSE AND ANSWER
      DATA (METHOD(I),I=1,9)/6HSIMPLE,5HSEEK1,5HSEEK2,5HSEEK3,4HALTS,6HA
     1PPROX,6HRANDOM,4HGEOM,6HADRANS/
C     SUBROUTINE OPTIPAC IS ESSENTIALLY AN EXTENSION OF THE SMALL USER-
C     WRITTEN MAIN PROGRAM. IT PERFORMS THE FOLLOWING FUNCTIONS...
C     1. IT CALLS SUBR.DATA TO READ IN ALL NECESSARY DATA
C     2. IT ASSIGNS VALUES TO CERTAIN PARAMETERS AT LEVEL=0
C     3. IT CALLS THE REQUESTED METHOD SUBROUTINE
C     4. IT COMPUTES THE NET EXECUTION TIME FOR THE METHOD AND PRINTS IT
C        OUT
C     5. AFTER A NORMAL EXIT FROM A METHOD SUBROUTINE IT CALLS SUBR.
C        SENSE TO PERFORM A SENSITIVITY ANALYSIS ON THE SOLUTION
    1 CONTINUE
C     INITIALIZE THE EXIT MODE FLAG KO
      KO=0
C     CALL SUBR.DATA TO READ IN ALL NECESSARY DATA FOR THE METHOD CHOSEN
      CALL DATA (N,NCONS,NEQUS,M,NTOTER,RMAX,RMIN,XSTRT,GS,STEP,DELX,TES
     1T,WATE,NTERMS,EX,CONST,B,C,A,NSENSE,FSENSE)
C     THE STOPPING CRITERION IS INDEX=99 SO EVERY COMPLETE DATA DECK
C     SHOULD END WITH 099 PUNCHED IN COLUMNS 1,2,AND 3
      IF(INDEX.EQ.99) RETURN
      IF(KO.EQ.1)RETURN
C     IF KO=1 AFTER CALL TO DATA, THERE IS NO POINT CONTINUING WITH THE
C     RUN BECAUSE SEVERAL READ STATEMENTS WILL HAVE BEEN SKIPPED AND THE
C     GENERAL DATA SEQUENCE IS NOW SHIFTED OUT OF PHASE
C     ZERO  U  AND CLEAR THE X(I) ARRAY AND ALL COMMON BLOCK WORKING
C     ARRAYS BEFORE CALLING A NEW METHOD
      U=0.0
      DO 2 I=1,N
      X(I)=0.0
    2 CONTINUE
      DO 4 I=1,100
```

```
      WORK1(I)=0.0
      WORK2(I)=0.0
      WORK3(I)=0.0
      WORK4(I)=0.0
      WORK5(I)=0.0
      WORK6(I)=0.0
      WORK9(I)=0.0
      WORK10(I)=0.0
      WORK11(I)=0.0
      WORK12(I)=0.0
      WORK13(I)=0.0
      WORK14(I)=0.0
      WORK15(I)=0.0
      WORK16(I)=0.0
      WORK18(I)=0.0
      WORK19(I)=0.0
    4 IWORK1(I)=0
C     CALL SUBR.SECOND TO GET THE STARTING EXECUTION TIME FOR THE METHOD
      CALL SECOND(START)
      IF(LEVEL.EQ.0.AND.IDATA.EQ.1)WRITE(6,300)
C     GO TO THE PART OF OPTIPAC WHICH SETS PARAMETERS FOR LEVEL=0 AND
C     CALLS THE REQUESTED METHOD SUBROUTINE
    3 JACK=INDEX+1
      GO TO (10,11,12,11,14,15,16,17,18),JACK
   10 IF(LEVEL.NE.0) GO TO 110
      NSTOP=4*M+10
      IF(IDATA.EQ.1)WRITE(6,309)NSTOP
  110 CALL SIMPLE(X,U,M,N,A,B,C,WORKA)
      GOTO20
   11 IF(LEVEL.NE.0) GO TO 1111
      F=.01
      MAXM=300
      G=.01
      IF(INDEX.EQ.1)NSHOT=1
      IF(INDEX.EQ.1)NTEST=100
C     NOTE... AVOID ZERO STARTING VALUES BY ADDING A SMALL INCREMENT
      DO 211 I=1,N
  211 XSTRT(I)=(RMAX(I)+RMIN(I))/2. +0.000001
      IF(IDATA.NE.1)GOTO1111
      WRITE(6,303)F
      WRITE(6,304)MAXM
      WRITE(6,305)G
      IF(INDEX.EQ.1)WRITE(6,312)NSHOT
      IF(INDEX.EQ.1)WRITE(6,313)NTEST
      WRITE(6,319)(XSTRT(I),I=1,N)
 1111 IF(INDEX.NE.3)GOTO111
      IF(LEVEL.NE.0)GOTO1112
      R=1.0
      REDUCE=0.04
      IF(IDATA.NE.1)GOTO1112
      WRITE(6,337)R
      WRITE(6,338)REDUCE
 1112 CALL       SEEK3(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DST
     1AR,NTERMS,NTOTER)
```

```
      GOTO20
  111 CALL SEEK1(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DSTAR,NT
     1ERMS,NTOTER)
      GO TO 20
   12 IF(LEVEL.NE.0) GO TO 112
      F=1.0E-06
      MAXM=50
      PD=0.75
      EPS=1.0E-8
      ICT=4
      IFENCE=0
      PL=1.3
      DO 212 I=1,N
      XSTRT(I)=(RMAX(I)+RMIN(I))/2. + 0.000001
  212 GS(I)=15.0
      IF(IDATA.NE.1)GOTO112
      WRITE(6,303)F
      WRITE(6,304)MAXM
      WRITE(6,307)PD
      WRITE(6,332)EPS
      WRITE(6,333)ICT
      WRITE(6,334)IFENCE
      WRITE(6,308)PL
      WRITE(6,319)(XSTRT(I),I=1,N)
      WRITE(6,320)(GS(I),I=1,N)
  112 CALLSEEK2(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,GS)
      GO TO 20
   14 IF(LEVEL.NE.0) GO TO 114
      F=0.01
      MAXM=300
      G=0.01
      PL=1.5
      NSTOP=4*M+10
      NSMAX=40
      TES=0.0001
      DO 214 I=1,N
      XSTRT(I)=(RMAX(I)+RMIN(I))/2.0+.000001
      STEP(I)=0.10*ABS(RMAX(I)-RMIN(I))
  214 DELX(I)=.001*ABS(RMAX(I)-RMIN(I))
      IF(NEQUS.EQ.0)GOTO2215
      DO 2214 I=1,NEQUS
 2214 WATE(I)=10.0E+20
 2215 IF(IDATA.NE.1)GOTO114
      WRITE(6,303)F
      WRITE(6,304)MAXM
      WRITE(6,305)G
      WRITE(6,308)PL
      WRITE(6,309)NSTOP
      WRITE(6,310)NSMAX
      WRITE(6,315)TES
      WRITE(6,319)(XSTRT(I),I=1,N)
      WRITE(6,321)(STEP(I),I=1,N)
      WRITE(6,322)(DELX(I),I=1,N)
      IF(NEQUS.GT.0)WRITE(6,324)(WATE(I),I=1,NEQUS)
```

```
114 CALL ALTS(X,U,N,XSTRT,RMAX,RMIN,WATE,STEP,NEQUS,NCONS,PSI,PHI,M,NN
   1,A,B,C,WORKA,DSTAR,NTERMS,NTOTER,DELX,XX)
    GO TO20
 15 CONTINUE
    IF(LEVEL.NE.0)GOTO115
    F=0.01
    NSTOP=4*M+10
    NSMAX=40
    DO215 I=1,N
    XSTRT(I)=(RMAX(I)+RMIN(I))/2. + 0.000001
    STEP(I)=0.1*ABS(RMAX(I)-RMIN(I))
    DELX(I)=0.001*ABS(RMAX(I)-RMIN(I))
215 TEST(I)=0.001*ABS(RMAX(I)-RMIN(I))
    IF(IDATA.NE.1)GOTO115
    WRITE(6,303)F
    WRITE(6,309)NSTOP
    WRITE(6,310)NSMAX
    WRITE(6,319)(XSTRT(I),I=1,N)
    WRITE(6,321)(STEP(I),I=1,N)
    WRITE(6,322)(DELX(I),I=1,N)
    WRITE(6,323)(TEST(I),I=1,N)
115 CALL APPROX(X,U,N,DELX,STEP,TEST,M,NN,A,B,C,WORKA,XSTRT,RMAX,RMIN,
   1PHI,PSI,NCONS,NEQUS,UART,DSTAR,NTERMS,NTOTER,XX)
    GO TO 20
 16 IF(LEVEL.NE.0) GO TO 116
    F=.001
    MAXM=400
    MSTART=128
    NSHRIN=4
    NSMAX=300
    IF(IDATA.NE.1)GOTO116
    WRITE(6,303)F
    WRITE(6,304)MAXM
    WRITE(6,353)MSTART
    WRITE(6,352)NSHRIN
    WRITE(6,310)NSMAX
116 CALL RANDOM(X,U,N,RMAX,RMIN,Z,UU,NRET,NCONS,PHI)
    GO TO 20
 17 IF(LEVEL.NE.0)GOTO117
    F=0.01
    MAXM=300
    G=0.001
    IF(IDATA.NE.1)GOTO117
    WRITE(6,303)F
    WRITE(6,304)MAXM
    WRITE(6,305)G
117 CALL GEOM(NTOTER,N,NCONS,NTERMS,EX,CONST,AA,BBB,CC,DSTAR,RMAX,RMIN
   1,X,XSTRT)
    GOTO20
 18 IF(LEVEL.NE.0) GO TO 118
    MAXM=75
    MSTART=128
    NSMAX=50
    DO 218I=1,N
```

```
  218 XSTRT(I)=(RMAX(I)+RMIN(I))/2. + 0.000001
      IF(IDATA.NE.1)GOTO118
      WRITE(6,304)MAXM
      WRITE(6,353)MSTART
      WRITE(6,310)NSMAX
      WRITE(6,319)(XSTRT(I),I=1,N)
  118 CALL ADRANS(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,UART,NCONS,NEQUS,DSTAR,N
     1TOTER,NTERMS)
C     CALL SUBR.SECOND TO GET THE FINAL TIME FOR THE METHOD AND COMPUTE
C     THE NET EXECUTION TIME AND PRINT IT OUT
   20 CALL SECOND(FINISH)
      T=FINISH-START
      WRITE(6,104)T
      IF(INDEX.EQ.0.OR.INDEX.EQ.7)GOTO1
      IF(KO.EQ.0) GO TO 22
      IF(NSENSE.EQ.1)WRITE(6,100)
      GO TO 1
C     SENSITIVITY ANALYSIS IS PERFORMED ONLY AFTER A NORMAL EXIT(KO=0)
C     FROM THE METHOD SUBROUTINE,  AND  WHEN   THE  WORD  SENSITIVITY
C     APPEARS IN COLUMNS 13 TO 23 ON THE FIRST DATA CARD FOR THAT METHOD
   22 IF(NSENSE.NE.1)GOTO1
      IF(FSENSE.LE.0.0)GOTO23
      CALL SENSE(X,N,NCONS,NEQUS,FSENSE,INDEX)
      GOTO1
C     USER HAS NOT ENTERED A VALUE FOR FSENSE ON THE (NCONS) DATA CARD
   23 WRITE(6,101)
      GOTO1
  100 FORMAT(62H0     ERROR IN RESULTS SO SENSITIVITY ANALYSIS IS NOT PER
     1FORMED)
  101 FORMAT(1H-,92HERROR***SENSITIVITY ANALYSIS OMITTED - NO VALUE FOR
     1 FSENSE ENTERED ON THE (NCONS) DATA CARD//)
  104 FORMAT(1H-,14X,17HEXECUTION  TIME =,F8.4,9H  SECONDS//)
  300 FORMAT(1H-,6X,67HTHE FOLLOWING PARAMETERS ARE ASSIGNED VALUES INTE
     1RNALLY FOR LEVEL=0/7X,67H-----------------------------------------
     2--------------------------/)
  301 FORMAT(61H0NUMBER OF INDEPENDENT VARIABLES . . . . . . . . . .
     1 N   =,I6)
  302 FORMAT(61H0NUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . . .   NCO
     1NS   =,I6)
  303 FORMAT(61H0FRACTION OF RANGE USED AS STEP SIZE . . . . . . . .
     1 F   =,E19.8)
  304 FORMAT(61H0MAXIMUM NUMBER OF MOVES PERMITTED . . . . . . . . .    MA
     1XM   =,I6)
  305 FORMAT(61H0STEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
     1 G   =,E19.8)
  306 FORMAT(61H0NUMBER OF EQUALITY CONSTRAINTS. . . . . . . . . . .   NEQ
     1US   =,I6)
  307 FORMAT(61H0STEP LENGTH MULTIPLIER FOR INITIAL PATTERN MOVE .
     1PD   =,E19.8)
  308 FORMAT(61H0ACCELERATION FACTOR FOR PATTERN MOVE STEP SIZES .
     1PL   =,E19.8)
  309 FORMAT(61H0NUMBER OF ITERATIONS PERMITTED. . . . . . . . . . .   NST
     1OP   =,I6)
  310 FORMAT(61H0MAXIMUM NUMBER OF LINEARIZED STEPS. . . . . . . . .   NSM
     1AX   =,I6)
```

```
312 FORMAT(61H0NUMBER OF SHOTGUN SEARCHES PERMITTED. . . . . . .   NSH
   1OT  =,I6)
313 FORMAT(61H0NUMBER OF TEST POINTS IN SHOTGUN SEARCH . . . . .   NTE
   1ST  =,I6)
314 FORMAT(61H0NUMBER OF CONSTRAINT EQUATIONS (ROWS) IN SIMPLEX.
   1 M  =,I6)
315 FORMAT(61H0CONVERGENCE CRITERION FOR OPTIMIZATION FUNCTION .     T
   1ES  =,E19.8)
316 FORMAT(61H0TOTAL NUMBER OF TERMS IN ALL RELATIONS. . . . . .  NTOT
   1ER  =,I6)
317 FORMAT(61H0ESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . . RMAX(
   1I)  =,//(5E16.8))
318 FORMAT(61H0ESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . . RMIN(
   1I)  =,//(5E16.8))
319 FORMAT(61H0STARTING VALUES OF X(I) . . . . . . . . . . . . .XSTRT(
   1I)  =,//(5E16.8))
320 FORMAT(61H0STEP LENGTH MULTIPLIERS FOR UNIVARIABLE SEARCH. .   GS(
   1I)  =,//(5E16.8))
321 FORMAT(61H0INITIAL STEP SIZE INPUT BY USER . . . . . . . . . STEP(
   1I)  =,//(5E16.8))
322 FORMAT(61H0INCREMENTS FOR APPROXIMATING PARTIAL DERIVATIVES. DELX(
   1I)  =,//(5E16.8))
323 FORMAT(61H0LOWER BOUND ON STEP LENGTH REDUCTION. . . . . . . TEST(
   1I)  =,//(5E16.8))
324 FORMAT(61H0WEIGHTING FACTORS . . . . . . . . . . . . . . . . WATE(
   1I)  =,//(5E16.8))
326 FORMAT(61H0NUMBER OF TERMS IN EACH RELATION. . . . . . . . NTERMS(
   1I)  =,//(5E16.8))
327 FORMAT(61H0EXPONENTS OF EACH TERM IN EACH RELATION . . . . . EX(I,
   1J)  =,//(5E16.8))
328 FORMAT(61H0CONSTANT (POSITIVE) COEFFICIENTS OF EACH TERM . .CONST(
   1J)  =,//(5E16.8))
329 FORMAT(61H0RIGHT HAND SIDE OF SIMPLEX ARRAY. . . . . . . . .    B(
   1M)  =,//(5E16.8))
330 FORMAT(61H0COEFFICIENTS OF SIMPLEX OBJECTIVE FUNCTION. . . .    C(
   1N)  =,//(5E16.8))
331 FORMAT(61H0COEFFICIENTS OF SIMPLEX CONSTRAINT EQUATIONS. . .  A(M,
   1N)  =,//(5E16.8))
332 FORMAT(61H0MAX. RELATIVE CHANGE IN U FOR CONVERGENCE . . . .     E
   1PS  =,E19.8)
333 FORMAT(61H0NO. OF TIMES STEP SIZE DIVIDED BY 10.0  . . . . .     I
   1CT  =,I6)
334 FORMAT(61H0OPTION TO STOP AFTER UNIVARIABLE SEARCH FAILS . .  IFEN
   1CE  =,I6)
337 FORMAT(61H0PENALTY MULTIPLIER USED IN SEEK3. . . . . . . . .
   1 R  =,E19.8)
338 FORMAT(61H0REDUCTION FACTOR FOR (R) AFTER EACH MINIMIZATION.  REDU
   1CE  =,E19.8)
352 FORMAT(61H0SHRINKAGE FACTOR. . . . . . . . . . . . . . . . .  NSHR
   1IN  =,I6)
353 FORMAT(61H0STARTING VALUE FOR RANDOM NUMBERS . . . . . . . .  MSTA
   1RT  =,I6)
    END
```

```
      SUBROUTINE SENSE(X,N,NCONS,NEQUS,FSENSE,INDEX)
      DIMENSION X(1)
      COMMON /NAME/METHOD(9)
      COMMON /A3/XTEMP(100),ABOVE(100),BELOW(100)
      WRITE(6,1)METHOD(INDEX+1)
      WRITE(6,8)FSENSE
C     IN THE FOLLOWING SENSITIVITY ANALYSIS, EACH VARIABLE IN TURN IS
C     MULTIPLIED BY THE FACTORS (1.+FSENSE) AND (1.-FSENSE) AND ALL THE
C     CONSTRAINTS ARE EVALUATED AT EACH POINT.
C     STORE THE OPTIMUM VALUES OF X(I) IN XTEMP(I)
      DO 10 I=1,N
   10 XTEMP(I)=X(I)
      DO 50 I=1,N
      X(I)=(1.-FSENSE)*XTEMP(I)
      WRITE(6,2)I
      WRITE(6,3)I,X(I)
      CALL UREAL(X,ULESS)
      IF(NCONS.EQ.0)GOTO20
      CALL CONST(X,NCONS,BELOW)
   20 X(I)=(1.+FSENSE)*XTEMP(I)
      WRITE(6,4)I,X(I)
      CALL UREAL(X,UMORE)
      WRITE(6,5)ULESS,UMORE
      IF(NCONS.EQ.0)GOTO30
      CALL CONST(X,NCONS,ABOVE)
      WRITE(6,6)(J,BELOW(J),ABOVE(J),J=1,NCONS)
   30 IF(NEQUS.EQ.0)GOTO40
      CALL EQUAL(X,ABOVE,NEQUS)
      X(I)=(1.-FSENSE)*XTEMP(I)
      CALL EQUAL(X,BELOW,NEQUS)
      WRITE(6,7)(J,BELOW(J),ABOVE(J),J=1,NEQUS)
   40 X(I)=XTEMP(I)
   50 CONTINUE
    1 FORMAT(1H-,45HSENSITIVITY ANALYSIS OF THE OPTIMUM FOUND BY ,A6/1X,
     151H---------------------------------------------------//)
    2 FORMAT(1H-,23X,10HVARYING X(,I2,6H) ONLY/24X,18H------------------
     1//)
    3 FORMAT(1H+,2X,2HX(,I2,3H) =,E18.8)
    4 FORMAT(31X,2HX(,I2,3H) =,E16.8)
    5 FORMAT(1H0,6X,3HU =,E18.8,10X,E16.8/)
    6 FORMAT(1X,4HPHI(,I2,3H) =,E18.8,10X,E16.8)
    7 FORMAT(1H0/1X,4HPSI(,I2,3H) =,E18.8,10X,E16.8)
    8 FORMAT(1H0,52HFRACTION OF OPTIMUM X(I) USED AS INCREMENT, FSENSE =
     1,E16.8//)
      RETURN
      END
```

```
      SUBROUTINE ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
      DIMENSION X(1),PHI(1),PSI(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /NAME/METHOD(9)
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)METHOD(INDEX+1)
      WRITE(6,19)U
      GOTO2
    1 WRITE(6,20)METHOD(INDEX+1)
      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,30HRESULTS AT LAST ITERATION OF   ,A6/17X,36H-------
     1-----------------------------/)
   19 FORMAT(29X,3HU =,E16.8//)
   20 FORMAT(1H1,21X,27HOPTIMUM SOLUTION FOUND BY  ,A6/22X,33H----------
     1-----------------------/)
   21 FORMAT(20X,12HMINIMUM  U =,E16.8//)
   22 FORMAT(25X,2HX(,I2,3H) =,E16.8)
   23 FORMAT(1H-,22HINEQUALITY CONSTRAINTS)
   24 FORMAT(23X,4HPHI(,I2,3H) =,E16.8)
   25 FORMAT(1H-,22H  EQUALITY CONSTRAINTS)
   26 FORMAT(23X,4HPSI(,I2,3H) =,E16.8)
   30 RETURN
      END


      SUBROUTINE DATA(N,NCONS,NEQUS,M,NTOTER,RMAX,RMIN,XSTRT,GS,STEP,DEL
     1X,TEST,WATE,NTERMS,EX,CONST,B,C,A,NSENSE,FSENSE)
      DIMENSION RMAX(1),RMIN(1),XSTRT(1),GS(1),STEP(1),DELX(1),TEST(1),
     1WATE(1),NTERMS(1),EX(NTOTER,1),CONST(1),B(1),C(1),A(M,1),TITLE(17)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /NAME/METHOD(9)
C
C
C     THE FIRST DATA CARD (INDEX,LEVEL,IPRINT,IDATA) MAY CONTAIN A HEAD-
C     ING STARTING IN COLUMN 13 AND ENDING IN OR BEFORE COLUMN 80
C
      READ(5,100)INDEX,LEVEL,IPRINT,IDATA,(TITLE(I),I=1,17)
      IF(INDEX.EQ.99)RETURN
      WRITE(6,241)(TITLE(I),I=1,17)
```

```
C     CHECK TO SEE IF SENSITIVITY ANALYSIS HAS BEEN REQUESTED
      NSENSE=0
      IF(TITLE(1).EQ.4HSENS.AND.TITLE(2).EQ.4HITIV)NSENSE=1
C     SENSITIVITY ANALYSIS IS NOT AVAILABLE TO SIMPLE OR GEOM
      IF(INDEX.EQ.0.OR.INDEX.EQ.7)NSENSE=0
      IF(IDATA.NE.1)GOTO599
      WRITE(6,240)METHOD(INDEX+1)
      WRITE(6,197)INDEX
      WRITE(6,198)LEVEL
      WRITE(6,199)IPRINT
      WRITE(6,200)IDATA
  599 CONTINUE
C
C     CHECK THAT VALUES OF IDATA AND LEVEL ARE ACCEPTABLE
C
      IF(LEVEL.GT.1.OR.LEVEL.LT.0)GO TO 600
      IF(IDATA.GT.1.OR.IDATA.LT.0)GO TO 601
      GO TO 602
  600 WRITE(6,235)
      KO=1
      RETURN
  601 WRITE(6,236)
      KO=1
      RETURN
  602 CONTINUE
C
C     CONTROL RETURNED TO OPTIPAC IF INDEX OUTSIDE RANGE 0.LE.INDEX.LE.8
C
      IF(INDEX.LE.8.OR.INDEX.GE.0)GO TO 603
      IF(INDEX.EQ.99)RETURN
      WRITE(6,242)INDEX
      KO=1
      RETURN
  603 IF(INDEX.EQ.0.AND.LEVEL.EQ.1)GO TO 13
      IF(INDEX.EQ.0)GO TO 15
      IF(NSENSE.EQ.1)GOTO604
C
C     NCONS READ FOR INDEX=1,2,3,4,5,6,7,8
C
      READ(5,101)NCONS
      IF(IDATA.EQ.1)WRITE(6,202)NCONS
      GOTO605
C
C     NCONS,FSENSE READ FOR INDEX=1,2,3,4,5,6,8   WHEN NSENSE=1
C
C     IF SENSITIVITY ANALYSIS HAS BEEN REQUESTED (NSENSE=1) THEN THE
C     FRACTIONAL INCREMENT FSENSE APPEARS ON THE SAME CARD AS NCONS. THE
C     FORMAT IS (I5,E16.8)
  604 READ(5,107)NCONS,FSENSE
      IF(IDATA.EQ.1)WRITE(6,202)NCONS
      IF(IDATA.EQ.1)WRITE(6,254)FSENSE
  605 IF(INDEX.EQ.8.AND.LEVEL.EQ.0)GO TO 11
      IF(INDEX.EQ.8)GO TO 9
      IF(INDEX.EQ.7.AND.LEVEL.EQ.0)GO TO 22
```

```
      IF(INDEX.EQ.6.AND.LEVEL.EQ.0)GO TO 18
      IF(LEVEL.EQ.0)GO TO 11
C
C      F     READ FOR INDEX=1,2,3,4,5,6,7   WHEN LEVEL =1
C
      READ(5,104)F
      IF(IDATA.EQ.1)WRITE(6,203)F
      IF(INDEX.EQ.5)GO TO 11
    9 CONTINUE
C
C     MAXM   READ FOR INDEX=1,2,3,4,6,7,8   WHEN LEVEL=1
C
      READ(5,102)MAXM
      IF(IDATA.EQ.1)WRITE(6,204)MAXM
      IF(INDEX.EQ.6)GO TO 48
      IF(INDEX.EQ.8.OR.INDEX.EQ.2)GO TO 11
   10 CONTINUE
C
C     G      READ FOR INDEX=1,3,4,7       WHEN LEVEL =1
C
      READ(5,104)G
      IF(IDATA.EQ.1)WRITE(6,205)G
      IF(INDEX.EQ.7)GO TO 22
   11 CONTINUE
C
C     NEQUS READ FOR INDEX=1,2,3,4,5,8
C
      READ(5,101)NEQUS
      IF(IDATA.EQ.1)WRITE(6,206)NEQUS
      IF(INDEX.EQ.8.AND.LEVEL.EQ.1)GOTO48
      IF(LEVEL.EQ.1)GOTO(14,50,18,12,13),INDEX
      IF(INDEX.EQ.5)GOTO51
      GO TO 18
C
C     MSTART  READ FOR INDEX=6,8  WHEN LEVEL=1
C
   48 IF(LEVEL.EQ.0)GOTO52
      READ(5,102)MSTART
      IF(IDATA.EQ.1)WRITE(6,253)MSTART
      IF(INDEX.EQ.8)GOTO52
C
C     NSHRIN   READ FOR INDEX=6  WHEN LEVEL=1
C
      READ(5,102)NSHRIN
      IF(IDATA.EQ.1)WRITE(6,252)NSHRIN
      GOTO52
   50 CONTINUE
C
C     PD     READ FOR INDEX=2   WHEN LEVEL =1
C
      READ(5,104)PD
      IF(IDATA.EQ.1)WRITE(6,207)PD
C
```

```
C     EPS,ICT,IFENCE READ FORINDEX=2  WHEN LEVEL=1
C
      READ(5,104)EPS
      IF(IDATA.EQ.1)WRITE(6,232)EPS
      READ(5,101)ICT
      IF(IDATA.EQ.1)WRITE(6,233)ICT
      READ(5,101)IFENCE
      IF(IDATA.EQ.1)WRITE(6,234)IFENCE
   12 CONTINUE
C
C     PL      READ FOR INDEX= 2,4             WHEN LEVEL=1
C
      READ(5,104)PL
      IF(IDATA.EQ.1)WRITE(6,208)PL
      IF(INDEX.EQ.2)GO TO 18
   13 CONTINUE
C
C     NSTOP   READ FOR INDEX= 0,4,5           WHEN LEVEL=1
C
      READ(5,102)NSTOP
      IF(IDATA.EQ.1)WRITE(6,209)NSTOP
      IF(INDEX.EQ.0)GO TO 15
   52 CONTINUE
C
C     NSMAX   READ FOR INDEX= 4,5,6,8           WHEN LEVEL=1
C
      READ(5,102)NSMAX
      IF(IDATA.EQ.1.AND.INDEX.NE.6)WRITE(6,210)NSMAX
      IF(IDATA.EQ.1.AND.INDEX.EQ.6)WRITE(6,244)NSMAX
      IF(INDEX.EQ.6.OR.INDEX.EQ.8)GO TO 18
      IF(INDEX.EQ.4)GO TO 16
   51 CONTINUE
      GO TO 18
   14 CONTINUE
C
C     NSHOT   READ FOR INDEX= 1               WHEN LEVEL=1
C
      READ(5,102)NSHOT
      IF(IDATA.EQ.1)WRITE(6,212)NSHOT
C
C     NTEST   READ FOR INDEX= 1               WHEN LEVEL=1
C
      READ(5,102)NTEST
      IF(IDATA.EQ.1)WRITE(6,213)NTEST
      GO TO 18
   15 CONTINUE
      GO TO 23
   16 CONTINUE
C
C     TES     READ FOR INDEX =4  WHEN LEVEL =1
C
      READ(5,104)TES
      IF(IDATA.EQ.1)WRITE(6,215)TES
```

```
   18 CONTINUE
C
C     R, REDUCE    READ FOR INDEX=3
C
      IF(INDEX.NE.3)GOTO609
      IF(LEVEL.EQ.0)GOTO609
      READ(5,104)R
      READ(5,104)REDUCE
      IF(IDATA.EQ.1)WRITE(6,237)R
      IF(IDATA.EQ.1)WRITE(6,238)REDUCE
C
C     RMAX,RMIN  READ FOR INDEX= 1,2,3,4,5,6,8
C
C     NOTE ALL SUBSCRIPTED VARIABLES ARE ZEROED IMMEDIATELY BEFORE THEY
C     ARE READ
C
  609 DO 610 J=1,N
      RMAX(J)=0.
      RMIN(J)=0.
  610 CONTINUE
      READ(5,105)(RMAX(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,217)(RMAX(I),I=1,N)
      READ(5,105)(RMIN(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,218)(RMIN(I),I=1,N)
C
      IF(LEVEL.NE.1)GO TO 24
      IF(INDEX.EQ.6)GO TO 24
C
C     XSTRT  READ FOR INDEX= 1,2,3,4,5,8    WHEN LEVEL=1
C
      DO 611 J=1,N
  611 XSTRT(J)=0.
      READ(5,105)(XSTRT(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,219)(XSTRT(I),I=1,N)
      IF(INDEX.EQ.2)GO TO 19
      IF(INDEX.EQ.4)GO TO 20
      IF(INDEX.EQ.5)GO TO 20
      GO TO 24
   19 CONTINUE
C
C     GS     READ FOR INDEX= 2              WHEN LEVEL=1
C
      DO 612 J=1,N
  612 GS(J)=0.
      READ(5,105)(GS(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,220)(GS(I),I=1,N)
      GO TO 24
   20 CONTINUE
C
C     STEP   READ FOR INDEX= 4,5            WHEN LEVEL=1
C
      DO 613 J=1,N
  613 STEP(J)=0.
```

```
      IF(IDATA.EQ.1)WRITE(6,221)(STEP(I),I=1,N)
C
C     DELX   READ FOR INDEX= 4,5                    WHEN LEVEL=1
C
      DO 614 J=1,N
      DELX(J)=0.
  614 TEST(J)=0.
      READ(5,105)(DELX(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,222)(DELX(I),I=1,N)
      IF(INDEX.EQ.4)GO TO 21
C
C     TEST   READ FOR INDEX= 5                    WHEN LEVEL=1
      READ(5,105)(TEST(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,223)(TEST(I),I=1,N)
      GO TO 24
   21 CONTINUE
C
C     WATE   READ FOR INDEX = 4  WHEN NEQUS.GT.0 AND  LEVEL=1
C
      IF(NEQUS.EQ.0)GOTO24
      DO 615 J=1,NEQUS
  615 WATE(J)=0.
      READ(5,105)(WATE(I),I=1,NEQUS)
      IF(IDATA.EQ.1)WRITE(6,224)(WATE(I),I=1,NEQUS)
      GO TO 24
   22 CONTINUE
C
C     DATA FOR GEOM  INDEX=7
C
C     NTERMS,EX,CONST,  READ  FOR INDEX = 7
C
C     NTERMS(J)=NO. OF TERMS IN EACH RELATION
C     EX(I,J)  =EXPTS FOR EACH TERM OF EACH RELATION
C
C     CONST(J) =CONSTANT COEFFICIENTS OF EACH TERM
C     NO.OF VARIABLES=N,NO.OF CONSTRAINTS=NCONS
C
      NT=NCONS+1
C
C     NO.OF TERMS IN EACH RELATION=NTERMS(NT)
C
      DO 616 J=1,NT
  616 NTERMS(J)=0
      READ(5,106) (NTERMS(J),J=1,NT)
      IF(IDATA.EQ.1)WRITE(6,226)(NTERMS(J),J=1,NT)
C
C     NTOTER=TOTAL NO.OF TERMS
C
      NCHEK=0
      DO 500 J=1,NT
C
C     CHECK USERS ESTIMATE OF NTOTER
C
  500 NCHEK=NCHEK+NTERMS(J)
```

```
      IF(NTOTER.EQ.NCHEK)GOTO498
      KO=1
      WRITE(6,255)NCHEK
      GO TO 24
C
C     EX(NTOTER,N)=EXPONENTS FOR EACH TERM OF EACH RELATION
C
  498 DO 617 J=1,N
      DO 617 I=1,NTOTER
  617 EX(I,J)=0.
      READ(5,105)((EX(I,J),J=1,N),I=1,NTOTER)
      IF(IDATA.EQ.1)WRITE(6,227)((EX(I,J),J=1,N),I=1,NTOTER)
C
C     CONST(NTOTER)=CONSTANTS FOR EACH RELATIONSHIP
C
      DO 618 J=1,NTOTER
  618 CONST(J)=0.
      READ(5,105) (CONST(J),J=1,NTOTER)
      IF(IDATA.EQ.1)WRITE(6,228)(CONST(J),J=1,NTOTER)
      GO TO 24
C
   23 CONTINUE
C
C     B,C,A  READ FOR INDEX= 0
C
      DO 619 I=1,M
      DO 619 J=1,N
      B(J)=0.
      C(I)=0.
  619 A(I,J)=0.
      READ(5,105)(B(J),J=1,M)
      IF(IDATA.EQ.1)WRITE(6,229)(B(J),J=1,M)
      READ(5,105)(C(I),I=1,N)
      IF(IDATA.EQ.1)WRITE(6,230)(C(I),I=1,N)
      READ(5,105)((A(I,J),J=1,N),I=1,M)
      IF(IDATA.EQ.1)WRITE(6,231)((A(I,J),J=1,N),I=1,M)
   24 CONTINUE
  100 FORMAT(4I3,17A4)
  101 FORMAT(I5)
  102 FORMAT(I6)
  103 FORMAT(I3)
  104 FORMAT(E16.8)
  105 FORMAT(5E16.8)
  106 FORMAT(16I5)
  107 FORMAT(I5,E16.8)
  197 FORMAT(61H0INDEX NUMBER OF METHOD USED . . . . . . . . . . . .    IND
     1EX  =,I6)
  198 FORMAT(61H0USERS LEVEL OF SOPHISTICATION . . . . . . . . . .    LEV
     1EL  =,I6)
  199 FORMAT(61H0INTERMEDIATE OUTPUT EVERY IPRINT(TH) CYCLE. . . .   IPRI
     1NT  =,I6)
  200 FORMAT(61H0INPUT DATA IS PRINTED OUT FOR IDATA=1 ONLY. . . .    IDA
     1TA  =,I6)
```

```
202 FORMAT(61HONUMBER OF INEQUALITY (.GE.) CONSTRAINTS . . . . .    NCO
   1NS   =,I6)
203 FORMAT(61HOFRACTION OF RANGE USED AS STEP SIZE . . . . . . . .
   1 F   =,E19.8)
204 FORMAT(61HOMAXIMUM NUMBER OF MOVES PERMITTED . . . . . . . . .   MA
   1XM   =,I6)
205 FORMAT(61HOSTEP SIZE FRACTION USED AS CONVERGENCE CRITERION.
   1 G   =,E19.8)
206 FORMAT(61HONUMBER OF EQUALITY CONSTRAINTS. . . . . . . . . . .  NEQ
   1US   =,I6)
207 FORMAT(61HOSTEP LENGTH MULTIPLIER FOR INITIAL PATTERN MOVE .
   1PD   =,E19.8)
208 FORMAT(61HOACCELERATION FACTOR FOR PATTERN MOVE STEP SIZES .
   1PL   =,E19.8)
209 FORMAT(61HONUMBER OF ITERATIONS PERMITTED. . . . . . . . . . .  NST
   1OP   =,I6)
210 FORMAT(61HOMAXIMUM NUMBER OF LINEARIZED STEPS. . . . . . . . .  NSM
   1AX   =,I6)
212 FORMAT(61HONUMBER OF SHOTGUN SEARCHES PERMITTED. . . . . . . .  NSH
   1OT   =,I6)
213 FORMAT(61HONUMBER OF TEST POINTS IN SHOTGUN SEARCH . . . . . .  NTE
   1ST   =,I6)
215 FORMAT(61HOCONVERGENCE CRITERION FOR OPTIMIZATION FUNCTION .      T
   1ES   =,E19.8)
217 FORMAT(61HOESTIMATED UPPER BOUND ON RANGE OF X(I). . . . . . . RMAX(
   1I)   =,//(5E16.8))
218 FORMAT(61HOESTIMATED LOWER BOUND ON RANGE OF X(I). . . . . . . RMIN(
   1I)   =,//(5E16.8))
219 FORMAT(61H-STARTING VALUES OF X(I) . . . . . . . . . . . . . .XSTRT(
   1I)   =,//(5E16.8))
220 FORMAT(61HOSTEP LENGTH MULTIPLIERS FOR UNIVARIABLE SEARCH. .    GS(
   1I)   =,//(5E16.8))
221 FORMAT(61HOINITIAL STEP SIZE INPUT BY USER . . . . . . . . . . STEP(
   1I)   =,//(5E16.8))
222 FORMAT(61HOINCREMENTS FOR APPROXIMATING PARTIAL DERIVATIVES. DELX(
   1I)   =,//(5E16.8))
223 FORMAT(61HOLOWER BOUND ON STEP LENGTH REDUCTION. . . . . . . . TEST(
   1I)   =,//(5E16.8))
224 FORMAT(61HOWEIGHTING FACTORS . . . . . . . . . . . . . . . . . WATE(
   1I)   =,//(5E16.8))
226 FORMAT(61HONUMBER OF TERMS IN EACH RELATION. . . . . . . . . NTERMS(
   1I)   =,//(16I5))
227 FORMAT(61HOEXPONENTS OF EACH TERM IN EACH RELATION . . . . . . EX(I,
   1J)   =,//(5E16.8))
228 FORMAT(61HOCONSTANT (POSITIVE) COEFFICIENTS OF EACH TERM . .CONST(
   1J)   =,//(5E16.8))
229 FORMAT(61HORIGHT HAND SIDE OF SIMPLEX ARRAY. . . . . . . . . .    B(
   1M)   =,//(5E16.8))
230 FORMAT(61HOCOEFFICIENTS OF SIMPLEX OBJECTIVE FUNCTION. . . .      C(
   1N)   =,//(5E16.8))
231 FORMAT(61HOCOEFFICIENTS OF SIMPLEX CONSTRAINT EQUATIONS. . .    A(M,
   1N)   =,//(5E16.8))
232 FORMAT(61HOMAX. RELATIVE CHANGE IN U FOR CONVERGENCE . . . .      E
   1PS   =,E19.8)
```

```
233 FORMAT(61HONO. OF TIMES STEP SIZE DIVIDED BY 10.0 . . . . . .      I
   1CT  =,I6)
234 FORMAT(61HOOPTION TO STOP AFTER UNIVARIABLE SEARCH FAILS . .  IFEN
   1CE  =,I6)
235 FORMAT(1H-,56HERROR***INPUT VALUE FOR (LEVEL) IS NEGATIVE OR TOO L
   1ARGE/)
236 FORMAT(1H0,78HERROR***VALUE FOR (IDATA) IS INCORRECT,  1 OR 0 ARE
   1THE ONLY ACCEPTABLE VALUES/)
237 FORMAT(61HOPENALTY MULTIPLIER USED IN SEEK3. . . . . . . . . .
   1 R  =,E19.8)
238 FORMAT(61HOREDUCTION FACTOR FOR (R) AFTER EACH MINIMIZATION.  REDU
   1CE  =,E19.8)
240 FORMAT(1H-,20X,33HLISTING OF ALL DATA READ IN FOR  ,A6/21X,39H----
   1-----------------------------------/)
241 FORMAT(1H1,17A4)
242 FORMAT(1H-,28HERROR***THE VALUE OF INDEX =,I6,43H  IS OUTSIDE THE
   1ALLOWABLE RANGE OF  0 TO 8/)
244 FORMAT(61HOMAXIMUM NO. OF CONSECUTIVE INFEASIBLE POINTS. . .   NSM
   1AX  =,I6)
251 FORMAT(1H0,16I6)
252 FORMAT(61HOSHRINKAGE FACTOR. . . . . . . . . . . . . . . . . .  NSHR
   1IN  =,I6)
253 FORMAT(61HOSTARTING VALUE FOR RANDOM NUMBERS . . . . . . . . .  MSTA
   1RT  =,I6)
254 FORMAT(61HOFRACTIONAL INCREMENT FOR SENSITIVITY ANALYSIS . .  FSEN
   1SE  =,E16.8)
255 FORMAT(1H0,80HERROR***USERS ESTIMATE OF (NTOTER) IS INCORRECT - TH
   1E CORRECT VALUE IS  NTOTER =,I6)
    RETURN
    END


      SUBROUTINE SIMPLE(X,U,M,N,A,B,C,E)
      DIMENSION X(1),A(M,1),B(1),C(1),E(M,1),MO(2)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON/A4/P(100),XX(100),Y(100),PE(100)
      COMMON/A8/JH(100)
C
C     SUBROUTINE SIMPLE IS USED PRIMARILY AS A MEANS TO CALCULATE
C     A VALUE OF THE OBJECTIVE FUNCTION AT THE OPTIMUM CONDITIONS
C     OR IF THE SOLUTION IS NOT VALID THIS SUBROUTINE THEN OUTPUTS
C     THE DIAGNOSTIC MESSAGES
C     THE ACTUAL ITERATIVE PROCESS OF THE REVISED SIMPLEX TECHNIQUE IS
C     PERFORMED IN SUBROUTINE SIMP
C
C
      CALL SIMP(M,N,MO,X,E,A,B,C,NSTOP)
C
C     THE FOLLOWING   STATEMENTS  ARE TO DETERMINE THE CONDITION OF THE
C     SOLUTION ON RETURN FROM THE SUBROUTINE SIMP
C
      IF(MO(1).GT.5)GOTO18
      MODE1=MO(1)+1
```

```
      GO TO (21,15,16,15,17,18),MODE1
C
C     NO FEASIBLE SOLUTION CAN BE FOUND FROM THE GIVEN DATA
C
   15 WRITE(6,51)
      GOTO20
C
C     AN UNBOUNDED OPTIMUM HAS BEEN FOUND
C
   16 WRITE(6,52)
      GO TO 20
C
C     THE MAX. NUMBER OF ALLOWABLE ITERATIONS HAS BEEN EXCEEDED
C     THE SOLUTION IS STILL FEASIBLE
C
   17 WRITE(6,53) MO(2)
      GO TO 20
C
C     THE MAX. NUMBER OF ALLOWABLE ITERATIONS HAS BEEN EXCEEDED
C     THE SOLUTION AT THE TIME OF INTERUPTION WAS NOT FEASIBLE
C
   18 WRITE(6,54)MO(2)
   20 KO=1
      GO TO 11
C
C     THE SOLUTION IS VALID --- CALCULATE THE OPTIMIZATION FUNCTION
C     AND OUTPUT THE RESULTS
C
   21 U=0.0
      DO 23 J=1,N
   23 U=U+C(J)*X(J)
C
C     IF THE INDEX DOES NOT EQUAL ZERO THE OUTPUT FROM THE SUBROUTINE
C     SIMPLE IS OMITTED.
C
      IF(INDEX.GT.0) GO TO 11
      WRITE(6,30)
      WRITE(6,31)U
      WRITE(6,32)(I,X(I),I=1,N)
   11 RETURN
   30 FORMAT(1H1,22X,36HOPTIMUM  SOLUTION  FOUND  BY  SIMPLE/23X,36H----
     1-----------------------------------/)
   31 FORMAT(20X,12HMINIMUM  U =,E16.8//)
   32 FORMAT(25X,2HX(,I2,3H) =,E16.8)
   51 FORMAT(1X,44H NO FEASIBLE SOLUTION CAN BE FOUND BY SIMPLE)
   52 FORMAT(1H0,43HTHE SIMPLEX ROUTINE FOUND UNBOUNDED OPTIMUM)
   53 FORMAT(1H0,97HTHE MAXIMUM ALLOWABLE NO OF ITERATIONS FOR SIMPLEX H
     1AS BEEN EXCEEDED,--SOLUTION IS STILL FEASIBLE/1H0,17HNO OF ITERATI
     1ONS=,I5)
   54 FORMAT(1H0,85HNO FEASIBLE SOLUTION EXISTS FOR SIMPLEX-PROGRAM STOP
     1PED ON ALLOWABLE NO OF ITERATIONS/1H0,17HNO OF ITERATIONS=,I5)
      END
```

```
      SUBROUTINE SIMP(M,N,KO,KB,E,A,B,C,NSTOP)
      DIMENSION B(1),C(1),E(1),KO(2),KB(1),A(M,1)
      COMMON /A4/P(100),X(100),Y(100),PE(100)
      COMMON /A8/JH(100)
      EQUIVALENCE (XX,LL)
      LOGICAL FEAS,VER,NEG,TRIG,KQ,ABSC
C
C     THE PURPOSE OF THE SUBROUTINE SIMP IS TO PERFORM THE ITERATIVE
C     METHOD OF LINEAR PROGRAMMING KNOWN AS THE SIMPLEX METHOD
C     SIMP IS A MODIFIED VERSION OF SUBROUTINE SIMPLE IN THE LIBRARY OF
C     THE I.B.M. 7040 COMPUTER AT MCMASTER UNIVERSITY
C
C         SET INITIAL VALUES, SET CONSTANT VALUES
      ITER = 0
      NUMVR = 0
      NUMPV = 0
      TEXP  =   .5**16
C
C     IF LEVEL=0 THE MAXIMUM NUMBER OF ITERATIONS ALLOWED IS SET
C     AUTOMATICALLY AT 4*M+10 IN OPTIPAC..AT LEVEL=1 NSTOP IS READ IN
C     AS DATA. THIS APPLIES FOR INDEX=0,4,5
C
      NCUT=NSTOP
      NVER   =   M/2   +   5
      M2 = M**2
C
C     THE LOGICAL VARIABLE FEAS IS USED TO DETERMINE WHETHER THE
C     SOLUTION IS FEASIBLE OR NOT
C
      FEAS = .FALSE.
C
C* 'NEW'    START  PHASE ONE WITH SINGLETON BASIS
C
C     SELECT THOSE COLUMNS IN A(I,J) WHICH HAVE ONLY ONE NON ZERO
C     COEFFICIENT
C     SET KB(J)=1 (WHERE J= THE COLUMN NUMBER)
C     NOTE THAT IF THE ABOVE CONDITION IS TRUE BUT THE CORRESPONDING A
C     VALUE IS NEGATIVE (IE THERE IS A POSSIBILITY THAT THE NON-
C     NEGATIVITY CONSTRAINT HAS BEEN VIOLATED ) THEN SET KB(J)=0 FOR
C     THAT COLUMN
C
      DO 1402  J = 1,N
        KB(J) = 0
        KQ = .FALSE.
        DO 1403  I = 1,M
          IF (A(I,J).EQ.0.0)  GO TO 1403
          IF (KQ.OR.A(I,J).LT.0.0) GO TO 1402
          KQ = .TRUE.
 1403   CONTINUE
        KB(J) = 1
 1402 CONTINUE
 1400 DO 1401  I = 1,M
        JH (I) = -1
 1401 CONTINUE
C
```

```
C* 'VER'    CREATE INVERSE FROM 'KB' AND 'JH'     (STEP 7)
C
 1320 VER = .TRUE.
      INVC = 0
      NUMVR   =  NUMVR  +1
      TRIG = .FALSE.
      DO 1101  I = 1,M2
        E(I) = 0.0
 1101 CONTINUE
      MM=1
C
C     SET E(1) AND EVERY I=N*(M+1) VALUE OF E(I) EQUAL TO 1.0 UP TO
C     I=M**2 (N=SET OF INTEGERS).
C     SET X(I)=B(I) FOR I=1,M (IE LET X(I) BE THE VARIABLE IN THE BASIS)
C
      DO 1113  I = 1,M
        E(MM) = 1.0
        PE(I)  =  0.0
        X(I) =  B(I)
        IF (JH(I) .NE.0) JH(I) = -1
        MM = MM + M + 1
 1113 CONTINUE
C                 FORM INVERSE
      DO 1102  JT = 1,N
        IF (KB(JT).EQ.0)  GO TO 1102
        GO TO 600
C
C     TRANSFER CONTROL TO THE MACRO -JMY- BEGINING  AT STATEMENT NUMBER
C     600 FOR ALL COLUMNS THAT HAVE KB(J)=1.0
C     LET TY=PIVOT ELEMENT
C     SET IR=ROW NUMBER IN WHICH THE PIVOT ELEMENT OCCURS
C     CALCULATE A(I,JT)/B(I) SELECT THE LARGEST VALUE IN COLUMN JT
C     SET TY=(THE VALUE OF THE ABOVE RATIO)
C     CHECK THAT TY.GT.0.    RESET THE FLAG KB(JT)=0
C
C 600   CALL JMY
C                          CHOOSE PIVOT
C
 1114   TY = 0.0
        KQ = .FALSE.
        DO 1104 I = 1,M
          IF (JH(I).NE.-1.OR.ABS(Y(I)).LE.TPIV)  GO TO 1104
          IF (KQ) GO TO 1116
          IF (X(I).EQ.0.)  GO TO 1115
          IF (ABS(Y(I)/X(I)).LE.TY)  GO TO 1104
          TY = ABS(Y(I)/X(I))
          GO TO 1118
 1115     KQ = .TRUE.
          GO TO 1117
 1116     IF (X(I).NE.0..OR.ABS(Y(I)).LE.TY)  GO TO 1104
 1117     TY = ABS(Y(I))
 1118     IR = I
 1104   CONTINUE
        KB(JT) = 0
```

```
C                             TEST PIVOT
        IF (TY.LE.0.)          GO TO 1102
C                             PIVOT
        GO TO 900
C
C     TRANSFER CONTROL TO THE MACRO -PIV- BEGINING AT STATEMENT NUMBER
C     900
C
C 900   CALL PIV
 1102 CONTINUE
C
C                       RESET ARTIFICIALS
C
      DO 1109  I = 1,M
        IF (JH(I).EQ.-1)  JH(I) = 0
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1109 CONTINUE
C
C     THE LOGICAL VARIABLE VER IS USED TO DETERMINE IF THE SOLUTION IS
C     IN PHASE 1 OR IN PHASE 2
C
 1200 VER = .FALSE.
C
C                    ***           PERFORM ONE ITERATION           ***
C* 'XCK'     DETERMINE FEASIBILITY                           (STEP 1)
C
      NEG = .FALSE.
      IF (FEAS)  GO TO 500
      FEAS= .TRUE.
      DO 1201  I = 1,M
        IF (X(I).LT.0.0)  GO TO 1250
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1201 CONTINUE
C* 'GET'     GET APPLICABLE PRICES                           (STEP 2)
      IF (.NOT.FEAS)  GO TO 501
  500 DO 503 I = 1,M
        P(I) = PE(I)
        IF (X(I).LT.0.)  X(I) = 0.
  503 CONTINUE
      ABSC = .FALSE.
      GO TO 599
 1250 FEAS = .FALSE.
      NEG  = .TRUE.
  501 DO 504  J = 1, M
        P(J) = 0.
  504 CONTINUE
      ABSC = .TRUE.
      DO 505  I = 1,M
        MM = I
        IF (X(I).GE.0.0)  GO TO 507
        ABSC = .FALSE.
        DO 508 J = 1,M
          P(J) = P(J) + E(MM)
          MM = MM + M
  508   CONTINUE
```

```
         GO TO 505
  507    IF (JH(I).NE.0) GO TO 505
         IF (X(I).NE.0.) ABSC = .FALSE.
         DO 510  J = 1,M
           P(J) = P(J) - E(MM)
           MM = MM + M
  510    CONTINUE
  505 CONTINUE
C
C* 'MIN'    FIND MINIMUM REDUCED COST                      (STEP 3)
C
  599 JT = 0
      BB  =  0.0
      DO 701  J =1,N
         IF (KB(J).NE.0)   GO TO 701
         DT = 0.0
         DO 303 I = 1,M
           DT = DT + P(I) * A(I,J)
  303    CONTINUE
         IF (FEAS)  DT = DT + C(J)
         IF  (ABSC)  DT = - ABS(DT)
         IF (DT.GE.BB)  GO TO 701
         BB = DT
         JT = J
  701 CONTINUE
C
C  TEST FOR NO PIVOT COLUMN
C
      IF (JT.LE.0)  GO TO 203
C
C  TEST FOR ITERATION LIMIT EXCEEDED
C
      IF (ITER.GE.NCUT)  GO TO 160
      ITER = ITER +1
C
C     START OF THE MACRO -JMY-
C
C* 'JMY'    MULTIPLY INVERSE TIMES A(.,JT)                 (STEP 4)
  600 DO 610  I= 1,M
        Y(I) = 0.0
  610 CONTINUE
      LL = 0
      COST = C(JT)
C
C     LET Y(I) (WHERE I=THE ROW NUMBER) BE THE COEFFICIENT OF THE
C     VARIABLE IN THE BASIS IN COLUMN JT
C     SET COST=THE COEFFICIENT OF THE JT-TH TERM IN THE OBJECTIVE
C     FUNCTION
      DO 605   I= 1,M
         AIJT = A(I,JT)
         IF (AIJT.EQ.0.) GO TO 602
         COST = COST + AIJT * PE(I)
         DO  606  J = 1,M
           LL = LL + 1
```

```
          Y(J) = Y(J) + AIJT * E(LL)
  606   CONTINUE
        GO TO 605
  602   LL = LL + M
  605 CONTINUE
C
C          COMPUTE PIVOT TOLERANCE
C
      YMAX = 0.0
C
C     SET YMAX=THE LARGEST VALUE OF Y(I)
C     SET PIV=YMAX*0.5**16
C
      DO 620   I   =   1,M
        YMAX = AMAX1( ABS(Y(I)),YMAX )
  620 CONTINUE
      TPIV   =   YMAX * TEXP
C                RETURN TO INVERSION ROUTINE, IF INVERTING
C
C     END OF MACRO  -JMY-
C
      IF (VER)  GO TO 1114
C       COST TOLERANCE CONTROL
      RCOST = YMAX/BB
      IF (TRIG.AND.BB.GE.-TPIV)  GO TO 203
      TRIG = .FALSE.
      IF (BB.GE.-TPIV)  TRIG = .TRUE.
C* 'ROW'     SELECT PIVOT ROW                          (STEP 5)
C AMONG EQS. WITH X=0, FIND MAXIMUM Y  AMONG ARTIFICIALS, OR, IF NONE,
C  GET MAX POSITIVE Y(I) AMONG REALS.
      IR = 0
      AA = 0.0
      KQ = .FALSE.
      DO  1050  I =1,M
        IF (X(I).NE.0.0.OR.Y(I).LE.TPIV)  GO TO 1050
        IF (JH(I).EQ.0)  GO TO 1044
        IF (KQ) GO TO 1050
 1045   IF (Y(I).LE.AA)  GO TO 1050
        GO TO 1047
 1044   IF (KQ) GO TO 1045
        KQ = .TRUE.
 1047   AA = Y(I)
        IR = I
 1050 CONTINUE
      IF (IR.NE.0)  GO TO 1099
      AA = 1.0E+20
C                FIND MIN. PIVOT AMONG POSITIVE EQUATIONS
      DO 1010  I  = 1,M
        IF (Y(I).LE.TPIV.OR.X(I).LE.0.0.OR.Y(I)*AA.LE.X(I) ) GO TO 1010
        AA = X(I)/Y(I)
        IR = I
 1010 CONTINUE
      IF (.NOT.NEG)  GO TO 1099
C  FIND PIVOT AMONG NEGATIVE EQUATIONS, IN WHICH X/Y IS LESS THAN THE
```

```
C MINIMUM X/Y IN THE POSITIVE EQUATIONS, THAT HAS THE LARGEST ABSF(Y)
      BB = - TPIV
      DO 1030  I = 1,M
        IF (X(I).GE.0..OR.Y(I).GE.BB.OR.Y(I)*AA.GT.X(I) )  GO TO 1030
        BB = Y(I)
        IR = I
 1030 CONTINUE
C  TEST FOR NO PIVOT ROW
 1099 IF  (IR.LE.0)  GO TO 207
C* 'PIV'    PIVOT ON (IR,JT)                           (STEP 6)
      IA = JH(IR)
      IF (IA.GT.0)  KB(IA) = 0
C
C     START OF MACRO -PIV-
C
  900 NUMPV  =  NUMPV  +  1
      JH(IR) = JT
      KB(JT) = IR
C
C     SET YI=-(COEFFICIENT OF THE VARIABLE IN THE BASIS IN ROW IR)
C           =A(IR,JT)
C     SET Y(IR)=-1.0
C
      YI = -Y(IR)
      Y(IR) = -1.0
      LL = 0
C                                  TRANSFORM INVERSE
      DO  904  J = 1,M
        L = LL + IR
        IF (E(L).NE.0.0)  GO TO 905
        LL = LL + M
        GO TO 904
C
C     LET XY=INVERSE OF -A(IR,JT) AND E(LL)=INVERSE OF A(IR,JT)
C
C
C     SET X(IR)=B(IR)/A(IR,JT)           END OF MACRO -PIV-
C
  905   XY = E(L) / YI
        PE(J) = PE(J) + COST * XY
        E(L) = 0.0
        DO 906  I = 1,M
          LL  =  LL + 1
          E(LL) = E(LL)  + XY * Y(I)
  906   CONTINUE
  904 CONTINUE
C                                  TRANSFORM X
      XY  =  X(IR) / YI
      DO 908  I  = 1, M
        XOLD = X(I)
        X(I) = XOLD + XY * Y(I)
        IF (.NOT.VER.AND.X(I).LT.0..AND.XOLD.GE.0.)  X(I) = 0.
  908 CONTINUE
      Y(IR) = -YI
      X(IR) = -XY
```

```
      IF (VER) GO TO 1102
      IF (NUMPV.LE.M) GO TO 1200
C
C  TEST FOR INVERSION ON THIS ITERATION
C
      INVC   = INVC   +1
      IF  (INVC.EQ.NVER)   GO TO 1320
      GO  TO  1200
C
C*  END OF ALGORITHM, SET EXIT VALUES              ***
C
  207 IF (.NOT.FEAS.OR.RCOST.LE.-1000.)  GO TO 203
C
C             INFINITE SOLUTION
C
      K  =  2
      GO TO 250
C         PROBLEM IS CYCLING
  160 K  =  4
      GO TO 250
C
C         FEASIBLE OR INFEASIBLE SOLUTION
C
  203 K  =  0
  250 IF (.NOT.FEAS)  K = K + 1
      DO 1399  J = 1,N
        XX  =  0.0
        KBJ  =  KB(J)
        IF (KBJ.NE.0)  XX = X(KBJ)
        KB(J) = LL
 1399 CONTINUE
      KO(1) = K
      KO(2) = ITER
      RETURN
      END


      SUBROUTINE SEEK1(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,
     1 DSTAR,NTERMS,NTOTER)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),DSTAR(NTOTER
     1,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      IF(INDEX.EQ.1)WRITE(6,19)
      IF(INDEX.EQ.1.AND.IPRINT.GT.0)WRITE(6,7)
C
C     SUBR.SEARCH IS USED BY SEEK1 AND SEEK3,BOTH OF WHICH ARE CALLED BY
C     OTHER METHODS. NNDEX IS USED IN SEARCH (AND OPTIMF) TO IDENTIFY
C     SEEK1 OR SEEK3.(INDEX RETAINS THE VALUE FOR THE METHOD WHICH HAS
C     CALLED SEEK1 OR SEEK3).
C
      NNDEX=1
      KOUNT=0
```

```
      SUBROUTINE SEEK1(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,
     1 DSTAR,NTERMS,NTOTER)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),DSTAR(NTOTER
     1,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      IF(INDEX.EQ.1)WRITE(6,19)
      IF(INDEX.EQ.1.AND.IPRINT.GT.0)WRITE(6,7)
C
C     SUBR.SEARCH IS USED BY SEEK1 AND SEEK3,BOTH OF WHICH ARE CALLED BY
C     OTHER METHODS. NNDEX IS USED IN SEARCH (AND OPTIMF) TO IDENTIFY
C     SEEK1 OR SEEK3.(INDEX RETAINS THE VALUE FOR THE METHOD WHICH HAS
C     CALLED SEEK1 OR SEEK3).
C
      NNDEX=1
      KOUNT=0
    2 CALL SEARCH(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,
     1  DSTAR,NTERMS,NTOTER)
C     IF SEEK1 HAS BEEN CALLED BY ANOTHER METHOD RETURN AFTER CALL
C     TO SEARCH
C     RESET NNDEX=INDEX FOR FUTURE CALLS TO OPTIMF OR SEARCH BY THE
C     CALLING METHOD.
      NNDEX=INDEX
      IF(INDEX.NE.1)RETURN
      CALL SHOT(U,X,N,KK,PHI,PSI,NCONS,NEQUS,RMAX,RMIN)
C     CHECK TO SEE WHETHER SUBR.SHOT HAS FOUND AN IMPROVED POINT
      IF(KK.EQ.1) GO TO 4
      IF(KO.EQ.0)GOTO16
C     KO  CANNOT BE RESET IN SUBR.SHOT, THEREFORE IF KO=1 AT THIS STAGE
C     THEN SUBR.SEARCH FAILED AND SHOT FOUND NO IMPROVEMENT
      WRITE(6,5)
      GOTO16
    4 IF(IPRINT.GT.0)WRITE(6,25)U,(X(I),I=1,N)
      KOUNT=KOUNT+1
      IF(KOUNT.LE.NSHOT)GOTO13
      WRITE(6,17)NSHOT
      KO=1
      GOTO16
C     REDEFINE STARTING POINT FOR SEARCH
   13 DO 14 I=1,N
   14 XSTRT(I)=X(I)
      GOTO 2
C     PRINT OUT OPTIMUM(KO=0) OR LAST ITERATIONS RESULTS(KO=1)
   16 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
    5 FORMAT(1H-,71HDIRECT SEARCH HAS HUNG UP AND SHOTGUN SEARCH CANNOT
     1FIND A BETTER POINT/41HTRY A DIFFERENT STARTING POINT AT LEVEL=1/)
    7 FORMAT(1H-,15X,1HU,25X,23HINDEPENDENT VARIABLES X//)
   19 FORMAT(1H1,10X,38HDIRECT SEARCH OPTIMIZATION USING SEEK1//)
   17 FORMAT(1H-,48HSHOTGUN SEARCH FOUND AN IMPROVEMENT BUT  NSHOT =,I6,
     118H HAS BEEN EXCEEDED/1X,34HTRY RUNNING THIS PROBLEM ON ADRANS/)
   25 FORMAT(1H-,7H.SHOT. ,5E16.8/(24X,4E16.8))
      RETURN
      END
```

```
      SUBROUTINE SHOT(U,X,N,KK,PHI,PSI,NCONS,NEQUS,RMAX,RMIN)
      DIMENSION PHI(1),PSI(1),RMAX(1),RMIN(1),X(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON/A2/RR(100),XX(100)
      COMMON /A5/RF(100)
C     U=OPTIMUM DETERMINED BY DIRECT SEARCH.  IT IS CHANGED TO IMPROVED
C     VALUE IF SUCH A VALUE IS OBTAINED
C     XX= TRIAL VALUES OF X(I) FROM SHOTGUN SEARCH
C     RF= FRACTION OF RANGE USED IN SHOTGUN SEARCH
C     KK= INDICATOR TO SHOW IF U RETURNED IS AN IMPROVEMENT
C     INITIALIZE RANDOM NUMBER GENERATOR
      CALL FRANDN(RR,N,1)
      UMIN=U
      KK=0
C     THIS SHOTGUN SEARCH IS INTENDED TO GET THE SOLUTION OFF A FENCE
C     RATHER THAN TO INCH IT TOWARDS THE OPTIMUM. THEREFORE LARGE STEPS,
C     EQUAL 10. TIMES THE INITIAL STEP SIZE IN SEARCH ARE TRIED.
      DO 1 I=1,N
    1 RF(I)=10.*F*ABS(RMAX(I)-RMIN(I))
      DO 4 J=1,NTEST
      CALL FRANDN(RR,N,0)
      DO 2 I=1,N
    2 XX(I)=(X(I)-RF(I))+RR(I)*2.0*RF(I)
      CALL OPTIMF(XX,UTEST,PHI,PSI,NCONS,NEQUS)
      IF(NVIOL.NE.0)GOTO4
      IF(UTEST.GE.UMIN)GOTO4
      UMIN=UTEST
      U=UTEST
      DO 3 I=1,N
    3 X(I)=XX(I)
      KK=1
    4 CONTINUE
      RETURN
      END


      SUBROUTINE SEARCH (X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,
     1  UART,DSTAR,NTERMS,NTOTER)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),
     1 DSTAR(NTOTER,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON/A1/XO(100),XB(100),DXXX(100),TXXX(100)
C
C     DIRECT  SEARCH  PORTION  OF  SEEK1  AND  SEEK3
C
C     SUBR.SEARCH IS USED BY SEEK1 AND SEEK3,BOTH OF WHICH ARE CALLED BY
C     OTHER METHODS. NNDEX IS USED IN SEARCH (AND OPTIMF) TO IDENTIFY
C     SEEK1 OR SEEK3.(INDEX RETAINS THE VALUE FOR THE METHOD WHICH HAS
C     CALLED SEEK1 OR SEEK3).
C     NNDEX=1  MEANS SEARCH HAS BEEN CALLED BY SEEK1
C     NNDEX=3  MEANS SEARCH HAS BEEN CALLED BY SEEK3
```

```
C       IN CASE SEARCH IS CALLED DIRECTLY BY ANOTHER METHOD,DEFINE NNDEX
        IF(NNDEX.NE.1.AND.NNDEX.NE.3)NNDEX=INDEX
        NVIOL1=1
        KKK=0
        M1 = 0
C       DEFINE INDICES OF X(I) FOR GEOMETRIC PROGRAMMING
        IF(INDEX.NE.7)GOTO 20
        K1=2
        K2=NTOTER-N
        GOTO 30
   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
C       CHECKS FOR PURPOSE OF CALL TO SEEK1
        NCALL=1
   90 IF(INDEX.NE.7) GO TO 100
        CALL GEOPT(NTOTER,N,NCONS,NTERMS,DSTAR,UART,X)
        GOTO 110
  100 CONTINUE
        CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
  110 IF(NCALL.NE.1)GOTO   120
        UARTO = UART
  120 CONTINUE
C       ONCE THE SOLUTION HAS BECOME FEASIBLE(NVIOL=0) THE PENALTY
C       FUNCTIONS IN OPTIMF PREVENT IT GOING INFEASIBLE.THEREFORE NVIOL1=0
C       MEANS THE SOLUTION HAS BECOME PERMANENTLY FEASIBLE
        IF(NVIOL.EQ.0)NVIOL1=0
        IF(INDEX.EQ.1) GO TO 130
        IF(INDEX.EQ.3) GO TO 130
        IF(INDEX.EQ.7)  GOTO  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 90
```

```
  200 CONTINUE
      IF(UART.LT.UARTO)  GOTO  230
      X(I)=X(I) - 2.0*DXXX(I)
      NCALL=3
      GO TO    90
  210 CONTINUE
      IF(UART.LT.UARTO)  GOTO  230
      NFAIL = NFAIL + 1
      X(I)=X(I)+DXXX(I)
      GOTO  240
  230 UARTO = UART
  240 CONTINUE
      IF(INDEX.NE.7)GOTO 250
      NUMB=K2-1
      IF(NFAIL.EQ.NUMB)GOTO 260
      GOTO315
  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) = X(I)
      M1 = M1 + 1
      IF(INDEX.EQ.1)GOTO330
      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 90
  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 IF(INDEX.EQ.7)GOTO387
```

```
      CALL UREAL(X,U)
      CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      IF(NVIOL.EQ.0)GOTO387
      IF(M1.GT.MAXM)WRITE(6,4)MAXM
      KO=1
  387 RETURN
    2 FORMAT(1H0,I4,3X,5E16.8/(24X,4E16.8))
    4 FORMAT(1H0,60HNO FEASIBLE SOLUTION AFTER ALLOWABLE NUMBER OF MOVES
     1, MAXM =,I6/)
      END


      SUBROUTINE OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      DIMENSION X(1),PHI(1),PSI(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,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=-1.0E-10
      NVIOL=0
C     SUBR.OPTIMF IS USED BY SEEK1 AND SEEK3,BOTH OF WHICH ARE CALLED BY
C     OTHER METHODS. NNDEX IS USED IN OPTIMF (AND SEARCH) TO IDENTIFY
C     SEEK1 OR SEEK3.(INDEX RETAINS THE VALUE FOR THE METHOD WHICH HAS
C     CALLED SEEK1 OR SEEK3).
C     NNDEX=1  MEANS SEARCH HAS BEEN CALLED BY SEEK1
C     NNDEX=3  MEANS SEARCH HAS BEEN CALLED BY SEEK3
      SUM1=0.0
      SUM2=0.0
      CALL UREAL(X,U)
      IF(NNDEX.EQ.3)GOTO110
C
C     SEEK1  PENALTY  FUNCTIONS -
C
C     A  ROUTINE TO CALCULATE A VALUE FOR AN ARTIFICIAL OBJECTIVE
C     FUNCTION OF THE FORM
C        UART=UREAL+SUM(ABS(PHI(I)))*10.E20+SUM(ABS(PSI(I)))*10.E20
C     WHERE
C     PSI(I) AND PHI(I) IN THE ABOVE EXPRESSION ARE THE VALUES OF THE
C     CORESPONDING EQUALITY AND INEQUALITY CONSTRAINTS THAT HAVE BEEN
C     VIOLATED
      IF(NCONS.EQ.0)GOTO2
      CALL CONST(X,NCONS,PHI)
      DO 1 I=1,NCONS
      IF(PHI(I).GE.ZERO)GOTO1
      SUM1=SUM1 + ABS(PHI(I))*10.0E+20
      NVIOL=NVIOL + 1
    1 CONTINUE
    2 IF(NEQUS.EQ.0)GOTO115
      CALL EQUAL(X,PSI,NEQUS)
      DO 3 I=1,NEQUS
    3 SUM2=SUM2 + ABS(PSI(I))*10.0E+20
      GOTO115
```

```
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
  115 UART=U+SUM1+SUM2
      RETURN
      END


      SUBROUTINE SEEK2(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,GS)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),GS(1)
      COMMONINDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PDO,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A1/DX(100),XO(100),DXS(100),XN(100)
      NNDEX=INDEX
      WRITE(6,101)
      KUT=0
      KOUNT=0
      DO 2 I=1,N
      X(I)=XSTRT(I)
      XO(I)=X(I)
      DX(I)=F*ABS(RMAX(I)-RMIN(I))
      DXS(I)=DX(I)
    2 CONTINUE
   61 CALL OPTIMF(X,UARTO,PHI,PSI,NCONS,NEQUS)
   62 U=UARTO
C     PERFORM THE UNIVARIABLE SEARCH
      DO 6 I=1,N
C     MAKE A MOVE IN THE POSITIVE DIRECTION
    3 X(I)=X(I)+DX(I)
      CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      IF(UART.LT.U)GOTO4
C     MAKE A MOVE IN THE NEGATIVE DIRECTION
      X(I)=X(I)-2.0*DX(I)
```

```
      CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      IF(UART.LT.U)GOTO5
C     RETURN TO ORIGINAL VALUE
      X(I)=X(I)+DX(I)
      GOTO6
    4 U=UART
C     INCREASE STEP LENGTH AFTER A SUCCESSFUL MOVE
      DX(I)=DX(I)*GS(I)
      X(I)=X(I)+DX(I)
      CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      IF(UART.LT.U)GOTO4
C     RETURN TO ORIGINAL POSITION AFTER A FAILURE
      X(I)=X(I)-DX(I)
      DX(I)=DXS(I)
C     DECIDE WHETHER OR NOT TO PROCEED WITH UNIVARIABLE SEARCH
C     (IFENCE=0 AT LEVEL=0)
      IF(IFENCE.EQ.1)GOTO6
      GOTO3
C     INCREASE STEP LENGTH AFTER A SUCCESSFUL NEGATIVE MOVE
    5 DX(I)=-DX(I)
      GOTO4
    6 CONTINUE
C     CHECK PERCENTAGE IMPROVEMENT IN  U
      CALL OPTIMF(X,UART,PHI,PSI,NCONS,NEQUS)
      IF(ABS(UART-UARTO).GT.EPS*ABS(UARTO))GOTO8
      IF(KUT.LT.ICT)GOTO7
      IF(NVIOL.EQ.0)GOTO99
      KO=1
      WRITE(6,105)
      GOTO99
C     REDUCE STEP SIZE BY A FACTOR OF 10.0
    7 DO 18 I=1,N
      DX(I)=DX(I)/10.0
   18 DXS(I)=DX(I)
      UARTO=UART
      KUT=KUT+1
      GOTO62
C     START PATTERN MOVES
    8 U=UART
      PD=PDO
      DO 42 I=1,N
   42 XN(I)=X(I)
   15 DO 9 I=1,N
    9 XN(I)=XN(I)+(X(I)-XO(I))*PD
      CALL OPTIMF(XN,UART,PHI,PSI,NCONS,NEQUS)
      IF(UART.LT.U)GOTO14
      IF(PD.LT.0.0)GOTO13
C     TRY A NEGATIVE PATTERN MOVE
      DO 40 I=1,N
   40 XN(I)=XN(I)-(X(I)-XO(I))*PD
      PD=-PDO
      GOTO15
C     RETURN TO ORIGINAL POINT
   13 DO 16 I=1,N
```

```
      UARTO=U
      KOUNT=KOUNT+1
      IF(IPRINT.EQ.0)GOTO17
      IF(KOUNT.EQ.IPRINT)WRITE(6,102)
      IF((KOUNT/IPRINT)*IPRINT.NE.KOUNT)GOTO17
      CALL UREAL(X,UU)
      WRITE(6,103)KOUNT,UU,(X(I),I=1,N)
   17 IF(KOUNT.EQ.MAXM)GOTO20
      GOTO62
C     ACCELERATE STEP LENGTH AFTER SUCCESSFUL PATTERN MOVES
   14 PD=PD*PL
      U=UART
      DO 11 I=1,N
   11 XN(I)=XN(I)+(X(I)-XO(I))*PD
      CALL OPTIMF(XN,UART,PHI,PSI,NCONS,NEQUS)
      IF(UART.LT.U)GOTO14
C     RETURN TO LAST POSITION AFTER PATTERN MOVE FAILS
      DO 41 I=1,N
   41 XN(I)=XN(I)-(X(I)-XO(I))*PD
      PD=PDO
      GOTO15
C     NO CONVERGENCE AFTER MAXM COMPLETE CYCLES
   20 WRITE(6,104)MAXM
      KO=1
   99 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
  101 FORMAT(1H1,46HOPTIMIZATION USING DIRECT SEARCH METHOD  SEEK2/)
  102 FORMAT(1H-,15X,1HU,25X,26HINDEPENDENT VARIABLES X(I)//)
  103 FORMAT(1H0,I4,3X,5E16.8/(24X,4E16.8))
  104 FORMAT(1H-,29H  OPTIMUM CANNOT BE FOUND IN ,I3,7H CYCLES)
  105 FORMAT(1H-,43HSEEK2 CANNOT FIND A FEASIBLE STARTING POINT/)
      RETURN
      END



      SUBROUTINE SEEK3(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DS
     1TAR,NTERMS,NTOTER)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),DSTAR(NTOTER
     1,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      IF(INDEX.EQ.3)WRITE(6,9)
      ULAST=10.0E+40
      KOUNT=0
C     DEFINE NNDEX=3 SO THAT OPTIMF AND SEARCH WILL FUNCTION CORRECTLY
      NNDEX=3
C     DEFINE R AND REDUCE FOR THE CASE WHERE SEEK3 HAS BEEN CALLED BY
C     ANOTHER METHOD
      IF(INDEX.NE.3)R=1.0
      IF(INDEX.NE.3)REDUCE=0.04
    1 CALL      SEARCH(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DST
     1AR,NTERMS,NTOTER)
C     IF SEEK3 HAS BEEN CALLED BY ANOTHER METHOD RETURN
C     RESET NNDEX=INDEX FOR FUTURE CALLS TO OPTIMF OR SEARCH BY THE
C     CALLING METHOD.
```

```
      NNDEX=INDEX
      IF(INDEX.NE.3)RETURN
      IF(KO.NE.1)GOTO5
      WRITE(6,14)
      GOTO6
    5 KOUNT=KOUNT+1
      IF(IPRINT.EQ.0)GOTO2
      IF(KOUNT.EQ.IPRINT)WRITE(6,10)
      IF((KOUNT/IPRINT)*IPRINT.NE.KOUNT)GOTO2
      WRITE(6,4)R
      WRITE(6,11)U,(X(I),I=1,N)
    2 IF(ABS(U-ULAST).GT.1.E-07*ABS(ULAST))GOTO7
C     OPTIMUM HAS BEEN REACHED
    6 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
      RETURN
    7 IF(R.GT.1.0E-20)GOTO8
      WRITE(6,12)R
      KO=1
      GOTO6
    8 ULAST=U
      R=R*REDUCE
      DO 3 I=1,N
    3 XSTRT(I)=X(I)
      GOTO1
    4 FORMAT(1H0,3HR =,E16.8)
    9 FORMAT(1H1,45HOPTIMIZATION USING DIRECT SEARCH METHOD SEEK3,//)
   10 FORMAT(1H0,38X,27HINDEPENDENT VARIABLES  X(I)//)
   11 FORMAT(1X,3HU =,E16.8,1X,4E16.8/(21X,4E16.8))
   12 FORMAT(1H0,23HNO CONVERGENCE WITH R =,E16.8)
   14 FORMAT(66H1SEEK3 UNABLE TO FIND A FEASIBLE STARTING POINT(ALL PHI(
     1I).GE.0.0)/)
      END



      SUBROUTINE ALTS(X,U,N,XSTRT,RMAX,RMIN,WATE,STEP,NEQUS,NCONS,PSI,PH
     1I,M,NN,A,B,C,WORKA,DSTAR,NTERMS,NTOTER,DELX,XX)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),WATE(1),STEP(1),PSI(1),PHI
     1(1),DELX(1),A(M,1),B(1),C(1),WORKA(M,1),DSTAR(NTOTER,1),NTERMS(1),
     2XX(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A5/ XINC(100)
      WRITE(6,1)
      UU=1.0E+40
      UBEST=1.0E+40
      NCY=0
C     CHECK INPUT VALUE OF STEP(I). THE LINEARIZATION PERFORMED IN SUBR.
C     LINEAR IS ONLY VALID FOR A SMALL STEP SIZE
      DO 9 I=1,N
      X(I)=XSTRT(I)
      RANGE=ABS(RMAX(I)-RMIN(I))
    9 IF(ABS(STEP(I)).GT.0.10*RANGE)STEP(I)=0.10*RANGE
```

```
C     CALL SUBR.FEASBL TO CHECK WHETHER XSTRT(I) IS FEASIBLE AND TO
C     DRIVE IT FEASIBLE IF NECESSARY.
      IF(NCONS.EQ.0.AND.NEQUS.EQ.0)GOTO20
      CALL FEASBL(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UDUMMY,DSTAR
     1,NTERMS,NTOTER)
      IF(KO.EQ.0)GOTO10
C     SUBR. LINEAR CAN HANDLE INFEASIBLE INEQUALITY CONSTRAINTS, BUT NOT
C     UNSATISFIED EQUALITIES
      IF(NEQUS.GT.0)RETURN
C     PROCEED WITH LINEARIZATION, RESET KO=0
      KO=0
      GOTO30
   10 IF(IPRINT.GT.0)WRITE(6,3)U,(X(I),I=1,N)
   20 CALL ASERCH(X,N,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,NCY,WATE)
      CALL UREAL(X,UARTO)
C     CHECK TO SEE IF THE RESULTS OF THIS SEARCH HAVE IMPROVED  U  OVER
C     THE PREVIOUS SEARCH(THIS METHOD TENDS TO OSCILLATE)
      IF((UARTO-UU).LT.0.0)GOTO21
C     CHECK FOR OSC+LLATION, I.E. NO SIGNIFICANT CHANGE FROM LAST SEARCH
      IF(ABS(UARTO-UU).LT.1.0E-08)GOTO23
      GOTO24
   21 IF((UARTO-UBEST).GE.0.0)GOTO24
C     DEFINE THE NEW 'BEST' POINT AND STORE IT IN UBEST AND XSTRT(I)
      UBEST=UARTO
      UU=UARTO
      DO 22 I=1,N
   22 XSTRT(I)=X(I)
      GOTO35
C     IF THE OPTIMIOATION FUNCTION IS OSCILLATING   , RETURN TO 'BEST'
C     POINT SO FAR
   23 WRITE(6,7)
      U=UBEST
      DO 26 I=1,N
   26 X(I)=XSTRT(I)
      GOTO110
C     STORE VALUE OF U FOR THIS ITERATION
   24 UU=UARTO
      GOTO35
   30 CALL UREAL(X,UARTO)
   35 IF(NEQUS.EQ.0)GOTO50
      CALL EQUAL(X,PSI,NEQUS)
      DO 40 I=1,NEQUS
   40 UARTO=UARTO+ABS(PSI(I))*WATE(I)
   50 CALL LINEAR(X,UO,PHI,PSI,A,B,C,DELX,STEP,M,NN,N,NCONS,NEQUS)
      CALL SIMPLE(XX,DELU,M,NN,A,B,C,WORKA)
      IF(KO.EQ.1)RETURN
      DO 60 I=1,N
      XINC(I)=XX(2*I-1)-XX(2*I)
   60 X(I)=X(I)+XINC(I)
      CALL UREAL(X,U)
      NCY=NCY+1
      IF(IPRINT.EQ.0)GOTO70
      WRITE(6,5)U,(X(I),I=1,N)
   70 IF(NCY.GT.NSMAX)GOTO100
```

```
      UART=U
      NVIOL=0
      IF(NEQUS.EQ.0)GOTO81
      CALL EQUAL(X,PSI,NEQUS)
      DO 80 I=1,NEQUS
   80 UART=UART+ABS(PSI(I))*WATE(I)
C     CHECK IF PREVIOUS MOVE WAS INFEASIBLE
   81 IF(NCONS.EQ.0)GOTO90
      CALL CONST(X,NCONS,PHI)
      DO 82 I=1,NCONS
   82 IF(PHI(I).LT.0.0)GOTO83
      GOTO90
C     IF LAST POINT FOUND BY LINEARIZATION WAS INFEASIBLE, BYPASS ASERCH
C     AND GO DIRECTLY TO LINEARIZATION
   83 IF(IPRINT.GT.0)WRITE(6,4)
      NVIOL=1
   90 IF(ABS(UARTO-UART).LT.TES*ABS(UARTO))GOTO110
      IF(NVIOL.EQ.0)GOTO20
      UARTO=UART
      GOTO50
  100 WRITE(6,6)NSMAX
C     PRINT OUT THE 'BEST' VALUE SO FAR
      U=UBEST
      DO 105 I=1,N
  105 X(I)=XSTRT(I)
      KO=1
  110 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
    1 FORMAT(1H1,35HOPTIMIZATION USING ALTERNATE SEARCH//)
    2 FORMAT(1H-,47HMETHOD UNABLE TO FIND A FEASIBLE STARTING POINT/)
    3 FORMAT(1H-,47HFEASIBLE STARTING POINT FOUND BY METHOD IS  U =,E16.
     18,11H  AT X(I) =//(6X,5E16.8))
    4 FORMAT(30X,31H(THE ABOVE POINT IS INFEASIBLE))
    5 FORMAT(7HOLINEAR,E15.8,4E16.8/(22X,4E16.8))
    6 FORMAT(1H-,80HMAXIMUM NUMBER OF ITERATIONS THROUGH ALTERNATE SEARC
     1H HAS BEEN EXCEEDED (NSMAX =,I6,1H)/1X,43HTHE BEST POINT FOUND SO
     2FAR IS LISTED BELOW/)
    7 FORMAT(1H-,68HSOLUTION IS OSCILLATING, ASSUME PREVIOUS 'BEST' POIN
     1T IS THE OPTIMUM/)
      RETURN
      END


      SUBROUTINE ASERCH(X,N,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,NCY,WATE)
      DIMENSION X(1),RMAX(1),RMIN(1),PHI(1),PSI(1),WATE(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A3/TEST(100),DLLX(100),XO(100)
      COMMON /A5/XINC(100)
      KOUNT=0
      J=0
C     NOTE... ASERCH  ASSUMES THAT ALL PHI(I).GE.0. ALREADY
C     INITIALIZE THE STEP LENGTHS AND CONVERGENCE CRITERIA
      DO 10 I=1,N
```

```
      DLLX(I)=F*ABS(RMAX(I)-RMIN(I))
   10 TEST(I)=G*DLLX(I)
      CALL UREAL(X,UO)
      IF(NEQUS.EQ.0)GOTO35
      CALL EQUAL(X,PSI,NEQUS)
      DO 30 I=1,NEQUS
   30 UO=UO+ABS(PSI(I))*WATE(I)
C     IF A LINEARIZATION HAS JUST BEEN COMPLETED, TRY A PATTERN MOVE
   35 IF(NCY.GT.0)GOTO150
C     MAKE EXPLORATORY SEARCH
   40 DO 50 I=1,N
   50 XO(I)=X(I)
      NFAIL=0
      DO 120 I=1,N
      LOOP=1
      X(I)=X(I)+DLLX(I)
   55 CALL UREAL(X,U)
      IF(NCONS.EQ.0)GOTO70
      CALL CONST(X,NCONS,PHI)
      DO 60 L=1,NCONS
   60 IF(PHI(L).LT.0.0)GOTO100
   70 IF(NEQUS.EQ.0)GOTO90
      CALL EQUAL(X,PSI,NEQUS)
      DO 80 L=1,NEQUS
   80 U=U+ABS(PSI(L))*WATE(L)
   90 IF(U.GE.UO)GOTO100
      UO=U
      GOTO120
  100 LOOP=LOOP+1
      IF(LOOP.GT.2)GOTO110
      X(I)=X(I)-2.0*DLLX(I)
      GOTO55
  110 X(I)=X(I)+DLLX(I)
      NFAIL=NFAIL+1
  120 CONTINUE
C     DEFINE STEP LENGTH FOR PATTERN MOVE AFTER EXPLORATORY MOVES
      DO 125 I=1,N
  125 XINC(I)=X(I)-XO(I)
      IF(NFAIL.LT.N)GOTO150
      NIL=0
      DO 140 I=1,N
      IF(ABS(DLLX(I)).LT.ABS(TEST(I)))GOTO130
      DLLX(I)=DLLX(I)/2.0
      GOTO140
  130 NIL=NIL+1
  140 CONTINUE
C     IF ALL STEP LENGTHS DLLX(I).LT.TEST(I) CONVERGENCE IS ASSUMED
      IF(NIL.EQ.N)RETURN
      GOTO40
C     MAKE PATTERN MOVE
C     XINC(I) FROM LAST LINEARIZATION IS CARRIED THROUGH COMMON /A5/
  150 IF(J.EQ.0)HURRY=1.0
      IF(J.NE.0)HURRY=PL**J
      DO 160 I=1,N
  160 X(I)=X(I)+XINC(I)*HURRY
```

```
      IF(NCONS.EQ.0)GOTO180
      CALL CONST(X,NCONS,PHI)
      DO 170 I=1,NCONS
  170 IF(PHI(I).LT.0.0)GOTO210
  180 CALL UREAL(X,U)
      IF(NEQUS.EQ.0)GOTO200
      CALL EQUAL(X,PSI,NEQUS)
      DO 190 I=1,NEQUS
  190 U=U+ABS(PSI(I))*WATE(I)
  200 IF(U.GT.UO)GOTO210
      UO=U
C     ACCELERATE THE STEP AFTER A SUCCESSFUL PATTERN MOVE
      J=J+1
      GOTO150
C     RETURN TO LAST GOOD POINT
  210 DO 220 I=1,N
  220 X(I)=X(I)-XINC(I)*HURRY
C     IF J=0 AT THIS STAGE, THEN EVEN THE SMALLEST PATTERN MOVE HAS
C     FAILED AND ANOTHER EXPLORATORY MOVE MUST BE ATTEMPTED
      IF(J.GT.0)GOTO227
      KOUNT=KOUNT+1
      IF(IPRINT.EQ.0)GOTO225
      KOWNT=KOUNT+NCY
      IF(KOWNT.EQ.IPRINT)WRITE(6,4)
      IF((KOUNT/IPRINT)*IPRINT.NE.KOUNT)GOTO225
      CALL UREAL(X,UU)
      WRITE(6,5)KOUNT,UU,(X(I),I=1,N)
  225 IF(KOUNT.GT.MAXM)GOTO230
      GOTO40
  227 J=0
      GOTO150
  230 WRITE(6,1)MAXM
      KO=1
    1 FORMAT(1H-,56HTHE MAXIMUM NUMBER OF MOVES PERMITTED IN  ASERCH (MA
     1XM =,I6,19H) HAS BEEN EXCEEDED/)
    4 FORMAT(1H-,12X,1HU,25X,26HINDEPENDENT VARIABLES X(I)//)
    5 FORMAT(1H0,I3,2X,5E16.8/(21X,4E16.8))
      RETURN
      END


      SUBROUTINE APPROX(X,U,N,DELX,STEP,TEST,M,NN,A,B,C,WORKA,XSTRT,RMAX
     1,RMIN,PHI,PSI,NCONS,NEQUS,UART,DSTAR,NTERMS,NTOTER,XX)
      DIMENSION WORKA(1),X(1),DELX(1),STEP(1),TEST(1),A(M,1),B(1),C(1),
     1XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),DSTAR(NTOTER,1),NTERMS(1),
     2XX(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A7/XINC(100),WORK19(100)
      COMMON /A8/JELLY(100)
      WRITE(6,4)
      NSTEPL=0
      TINY=1.0E-08
```

```
      ULAST=1.0E+40
      DO 22 I=1,NN
   22 XX(I)=0.0
      DO 23 I=1,N
      JELLY(I)=0
      X(I)=XSTRT(I)
      WORK19(I)=XSTRT(I)
   23 XINC(I)=0.0
      IF(NEQUS.EQ.0.AND.NCONS.EQ.0)GOTO26
C     APPROX REQUIRES THAT ALL PSI(I) BE SATISFIED, BUT IT CAN HANDLE
C     INFEASIBLE PH+(I). IF THE USER HAS CHOSEN XSTRT(I) SO AS TO MAKE
C     ALL PSI(I)=0. , THEN FEASBL IS BYPASSED BECAUSE IT WOULD UPSET THE
C     GOOD VALUES OF PSI(I) IN ORDER TO DRIVE ALL PHI(I) FEASIBLE
      IF(NEQUS.EQ.0)GOTO27
      CALL EQUAL(XSTRT,PSI,NEQUS)
      DO 21 I=1,NEQUS
   21 IF(ABS(PSI(I)).GT.1.E-04)GOTO27
      GOTO26
C     DEFINE G AND MAXM FOR SUBROUTINE FEASBL
   27 G=F
      MAXM=100*N
C     CALL SUBR. FEASBL TO TEST WHETHER THE INPUT STARTING VALUES(XSTRT)
C     ARE FEASIBLE OR NOT.IF NOT,FEASBL DETERMINES A FEASIBLE STARTING
C     POINT AND RETURNS IT IN THE ARRAY X(I).
      CALL FEASBL(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UDUMMY,DSTAR
     1,NTERMS,NTOTER)
      IF(KO.NE.1.OR.NEQUS.EQ.0)GOTO24
      WRITE(6,76)
      GO TO 100
   24 IF(IPRINT.GT.0)CALL UREAL(X,U)
      IF(IPRINT.GT.0)WRITE(6,77)U,(X(I),I=1,N)
C
C     CHECK INPUT VALUES OF STEP(I). IF ANY STEP(I) .GT. 10 PERCENT OF
C     THE RANGE THEN REDUCE IT TO .10*(RMAX(I)-RMIN(I))
   26 DO 25 I=1,N
      RANGE=ABS(RMAX(I)-RMIN(I))
      IF(STEP(I).GT.0.10*RANGE)STEP(I)=0.10*RANGE
   25 CONTINUE
   35 CALL LINEAR(X,U0,PHI,PSI,A,B,C,DELX,STEP,M,NN,N,NCONS,NEQUS)
      CALL SIMPLE(XX,DELU,M,NN,A,B,C,WORKA)
      IF(KO.EQ.1)GOTO65
      DO 36 I=1,N
      XINC(I)=XX(2*+-1)-XX(2*I)
   36 X(I)=X(I)+XINC(I)
      CALL UREAL(X,U)
      NSTEPL=NSTEPL+1
      IF(IPRINT.EQ.0)GOTO37
      IF(NSTEPL.EQ.IPRINT)WRITE(6,70)
      IF((NSTEPL/IPRINT)*IPRINT.EQ.NSTEPL)WRITE(6,71)NSTEPL,U,(X(I),I=1,
     1N)
   37 IF(NSTEPL.GE.NSMAX) GO TO 62
C     REGULATION OF ALLOWABLE MAXIMUM STEP LENGTH STEP(I) -
```

```
C      HALF    STEP(I)...IF THE LAST INCREMENT WAS FINITE(.GT.TINY) BUT
C                          LESS THAN 5 PERCENT OF THE ALLOWABLE STEP(I)
C                       IF THE VARIABLE IS OSCILLATING
C      DOUBLE  STEP(I)...IF THE LAST INCREMENT WAS .GT.0.99*THE ALLOWABLE
C                          STEP(I) AND VARIABLE WAS NOT OSCILLATING
C      OSCILLATION    ...VALUES OF X(I) ARE COMPARED EVERY SECOND(EVEN)
C                       ITERATION. IF THEY ARE EQUAL AND THE LAST IN-
C                       CREMENT WAS FINITE THEN OSCILLATION MUST HAVE
C                       OCCURRED. SET THE FLAG JELLY(I)=1 TO PREVENT ANY
C                       SUBSEQUENT DOUBLING OF THE VARIABLE.(OSCILLATION
C                       IS ASSUMED TO TAKE PLACE ABOUT THE OPTIMUM)
      IF((NSTEPL/2)*2.NE.NSTEPL)GOTO59
      LESS=0
      DO 58 I=1,N
      IF(ABS(XINC(I)).LE.TINY)GOTO57
      IF(ABS(X(I)-XSTRT(I)).GT.TINY)GOTO55
C     SET FLAG JELLY(I)=1 FOR THE OSCILLATING VARIABLE
      JELLY(I)=1
      IF(STEP(I).GT.TEST(I))GOTO54
      LESS=LESS+1
      GOTO57
   54 STEP(I)=STEP(I)/2.0
      GOTO57
   55 IF(ABS(XINC(I)).GT.0.05*STEP(I))GOTO56
      IF(STEP(I).GT.TEST(I))STEP(I)=STEP(I)/2.0
      GOTO57
C     DO NOT INCREASE STEP(I) IF VARIABLE HAS OSCILLATED(JELLY(I)=1)
   56 IF(JELLY(I).EQ.1)GOTO57
C     DO NOT INCREASE STEP(I) SO THAT STEP(I).GT..1*(RMAX(I)-RMIN(I))
      IF(STEP(I).GT.0.05*ABS(RMAX(I)-RMIN(I)))GOTO57
      IF(ABS(XINC(I)).LT.0.99*STEP(I))GOTO57
      STEP(I)=STEP(I)*2.0
   57 XSTRT(I)=X(I)
   58 CONTINUE
      IF(LESS.LT.N)GOTO62
      IF((U-ULAST).GT.0.0)GOTO65
      GOTO100
C     CHECK FOR STEP SIZE ADJUSTMENT EVERY ITERATION(OSCILLATION CHECKED
C     ONLY ON EVEN NUMBERED ITERATIONS)
   59 DO 61 I=1,N
      IF(ABS(XINC(I)).GT.0.05*STEP(I))GOTO60
      IF(ABS(XINC(I)).LT.TINY)GOTO61
      IF(STEP(I).GT.TEST(I))STEP(I)=STEP(I)/2.0
      GOTO61
   60 IF(JELLY(I).EQ.1)GOTO61
      IF(ABS(XINC(I)).LT.0.99*STEP(I))GOTO61
      IF(STEP(I).GT.0.05*ABS(RMAX(I)-RMIN(I)))GOTO61
      STEP(I)=STEP(I)*2.0
   61 CONTINUE
   62 IF(NSTEPL.GE.NSMAX.AND.NCONS.EQ.0)GOTO64
      IF(NCONS.EQ.0)GOTO67
C     CHECK WHETHER OR NOT THE POINT IS FEASIBLE
      NVIOL=0
      CALL CONST(X,NCONS,PHI)
```

```
      DO 63 I=1,NCONS
   63 IF(PHI(I).LT.-TINY)NVIOL=NVIOL+1
C     AN INFEASIBLE POINT IS NOT A CANDIDATE TO BE THE OPTIMUM
      IF(NVIOL.EQ.0)GOTO67
      IF(IPRINT.EQ.0)GOTO72
      IF((NSTEPL/IPRINT)*IPRINT.EQ.NSTEPL)WRITE(6,78)
   72 IF(NSTEPL.GE.NSMAX)GOTO64
      GOTO35
   67 IF((U-ULAST).GE.0.0)GOTO69
C     STORE NEW 'BEST' POINT IN ULAST AND WORK19(I)
      ULAST=U
      DO 68 I=1,N
   68 WORK19(I)=X(I)
   69 DO 51 I=1,N
      IF(ABS(XINC(I)).GE.TEST(I)) GO TO 35
   51 CONTINUE
      IF((U-ULAST).GT.0.0)GOTO65
      GOTO100
   64 WRITE(6,5) NSMAX
      KO=1
C     PRINT OUT BEST POINT FOUND SO FAR
   65 DO 66 I=1,N
   66 X(I)=WORK19(I)
  100 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
    4 FORMAT(1H1,60HOPTIMIZATION USING METHOD OF SUCCESSIVE LINEAR APPRO
     1XIMATION//)
    5 FORMAT(1H-,45HLIMIT ON NO. OF ITERATIONS EXCEEDED, NSMAX = ,I5/1X,
     143HTHE BEST POINT FOUND SO FAR IS LISTED BELOW/)
   70 FORMAT(1H-,15X,1HU,25X,23HINDEPENDENT VARIABLES X//)
   71 FORMAT(1H0,I4,3X,5E16.8/(24X,4E16.8))
   76 FORMAT(1H-,49HSUBR. FEASBL UNABLE TO FIND FEASIBLE STARTING PT./)
   77 FORMAT(1H-,53HFEASIBLE STARTING VALUES FOUND BY   FEASBL   ARE  U
     1=,E16.8,10H AT X(I) =//(1X,E15.8,4E16.8))
   78 FORMAT(30X,31H(THE ABOVE POINT IS INFEASIBLE))
   81 FORMAT(1H-,25HFINAL VALUES OF STEP(I) =,/(5E16.8))
      RETURN
      END


      SUBROUTINE LINEAR(X,U0,PHI,PSI,A,B,C,DELX,STEP,M,NN,N,NCONS,NEQUS)
      DIMENSION X(1),DELX(1),STEP(1),PHI(1),PSI(1),A(M,1),B(1),C(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A2/SIGN(100),PART(100)
C     ZERO ARRAYS TO BE USED
      DO 20 I=1,M
      B(I)=0.0
      DO 20 J=1,NN
      A(I,J)=0.0
   20 CONTINUE
      DO 22 I=1,NN
   22 C(I)=0.0
```

```
      DO 23 I=1,NCONS
   23 PHI(I)=0.0
      DO 24 I=1,NEQUS
   24 PSI(I)=0.0
C     LINEARIZE THE OPTIMIZATION FUNCTION
      CALL UREAL(X,U0)
      DO 10 I=1,N
      X(I)=X(I)+DELX(I)
      CALL UREAL(X,U)
      X(I)=X(I)-DELX(I)
      CTEMP=(U-U0)/DELX(I)
      C(2*I-1)=CTEMP
      C(2*I)=-CTEMP
   10 CONTINUE
C     SET UP EQUATIONS LIMITING THE STEP SIZE OF EACH VARIABLE FOR EACH
C     ITERATION
      DO 30 J=1,N
      JJ=J+N
      J2=2*J
      A(J,J2-1)=1.0
      A(J,J2)=-1.0
      A(JJ,J2-1)=-1.0
      A(JJ,J2)=1.0
      B(J)=ABS(STEP(J))
      B(JJ)=ABS(STEP(J))
   30 CONTINUE
C     SET UP SLACK VARIABLES IN STEP LENGTH LIMIT EQUATIONS
      MA=2*N
      DO 55 J=1,MA
      IJ=J+MA+NCONS
   55 A(J,IJ)=1.0
C     LINEARIZE THE INEQUALITY CONSTRAINTS, MULTIPLYING THROUGH BY -1.0
C     IF THE RIGHT HAND SIDE IS NEGATIVE
      IF(NCONS.EQ.0)GOTO48
      DO 29 I=1,NCONS
   29 PART(I)=0.0
      CALL CONST(X,NCONS,PART)
      DO 31 I=1,NCONS
      SIGN(I)=1.0
      IF(-PART(I).LT.0.0)SIGN(I)=-1.0
   31 CONTINUE
      DO 35 I=1,N
      X(I)=X(I)+DELX(I)
      CALL CONST(X,NCONS,PHI)
      X(I)=X(I)-DELX(I)
      DO 35 II=1,NCONS
      ATEMP=SIGN(II)*(PHI(II)-PART(II))/DELX(I)
      N2=2*N+II
      A(N2,2*I-1)=ATEMP
      A(N2,2*I)=-ATEMP
   35 CONTINUE
C     SET UP RIGHT HAND SIDES OF LINEARIZED INEQUALITY CONSTRAINTS AND
```

```
C     ADD SLACK VARIABLES
      DO 36 I=1,NCONS
      I2=2*N+I
      A(I2,I2)=-SIGN(I)
      B(I2)=-PART(I)*SIGN(I)
   36 CONTINUE
C     LINEARIZE THE  EQUALITY  CONSTRAINTS, MULTIPLYING THROUGH BY -1.C
C     IF THE RIGHT HAND SIDE IS NEGATIVE
   48 IF(NEQUS.EQ.0)GOTO52
      DO 47 I=1,NEQUS
   47 PART(I)=0.0
      CALL EQUAL(X,PART,NEQUS)
      DO 49 I=1,NEQUS
      SIGN(I)=1.0
      IF(-PART(I).LT.0.0)SIGN(I)=-1.0
   49 CONTINUE
      DO 50 I=1,N
      X(I)=X(I)+DELX(I)
      CALL EQUAL(X,PSI,NEQUS)
      X(I)=X(I)-DELX(I)
      DO 50 II=1,NEQUS
      ATEMP=SIGN(II)*(PSI(II)-PART(II))/DELX(I)
      II2=2*N+NCONS+II
      A(II2,2*I-1)=ATEMP
      A(II2,2*I)=-ATEMP
   50 CONTINUE
C     SET UP RIGHT HAND SIDES OF LINEARIZED EQUALITY CONSTRAINTS
      DO 51 I=1,NEQUS
      II2=2*N+NCONS+I
      B(II2)=-PART(I)*SIGN(I)
   51 CONTINUE
   52 RETURN
      END


      SUBROUTINE FEASBL(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,
     1DSTAR,NTERMS,NTOTER)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1),DSTAR(NTOTER
     1,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A5/STEPP(100)
C     THIS SUBROUTINE USES  SEEK3  TO DRIVE ALL PHI(I) FEASIBLE AND THEN
C     REDUCES THE PSI(I)S BY MINIMIZING  SIGMA(PSI(I)) SUBJECT TO THE
C     CONDITION THAT ALL PHI(I) REMAIN FEASIBLE(.GE.0.)
      NNDEX=INDEX
      KUT=0
      DO 9 I=1,N
    9 X(I)=XSTRT(I)
      IF(NCONS.EQ.0)GOTO13
      CALL CONST(X,NCONS,PHI)
      DO 10 I=1,NCONS
```

```
      IF(PHI(I).LT.0.0)GOTO11
   10 CONTINUE
      GOTO13
C     IF ANY PHI(I).LT.0. CALL SEEK3 TO DRIVE THEM FEASIBLE
   11 CALL SEEK3(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DSTAR,NT
     1ERMS,NTOTER)
      IF(NVIOL.EQ.0)GOTO13
C     IF SEEK3 COULD NOT  GET ALL PHI(I).GE.0. THEN SUBR.FEASBL CANNOT
C     OBTAIN A FEASIBLE POINT
      KO=1
      GOTO31
   13 IF(NEQUS.EQ.0)GOTO31
C     MINIMIZE SIGMA(PSI(I)) KEEPING ALL PHI(I).GE.0.
C     NOTE...THE FRACTION OF THE RANGE USED AS STEP SIZE SHOULD NOT
C     EXCEED 5 PERCENT. IF THE USER IS INTERESTED IN A VERY FEASIBLE
C     POINT(IE.ALL PSI(I)S VERY SMALL)HE CAN GIVE (F) A VERY SMALL VALUE
      PERCNT=0.05
      IF(ABS(F).LT.0.05)PERCNT=F
      DO 14 I=1,N
   14 STEPP(I)=PERCNT*(RMAX(I)-RMIN(I))
C     INITIALIZE  THE SUM OF THE PSI(I)S
      CALL SUMPSI(X,PSI,NEQUS,SUMO)
   15 NFAIL=0
      DO 25 I=1,N
      X(I)=X(I)+STEPP(I)
      CALL CONST(X,NCONS,PHI)
      DO 17 J=1,NCONS
C     IGNORE A MOVE WHICH MAKES ANY PHI(I).LT.0.0
      IF(PHI(J).LT.0.0)GOTO19
   17 CONTINUE
      CALL SUMPSI(X,PSI,NEQUS,SUM1)
      IF(SUM1.GE.SUMO)GOTO19
      SUMO=SUM1
      GOTO25
   19 X(I)=X(I)-2.0*STEPP(I)
      CALL CONST(X,NCONS,PHI)
      DO 21 L=1,NCONS
      IF(PHI(L).LT.0.0)GOTO23
   21 CONTINUE
      CALL SUMPSI(X,PSI,NEQUS,SUM2)
      IF(SUM2.GE.SUMO)GOTO23
      SUMO=SUM2
      GOTO25
   23 X(I)=X(I)+STEPP(I)
      NFAIL=NFAIL+1
   25 CONTINUE
      IF(NFAIL.EQ.N)GOTO27
      GOTO15
C     REDUCE STEPP(I) BY A FACTOR OF 4.0 UP TO 4 TIMES. THIS MEANS STEPP
C     REDUCES TO  LESS THAN .0002*(RMAX(I)-RMIN(I)), OR IF  F.LT.0.05
C     THEN MINIMUM STEPP(I)=(F/256)*(RMAX(I)-RMIN(I)). THEREFORE THE
C     USER MAY DRIVE THE PSI(I) VALUES AS SMALL AS HE WISHES BY ENTERING
C     A VERY SMALL VALUE OF  F  AT LEVEL=1
   27 KUT=KUT+1
```

```
      IF(KUT.GT.4)GOTO31
      DO 29 I=1,N
   29 STEPP(I)=STEPP(I)/4.0
      GOTO15
   31 CALL UREAL(X,U)
C     ZERO STEPP(I) SINCE BLOCK /A5/ IS USED BY CALLING METHODS
      DO 33 I=1,N
   33 STEPP(I)=0.0
      RETURN
      END


      SUBROUTINE SUMPSI(X,PSI,NEQUS,SUM)
      DIMENSION X(1),PSI(1)
      CALL EQUAL(X,PSI,NEQUS)
      SUM=0.0
      DO 1 I=1,NEQUS
      SUM=SUM + ABS(PSI(I))
    1 CONTINUE
      RETURN
      END


      SUBROUTINE RANDOM (X,U,N,RMAX,RMIN,Z,UU,NRET,NCONS,PHI)
      DIMENSION X(1),RMAX(1),RMIN(1),Z(NRET,1),UU(1),PHI(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,Q,REDUCE,NVIOL,KO,NNDEX
      COMMON/A1/AA(100),CC(100),WORK3(100),TEST1(100)
      COMMON /A5/R(100)
C
C     OPTIMIZATION USING DICKINSONS RANDOM SEARCH STRATEGY
C
      WRITE (6,200)
C     RANDOM DOES NOT HANDLE INEQUALITY CONSTRAINTS AND THEREFORE NEQUS
C     IS NOT INPUT.  SET NEQUS=0 TO AVOID GETTING AN INDEFINITE MESSAGE
      NEQUS=0
      NCYCLE=1
      DO 18 I=1,N
      CC(I)=0.
      AA(I)=0.
      TEST1(I)=0.
      X(I)=0.0
   18 CONTINUE
      DO 22 I=1,N
      CC(I)=RMAX(I)
      AA(I)=RMIN(I)
   22 TEST1(I)=F*ABS(CC(I)-AA(I))
C     NUMR IS THE NUMBER OF FEASIBLE RANDOM POINTS EVALUATED EACH CYCLE
      NUMR=NRET*NSHRIN
C     THE NUMBER OF FEASIBLE RANDOM POINTS RETAINED EACH CYCLE IS
C     NRET=NUMR/NSHRIN AND NRET ARRIVES THROUGH THE ARGUMENT LIST
C     GENERATE NRET FEASIBLE RANDOM POINTS
```

```
C     MSTART IS THE STARTING VALUE FOR GENERATING RANDOM NUMBERS.
C     AT LEVEL=0 MSTART=128 IS SET IN OPTIPAC. AT LEVEL=1 MSTART IS DATA
      MM=MSTART
      DO 21 J=1,NRET
      L=1
   50 CONTINUE
      CALL FRANDN(R,N,MM)
      MM=0
      DO 20 I=1,N
   20 X(I)=AA(I)+R(I)*(CC(I)-AA(I))
      IF(NCONS.EQ.0)GOTO52
      CALL CONST(X,NCONS,PHI)
      NVIOL=0
      DO 42 I=1,NCONS
      IF(PHI(I).GE.0.0)GOTO42
      NVIOL=NVIOL+1
   42 CONTINUE
      IF(NVIOL.EQ.0)GOTO52
      L=L+1
      IF (L.GT.NSMAX) GO TO 80
      GO TO 50
   52 CALL UREAL(X,UTEMP)
      DO 43 I=1,N
   43 Z(J,I)=X(I)
      UU(J)=UTEMP
   21 CONTINUE
C     FIND LARGEST VALUE OF UU(J)
      LARGE=1
      DO 10 J=2,NRET
      IF(UU(J).LE.UU(LARGE))GOTO10
      LARGE=J
   10 CONTINUE
C     PLACE LARGEST VALUE OF UU(J) AT UU(1) AND INTERCHANGE Z(J,I) WITH
C     Z(1,I)
      UTEMP=UU(LARGE)
      UU(LARGE)=UU(1)
      UU(1)=UTEMP
      DO 11 I=1,N
      ZTEMP=Z(LARGE,I)
      Z(LARGE,I)=Z(1,I)
      Z(1,I)=ZTEMP
   11 CONTINUE
C     GENERATE NUMR MORE FEASIBLE POINTS AND IF ANY HAS UU(J).LT.UU(1)
C     THEN INTERCHANGE THEM
      KK=1
   60 DO 12 K=1,NUMR
      L=1
   53 CONTINUE
      CALL FRANDN(R,N,0)
      DO 13 I=1,N
   13 X(I)=AA(I)+R(I)*(CC(I)-AA(I))
      IF(NCONS.EQ.0)GOTO55
      CALL CONST(X,NCONS,PHI)
```

```
      NVIOL=0
      DO 56 I=1,NCONS
      IF(PHI(I).GE.0.0)GOTO56
      NVIOL=NVIOL+1
   56 CONTINUE
      IF(NVIOL.LT.1)GOTO55
      L=L+1
      IF (L.GT.NSMAX) GO TO 80
      GO TO 53
   55 CALL UREAL(X,UXTRA)
      IF (UXTRA.GE.UU(1)) GO TO 12
      UU(1)=UXTRA
      DO 14 I=1,N
   14 Z(1,I)=X(I)
C     PUT NEW LARGEST UU(J) AT UU(1)
      DO 30 J=2,NRET
      IF (UU(J).LE.UU(1)) GO TO 30
      UTEMP=UU(J)
      UU(J)=UU(1)
      UU(1)=UTEMP
      DO 31 I=1,N
      XTEMP=Z(J,I)
      Z(J,I)=Z(1,I)
   31 Z(1,I)=XTEMP
   30 CONTINUE
   12 CONTINUE
C     SELECT NEW AA(I) AND CC(I)
      DO 15 I=1,N
      AA(I)=Z(1,I)
      CC(I)=Z(1,I)
      DO 16 J=2,NRET
      IF (Z(J,I).GT.AA(I)) GO TO 17
      AA(I)=Z(J,I)
      GO TO 16
   17 IF (Z(J,I).LT.CC(I)) GO TO 16
      CC(I)=Z(J,I)
   16 CONTINUE
   15 CONTINUE
      IF (KK-IPRINT) 27,28,62
   27 KK=KK+1
      GOTO62
   28 IF(NCYCLE.EQ.IPRINT)WRITE(6,9)
      WRITE(6,8)NCYCLE,UU(1)
      L2=0
   29 L1=L2+1
      L2=L1+4
      IF(L2.GT.N)L2=N
      WRITE(6,4)(CC(I),I=L1,L2)
      WRITE(6,2)(AA(I),I=L1,L2)
      IF(L2.LT.N)GOTO29
      KK=1
   62 IF(NCYCLE.GE.MAXM)GOTO61
      NCYCLE=NCYCLE+1
      DO 63 I=1,N
```

```
      IF(ABS(CC(I)-AA(I)).GT.ABS(TEST1(I)))GOTO60
   63 CONTINUE
C     SELECT SMALLEST UU(J)
   61 JMIN=2
      DO 19 J=3,NRET
      IF(UU(J).GE.UU(JMIN))GOTO19
      JMIN=J
   19 CONTINUE
      DO 54 I=1,N
   54 X(I)=Z(JMIN,I)
      IF(NCYCLE.GE.MAXM)GOTO100
      GOTO81
   80 WRITE(6,3)NSMAX
      WRITE(6,5)
      KO=1
      RETURN
  100 WRITE (6,6) MAXM
      KO=1
   81 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
    2 FORMAT(6X,5E16.8)
    3 FORMAT(1H0,40HNO FEASIBLE POINT FOUND AFTER GENERATING,I6,16H  RAN
     1DOM NUMBERS)
    4 FORMAT(1H0,5X,5E16.8)
    5 FORMAT(1X ,54HTRY SHRINKING THE RANGE OR INCREASING NSMAX AT LEVEL
     1=1/)
    6 FORMAT(//34H PROCESS FAILED TO CONVERGE AFTER ,I4,2X,6HCYCLES)
    8 FORMAT(1H0,I3,3H  (,E15.8,1H))
    9 FORMAT(6H-CYCLE,5X,6H(UMAX),22X,26HUPPER/LOWER BOUNDS ON X(I)//)
  200 FORMAT(1H1,58HOPTIMIZATION USING DICKINSONS RANDOM SEARCH METHOD
     1RANDOM//)
      RETURN
      END


      SUBROUTINE FRANDN(A,N,M)
      DIMENSION A(1)
C     THIS RANDOM NUMBER GENERATOR IS A MODIFIED  IBM  SUBROUTINE
C     B  IS A MACHINE-DEPENDENT CONSTANT AND  B=2.0**(I/2+1)+3.0
C     WHERE  I = NUMBER OF BITS IN AN INTEGER WORD (I=47 FOR CDC6400)
      B=262147.0
      X=M
      X=X/0.8719467
   20 IF(X.NE.0.0)Y=AMOD(ABS(X),3.18967)
      DO 10 K=1,N
      DO 11 J=1,2
   11 Y=AMOD(B*Y,1.0)
      A(K)=Y
C     AVOID Y=0. AND Y=1. TO PREVENT DIVIDING INTO ZERO
   10 IF(Y.EQ.0.0.OR.Y.EQ.1.0)Y=0.182818285
      RETURN
      END
```

```
      SUBROUTINE GEOM(NTOTER,N,NCONS,NTERMS,EX,CONST,AA,BB,C,DSTAR,RMAX,
     1RMIN,X,XSTRT)
      DIMENSION NTERMS(1),EX(NTOTER,1),CONST(1),AA(NTOTER,1),BB(NTOTER,1
     1),C(NTOTER,1),DSTAR(NTOTER,1),RMAX(1),RMIN(1),X(1),XSTRT(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A3/CK(100),GAM(100),T(100)
      COMMON /A5/D(100)
      COMMON /A7/SUM(100),USE(100)
      COMMON /A8/NUSE(100)
C
C     THE  GEOMETRIC  PROGRAMMING  METHOD  OF  OPTIMIZATION
C
C     THE PROGRAM IS DIVIDED INTO FIVE SECTIONS AS FOLLOWS.(NOTATION AS
C     IN MATHEMATICAL DESCRIPTION GIVEN IN LEVEL 1 DOCUMENTATION).
C
C       1.    CALCULATION OF THE DELTA SUB I SUPER J ARRAY
C       2.    RELAXATION METHOD TO FIND FEASIBLE STARTING VALUES OF T(I)
C       3.    CALCULATION OF THE  K SUB Q  VECTOR
C       4.    MAXIMIZATION OF DUAL BY DIRECT SEARCH (SEEK1)
C       5.    CONVERSION FROM DUAL BACK TO PRIMAL PROBLEM
C
C     SECTION  (1)
C
C     CALCULATION OF THE SET OF NORMAL AND NULL VECTORS  =  DELTA SUB I
C     SUPER J. THESE ARE DERIVED FROM THE INPUT EXPONENT ARRAY (EX).
C
C     NOTE...KO=0 INITIALLY. KO=1 IF A FAILURE OCCURS ANYWHERE IN GEOM.
      NT=NCONS+1
      NM=NTOTER-N
      N1=N+1
C
C     TRANSPOSE THE ROWS AND COLUMNS OF THE EXPONENT ARRAY (EX)INTO (AA)
C
   10 DO 11 I=1,NTOTER
      DO 11 J=1,N
   11 AA(J,I)=EX(I,J)
C
C     GAUSS REDUCE THE MATRIX (AA) BY ROWS KEEPING TRACK OF COLUMN INTER
C     -CHANGES. THIS CHANGES THE (AA) MATRIX INTO A UNIT MATRIX IN THE
C     N BY N POSITIONS AND MODIFIES ELEMENTS IN THE N BY (N1 TO NTOTER)
C     POSITIONS.THESE OPERATIONS ARE PERFORMED WITHIN SUBR. GAJON.
C     NOTE...ARRAY NUSE IS COMMONED BETWEEN GEOM AND GAJON.
C
      CALL GAJON(AA,NTOTER,N)
      IF(KO.NE.0)RETURN
C
C     FORM THE MATRIX (C)...IN THE  N BY NM POSITIONS OF (C) PLACE THE
C     NEGATIVES OF THE N BY (N+1)TO(NTOTER) ELEMENTS OF THE REDUCED (AA)
C     SET EQUAL TO 1 ALL (C) ELEMENTS FOR WHICH I=J+N. SET REMAINING (C)
C     ELEMENTS EQUAL TO ZERO.
C
      DO 12 I=1,N
      DO 12 J=N1,NTOTER
      JJ=J-N
```

```
   12 C(I,JJ)=-AA(I,J)
      DO 13 I=N1,NTOTER
      DO 13 J=1,NM
      JJ=J+N
      IF(I.EQ.JJ) GO TO 14
      C(I,J)=0.0
      GO TO 13
   14 C(I,J)=1.0
   13 CONTINUE
C
C     FOR EVERY COLUMN INTERCHANGE (STORED IN NUSE) MADE IN THE GAUSS
C     REDUCTION MAKE THE CORRESPONDING ROW INTERCHANGE IN THE MATRIX (C)
C     CALL THE RESULTING MATRIX (BB).
C
      DO 15 I=1,NTOTER
      DO 15 J=1,NM
      NISE=NUSE(I)
   15 BB(NISE,J)=C(I,J)
      DO 16 I=1,NM
      NUSE(I)=0
      RMIN(I)=0.0
      RMAX(I)=0.0
   16 SUM(I)=0.0
C
C     SUM THE FIRST NTERMS(1) ELEMENTS OF EVERY COLUMN OF (BB).
C
      NTER=NTERMS(1)
      DO 17 I=1,NM
      DO 17 J=1,NTER
   17 SUM(I)=SUM(I)+BB(J,I)
C
C     FIND THE FIRST COLUMN OF (BB) HAVING THE SUM OF ITS FIRST
C     NTERMS(1) ELEMENTS = SUM NOT EQUAL TO ZERO. DIVIDE EACH ELEMENT
C     IN THAT COLUMN BY SUM AND STORE THE RESULT IN DSTAR(J,1). THIS IS
C     THE DELTA SUB I SUPER 0 VECTOR.
C
      I=0
   18 I=I+1
      IF(I.GT.NM) GO TO 19
      IF(ABS(SUM(I)).GT.1.0E-8) GO TO 20
      GO TO 18
C     ARRAY (BB) MUST BE SINGULAR.
   19 WRITE(6,601)
      KO=1
      RETURN
   20 NUSE(I)=1
      DO 21 J=1,NTOTER
   21 DSTAR(J,1)=BB(J,I)/SUM(I)
C
C     COMPLETE THE DSTAR ARRAY--DSTAR(J,II)=BB(J,I)-SUM(I)*DSTAR(J,1)
C
      II=1
      DO 23 I=1,NM
      IF(NUSE(I).NE.0) GO TO 23
```

```
      II=II+1
      DO 22 J=1,NTOTER
   22 DSTAR(J,II)=BB(J,I)-SUM(I)*DSTAR(J,1)
   23 CONTINUE
C
C
C     SECTION  (2)
C
C     CALCULATION OF INITIAL VALUES OF THE THE DUAL VARIABLES T USING A
C     RELAXATION TECHNIQUE.
C     THE T VALUES ARE FEASIBLE IF FOR  I=1,NTOTER  THE FOLLOWING EQNS
C     HOLD...  0.0 .LE. DSTAR(I,1)+DSTAR(I,J)*T(J)  J=2,NM. (THESE SUMS
C     REPRESENT THE DELTA SUB I VECTOR IN THE MATHEMATICAL DESCRIPTION).
C
      KOUNT=0
      LIMIT=300*NM
C     NOTE...LIMIT DOES NOT STOP THE PROGRAM - SEE COMMENT BELOW.
      DO 97 I=1,NTOTER
   97 USE(I)=0.0
C     START WITH ALL DUAL VARIABLES T EQUAL TO ZERO
      DO 98 I=1,NM
   98 T(I)=0.0
C     CALCULATE THE SQRT OF THE SUM OF THE SQUARES OF ELEMENTS 2 TO NM
C     IN EACH ROW OF DSTAR.STORE THE RESULTS IN ARRAY (USE).
      DO 800 I=1,NTOTER
      DO 805  J=2,NM
  805 USE(I)=USE(I)+DSTAR(I,J)**2
  800 USE(I)=SQRT(ABS(USE(I)))
C
C     NORMALIZE THE (DSTAR) ARRAY BY DIVIDING ALL ELEMENTS IN A ROW BY
C     THE ROWS VALUE OF (USE). IF AN ELEMENT IS ZERO(LESS THAN 1.E-08)
C     LEAVE IT ZERO.IF A FIRST COLUMN ELEMENT (DSTAR(I,1)) IS ZERO,FORCE
C     IT NEGATIVE BY ADDING -1.E-06 .STORE THE MODIFIED (DSTAR) IN (BB).
C
      DO 801 I=1,NTOTER
      DO 801 J=1,NM
C     TEST AGAINST 1.E-08 RATHER THAN 0.0 TO ALLOW FOR ROUNDING ERROR.
      IF(USE(I).GT.1.0E-08)GOTO802
      BB(I,J)=DSTAR(I,J)
      GOTO801
  802 IF(J.EQ.1)GOTO103
      GOTO104
  103 BB(I,J)=(DSTAR(I,J)-1.0E-06)/USE(I)
      GOTO801
  104 BB(I,J)=(DSTAR(I,J))/USE(I)
  801 CONTINUE
  111 KOUNT=KOUNT+1
      IF(KOUNT.LT.LIMIT)GOTO105
C     IF NO FEASIBLE STARTING VALUES FOR T HAVE BEEN FOUND AFTER (LIMIT)
C     STEPS OF RELAXATION PROCEDURE, GO DIRECTLY TO SUBR.SEEK1 WHICH IS
C     CAPABLE OF FINDING ITS OWN STARTING VALUES.
      GO TO 203
C
C     CALCULATE THE DELTA SUB I VECTOR, STORING IT IN (USE).
```

```
C
  105 DO 106 I=1,NTOTER
      USE(I)=BB(I,1)
      DO 106 J=2,NM
      USE(I)=USE(I)+BB(I,J)*T(J)
  106 CONTINUE
C
C     FIND THE MOST NEGATIVE ELEMENT OF (USE), CALL IT SN AND CALL ITS
C     SUBSCRIPT IQ. WHEN NO NEGATIVE ELEMENTS ARE FOUND WE HAVE A SET OF
C     FEASIBLE T VARIABLES.
C
      SN=0.0
      DO 109 I=1,NTOTER
      IF(USE(I).GE.0.0)GOTO109
      IF(USE(I).LT.SN)GOTO108
      GOTO109
  108 IQ=I
      SN=USE(I)
  109 CONTINUE
C     SN MAY CONVERGE TO ZERO VERY SLOWLY,THEREFORE TEST AGAINST -1.E-08
      IF(SN.LT.-1.0E-08)GOTO110
      GOTO203
C
C     MODIFY THE T VALUES AND REPEAT THE ABOVE PROCEDURE.
C
  110 DO 107 J=2,NM
      T(J)=T(J)-BB(IQ,J)*SN
  107 CONTINUE
      GOTO111
C
C
C     SECTION  (3)
C
C     CALCULATION OF THE  K SUB Q  VECTOR (STORED IN (CK)).
C
  203 DO 200 IQ=1,NM
      CK(IQ)=DSTAR(1,IQ)*ALOG(CONST(1))
      DO 201 II=2,NTOTER
  201 CK(IQ)=CK(IQ)+DSTAR(II,IQ)*ALOG(CONST(II))
  200 CK(IQ)=EXP(CK(IQ))
C
C
C     SECTION  (4)
C
C     MAXIMIZE THE DUAL FUNCTION SN BY DIRECT SEARCH - SUBR SEEK1 THE
C     SEARCH STOPS WHEN NO INCREASE  IN SN IS OBTAINED BY CHANGING ANY
C     T VALUE BY +O-- F*G*RANGE (SEE LEVEL 1 DOCUMENTATION).
C     USE T VALUES FROM RELAXATION  AS STARTING VALUES FOR SEEK1  AND
C     SET RANGES OF T(I) VALUES TO ESTABLISH INITIAL STEP SIZE IN SEEK1
  207 DO 811 I=2,NM
      RMIN(I)=T(I)-0.50*ABS(T(I))
      RMAX(I)=T(I)+0.50*ABS(T(I))
  811 XSTRT(I)=T(I)
C     X  MUST BE FIRST ARGUMENT FOR SEEK1 TO PRESERVE VARIABLE DIMENSION
      CALL  SEEK1(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,SN,DSTAR,NTE
```

```
     1RMS,NTOTER)
      IF(KO.EQ.0)GOTO812
      WRITE(6,615)
      GOTO9999
  812 DUAL=-SN
      DO 813 I=2,NM
  813 T(I)=X(I)
C
C
C     SECTION  (5)
C
C     CONVERT FROM THE DUAL PROBLEM BACK TO THE PRIMAL (INPUT) PROBLEM.
C
C     FORM THE RIGHT HAND SIDE OF THE SET OF LINEAR EQNS IN THE UNKNOWNS
C     LOG(X(I)). DUAL, CONST(I), D(I), AND GAM(I) ARE ALL KNOWN AT THIS
C     STAGE. STORE THE R.H.S. IN AA(I,N1) FOR TRANSFER TO SUBR GELIM.
C
      NTEMP2=NTERMS(1)
      DO 700 I=1,NTEMP2
  700 AA(I,N1)=ALOG(D(I)*ABS(DUAL)/CONST(I))
      LYM1=1
      LYM2=NTERMS(1)
      DO 702 IQ=1,NCONS
      LYM1=LYM1+NTERMS(IQ)
      IQ1=IQ+1
      LYM2=LYM2+NTERMS(IQ1)
      DO 702 I=LYM1,LYM2
      AA(I,N1)=ALOG(D(I)/(CONST(I)*GAM(IQ)))
  702 CONTINUE
C
C     COEFFICIENTS OF THE UNKNOWN VARIABLES LOG(X(I)) ARE SIMPLY THE
C     ELEMENTS OF THE INPUT EXPONENT ARRAY (EX).
C
      DO 703 I=1,NTOTER
      DO 703 J=1,N
  703 AA(I,J)=EX(I,J)
C
C     CALL SUBR GELIM TO SOLVE THE SET OF EQNS BY GAUSS ELIMINATION.
C     NOTE... THE LOG(X) VALUES ARE RETURNED FROM GELIM IN AA(I,N1).
C
      CALL GELIM(NTOTER,N,AA)
      IF(KO.NE.0)GOTO9999
C
C     CALCULATE THE PRIMAL OPTIMIZATION FUNCTION FROM THE LOG(X) VALUES.
C
      DO 704 I=1,N
  704 USE(I)=AA(I,N1)
      SN=0.0
      NHI=NTERMS(1)
      DO 705 I=1,NHI
      PP=0.0
      DO 706 J=1,N
  706 PP=PP+EX(I,J)*USE(J)
  705 SN=SN+CONST(I)*EXP(PP)
      PRIMAL=SN
```

```
C      NOTE...THE VALUES OF PRIMAL AND DUAL SHOULD AGGREE TO SEVERAL
C      DECIMAL PLACES AT THE GLOBAL OPTIMUM
C
C      CONVERT LOG(X) VALUES TO X VALUES.
C
       DO 707 I=1,N
  707  X(I)=EXP(USE(I))
C
C      CALCULATE THE VALUES OF THE ORIGINAL(PRIMAL) CONSTRAINT EQUATIONS
C      ALL OF WHICH SHOULD BE  .LE.1.0 (PLACE RESULTS IN WORKING ARRAY
C      SUM(100)
C
       L1=NTERMS(1)+1
       DO 710 I=2,NT
       L2=L1+NTERMS(+)-1
       SUM(I)=0.0
       DO 709 K=L1,L2
       TERM=CONST(K)
       DO 708 J=1,N
  708  TERM=TERM*X(J)**EX(K,J)
  709  SUM(I)=SUM(I)+TERM
       L1=L2+1
  710  CONTINUE
C
C      PRINT OUT RESULTS
C
       WRITE(6,610)
       WRITE(6,611)PRIMAL
       WRITE(6,612)DUAL
       WRITE(6,613)(I,X(I),I=1,N)
       WRITE(6,614)
       WRITE(6,618)(I,SUM(I+1),I=1,NCONS)
       WRITE(6,616)
       WRITE(6,617)
  601  FORMAT(1H-,22HARRAY (BB) IS SINGULAR)
  610  FORMAT(1H1,24X,30HOPTIMUM SOLUTION FOUND BY GEOM/25X,30H----------
      1--------------------/)
  611  FORMAT(19X,15HMINIMUM  U(X) =,E16.8,9H (PRIMAL))
  612  FORMAT(19X,15HMAXIMUM  U(T) =,E16.8,7H (DUAL)///)
  613  FORMAT(27X,2HX(,I2,3H) =,E16.8)
  614  FORMAT(1H-,24H  INEQUALITY CONSTRAINTS/1X,24H(FEASIBLE PHI(I).LE.1
      1.0))
  615  FORMAT(1H-,47HSUBR.SEEK1 UNABLE TO MAXIMIZE THE DUAL FUNCTION/)
  616  FORMAT(1H-,73HNOTE...THE VALUES OF THE PRIMAL AND DUAL OPTIMIZATIO
      1N FUNCTIONS ESTABLISH/1X,73HUPPER AND LOWER BOUNDS RESPECTIVELY ON
      2 THE GLOBAL O-TIMUM. IF THEY DO NOT)
  617  FORMAT(1X,68HAGREE TO SEVERAL DECIMAL PLACES, TRY REDUCING F AND G
      1 TO IMPROVE THE/1X,21HMAXIMIZATION IN SEEK1/)
  618  FORMAT(25X,4HPHI(,I2,3H) =,E16.8)
 9999  RETURN
       END
```

```
      SUBROUTINE GAJON(AA,NTOTER,N)
      DIMENSION AA(NTOTER,1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A8/NUSE(100)
C
C
C     THIS SUBR. PERFORMS A GAUSS-JORDAN REDUCTION BY ROWS OF THE MATRIX
C     (AA) KEEPING TRACK OF COLUMN INTERCHANGES IN ARRAY (NUSE). THE
C     RESULT IS A UNIT MATRIX IN THE N BY N POSITIONS (OFF-DIAGONAL
C     ELEMENTS ARE SET =0.0 AFTER RETURN TO GEOM) AND A MODIFIED ARRAY
C     IN THE N BY (N1 TO NTOTER) POSITIONS (THE NEGATIVES OF THESE FORM
C     THE  N BY NM ELEMENTS OF (C) AFTER RETURN TO GEOM). NOTE... (NUSE)
C     IS NEEDED IN GEOM AND IS CARRIED THROUGH COMMON.
C
      NN=0
      NT1=N-1
      NT4=NTOTER-1
      DO 10 I=1,NTOTER
   10 NUSE(I)=I
C
C     SEARCH THE NNTH ROW FOR FIRST NON-ZERO ELEMENT. INTERCHANGE THAT
C     COLUMN WITH THE KTH COLUMN.
C
  101 NN=NN+1
      K=NN-1
   11 K=K+1
      IF(ABS(AA(NN,K)).GT.1.0E-6) GO TO 12
      IF(K.LE.NT4)GOTO11
C     A ROW OF (AA) IS ENTIRELY ZEROS IE. THE MATRIX IS SINGULAR.SINCE
C     AA IS THE TRANSPOSE OF EX, THIS  MEANS THAT ONE OF THE INPUT
C     VARIABLES DOES NOT APPEAR IN ANY TERM
      WRITE(6,20)K,K
      KO=1
      GO TO 13
   12 IF(K.EQ.NN) GO TO 14
      DO 15 I=1,N
      TEMP=AA(I,NN)
      AA(I,NN)=AA(I,K)
   15 AA(I,K),,EMP
      NTEMP=NUSE(NN)
      NUSE(NN)=NUSE(K)
      NUSE(K)=NTEMP
C
C     DIVIDE THE NNTH ROW BY  THE DIAGONAL ELEMENT IN IT (AA(NN,NN))
C
   14 J=NTOTER+1
  141 J=J-1
      IF(J.LT.NN) GO TO 16
      AA(NN,J)=AA(NN,J)/AA(NN,NN)
      GO TO 141
```

```
C     REDUCE ALL ROWS BELOW THE NNTH ROW.
   16 NA=NN+1
      IF(NA.GT.N)GO TO 171
      DO 17 I=NA,N
      DO 17 J=NA,NTOTER
   17 AA(I,J)=AA(I,J)-AA(I,NN)*AA(NN,J)
  171 IF(NN.LT.N) GO TO 101
      DO 18 I=1,NT1
      NT2=I+1
      DO 18 NL=NT2,N
      NT3=NL+1
      DO 18 J=NT3,NTOTER
   18 AA(I,J)=AA(I,J)-AA(I,NL)*AA(NL,J)
   20 FORMAT(1H-,38HTHE EXPONENT ARRAY IS SINGULAR IN ROW ,I4/1X,13HTHAT
     1 IS, THE ,I2,59H TH  INPUT VARIABLE DOES NOT APPEAR IN ANY OF THE
     2RELATIONS/)
   13 RETURN
      END



      SUBROUTINE GELIM(NTOTER,N,AA)
      DIMENSION AA(NTOTER,1)
      COMMON /A5/D(100)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
C
C     THIS SUBR. USES GAUSS ELIMINATION TO SOLVE A SET OF NTOTER EQNS
C     IN N UNKNOWNS WITH ONE RIGHT HAND SIDE. THE  COEFFICIENTS ENTER
C     THE SUBR. IN THE NTOTER BY N POSITIONS OF (AA). THE R.H.S. IS
C     STORED IN THE VECTOR AA(I,N1). THE SOLUTION VECTOR (IN THIS CASE
C     THE SET OF LOG(X) VALUES) IS RETURNED IN AA(I,N1).
C     NOTE...GELIM -EQUIRES THAT  NTOTER.GE.N
      KOUNT=NTOTER
      KO=0
      N1=N+1
C     THE ARGUMENT OF ALOG() MUST BE POSITIVE,THEREFORE DISCARD ANY
C     EQUATION FOR WHICH  D(I).LE.0.
C     IF ANY D(I).LE.0.0 , THEN ZERO THE CORRESPONDING ROW IN (AA) AND
C     DECREMENT KOUNT. (IF KOUNT.LT.N THEN THE MATRIX IS SINGULAR).
      I=0
  101 I=I+1
      IF(I.GT.NTOTER)GOTO102
      IF(D(I).GT.1.0E-10) GO TO 101
C     TEST AGAINST 1.E-10 RATHER THAN 0.0 TO ALLOW FOR ROUNDING ERROR.
      KOUNT=KOUNT-1
      DO 10 J=1,N1
   10 AA(I,J)=0.0
      GO TO 101
  102 CONTINUE
C     CHECK TO SEE IF THERE ARE SUFFICIENT VALID EQUATIONS REMAINING.
C     IF THERE ARE LESS THAN N EQUATIONS, THE N UNKNOWNS CANNOT BE
C     SOLVED FOR
      IF(KOUNT.LT.N)GOTO11
```

```
      KN=0
      GO TO 12
   11 WRITE(6,610)
      KO=1
      GO TO 99
C
C     LOCATE THE FIRST NON-ZERO ELEMENT IN THE KNTH COLUMN (AND KTH ROW)
C
   12 KN=KN+1
      K=KN-1
  121 K=K+1
      IF(ABS(AA(K,KN)).GT.1.0E-10)GOTO13
      IF(K.LT.NTOTER)GOTO121
      KO=1
      WRITE(6,611)KN
      GO TO 99
C
C     INTERCHANGE THE KTH AND KNTH ROWS.
C
   13 IF(K.EQ.KN) GO TO 15
      DO 14 I=KN,N1
      TEMP=AA(KN,I)
      AA(KN,I)=AA(K,I)
   14 AA(K,I)=TEMP
C
C     DIVIDE THE NEW KNTH ROW BY ITS DIAGONAL ELEMENT AA(KN,KN).
C
   15 J=N1+1
  151 J=J-1
      IF(J.LT.KN) GO TO 16
      AA(KN,J)=AA(KN,J)/AA(KN,KN)
      GO TO 151
   16 KN1=KN+1
C
C     REDUCE ALL ROWS BELOW THE KNTH ROW.
C
      DO 17 I=KN1,NTOTER
      PMULT=AA(I,KN)
      DO 17 J=KN,N1
   17 AA(I,J)=AA(I,J)-PMULT*AA(KN,J)
      IF(KN.LT.N)GOTO12
      NV1=N-1
      DO 18 I=1,NV1
      IPLUS=I+1
      DO 18 II=IPLUS,N
      PMULT=AA(I,II)
      DO 18 J=II,N1
   18 AA(I,J)=AA(I,J)-PMULT*AA(II,J)
  610 FORMAT(43H- CANNOT MAKE DUAL TO PRIMAL TRANSFORMATION)
  611 FORMAT(1H-,48HTHE MATRIX PASSED TO GELIM IS SINGULAR IN COLUMN,I3)
   99 RETURN
      END
```

```
      SUBROUTINE GEOPT(NTOTER,N,NCONS,NTERMS,DSTAR,SN,T)
      DIMENSION NTE-MS(1),DSTAR(NTOTER,1),T(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,R,REDUCE,NVIOL,KO,NNDEX
      COMMON /A3/CK(100),GAM(100),WORK11(100)
      COMMON /A5/D(100)
C     GEOPT IS CALLED FROM SEEKU , HENCE T IS VARIABLY DIMENSIONED
C
C     THIS SUBR.  EVALUATES THE OPTIMIZATION FUNCTION FOR A GIVEN SET
C     OF VARIABLES T. PENALTY FUNCTIONS ARE ADDED IF ANY CONSTRAINTS
C     ARE VIOLATED.GEOPT IS THE ANALOGUE OF (OPTIMF) USED ELSEWHERE IN
C     OPTIPAC.
C     NOTE*** SUBR.SEARCH WHICH CALLS GEOPT IS A MINIMIZATION TECHNIQUE
C     THEREFORE THE NEGATIVE OF THE OPTIMIZATION FUNCTION IS RETURNED.
C     THAT IS, MINIMIZING (-SN) IS EQUIVALENT TO MAXIMIZING (+SN).
C
C     EVALUATE THE D(I) VECTOR - ALL D(I).GT.0.0 IS THE CONSTRAINT
C
      NM=NTOTER-N
      DO 202 II=1,NTOTER
      D(II)=DSTAR(II,1)
      DO 202 IQ=2,NM
  202 D(II)=D(II)+T(IQ)*DSTAR(II,IQ)
      SN=-1.0E+10
C
C     ASSIGN PENALTY FUNCTIONS TO SN IF ANY D(I).LE.0.0
C
      DO 203 II=1,NTOTER
      IF(D(II).LT.0.0)SN=SN+1.0E+20*D(II)
      IF(SN.LT.-1.0E+10)GOTO215
  203 CONTINUE
C
C     EVALUATE THE GAM(I) VECTOR.
C
      NTEMP1=1
      NTEMP2=NTERMS(1)
      DO 204 J=1,NCONS
      GAM(J)=0.0
      NTEMP1=NTEMP1+NTERMS(J)
      JJ=J+1
      NTEMP2=NTEMP2+NTERMS(JJ)
      DO 205 II=NTEMP1,NTEMP2
  205 GAM(J)=GAM(J)+D(II)
  204 CONTINUE
C
C     CALCULATE THE OPTIMIZATION FUNCTION SN.
C
      SN=CK(1)
      DO 206 IQ=2,NM
  206 SN=SN*CK(IQ)**T(IQ)
      DO 207 II=1,NTOTER
  207 IF(D(II).GT.0.0)SN=SN*D(II)**(-D(II))
      DO 208 J=1,NCONS
  208 IF(GAM(J).GT.0.0)SN=SN*GAM(J)**GAM(J)
```

```
C
C     MAKE SN NEGAT+VE AGAIN BFORE RETURNING TO SUBR.SEEK1
C
  215 SN=-SN
      RETURN
      END


      SUBROUTINE ADRANS (X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,UART,NCONS,NEQUS,
     1DSTAR,NTOTER,NTERMS)
      DIMENSION X(1),XSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
      DIMENSION DSTAR(NTOTER,1),NTERMS(1)
      COMMON INDEX,LEVEL,IPRINT,IDATA,F,MAXM,G,NSHRIN,MSTART,PD,EPS,ICT,
     1IFENCE,PL,NSTOP,NSMAX,NSHOT,NTEST,TES,Q,REDUCE,NVIOL,KO,NNDEX
      COMMON /A1/R(100),AVE(100),XO(100),RANGE(100)
C     AFTER EVERY F+VE IMPROVEMENTS THROUGH THE ADAPTIVE RANDOM SEARCH
C     MODE, A LARGE- STEP IS TAKEN ALONG A MEAN PATH THROUGH THESE
C     5 POINTS. MORE STEPS ARE TAKEN ALONG THIS PATH UNTIL A NEW POINT
C     FAILS TO PRODUCE AN IMPROVEMENT. THE PROGRAM THEN CONTINUES THE PATTERN
C     OF FINDING 5 NEW IMPROVEMENTS BY THE ADAPTIVE RANDOM SEARCH
C     FOLLOWED BY AN EXTRAPOLATION ALONG THE MEAN PATH .
      NNDEX=INDEX
      WRITE(6,43)
      NCOUNT=0
      KOUNT=1
      KON3=0
      K1=0
C     TO SPEED UP THE METHOD, USE SUBROUTINE FEASBL TO OBTAIN AN INITIAL
C     STARTING POINT. NOTE...THE METHOD DOES NOT ACTUALLY REQUIRE A
C     FEASIBLE START ,SO IF FEASBL FAILS THEN ADRANS STILL PROCEEDS.
C     SET F=.05 TO DEFINE THE INITIAL STEP SIZE IN FEASBL
C     SET G=.01 TO DEFINE THE MINIMUM STEP SIZE IN FEASBL
      F=0.05
      G=0.01
      CALL FEASBL(X,U,N,XSTRT,RMAX,RMIN,PHI,PSI,NCONS,NEQUS,UART,DSTAR,
     1NTERMS,NTOTER)
      IF(IPRINT.GT.0)WRITE(6,66)U,(X(I),I=1,N)
      IF(KO.EQ.1)WRITE(6,67)
C     IGNORE A KO=1 MESSAGE FROM FEASBL
      KO=0
C     ZERO THE COMMON BLOCK ARRAYS SINCE THEY ARE USED IN SUBR. FEASBL
      DO 4 I=1,100
      R(I)=0.0
      AVE(I)=0.0
      XO(I)=0.0
    4 RANGE(I)=0.0
      DO 5 I=1,N
      XO(I)=X(I)
    5 RANGE(I)=ABS(RMAX(I)-RMIN(I))
C     SUBROUTINE OPTIMF IS THE OPTIMISATION FUNCTION WITH PENALTIES
      CALL OPTIMF (X,UO,PHI,PSI,NCONS,NEQUS)
C     RANDOM NUMBER GENERATION
      K=MSTART
```

```
  8   DO 9 I=1,N
  9   AVE(I)=0.
      M=1
 11   CALL FRANDN(R,N,K)
      K=0
      DO 10 I=1,N
C     GENERATE NUMBERS WITHIN HALF THE RANGE FROM XO(I)
   10 X(I)=XO(I)+RANGE(I)*(R(I)-.50)**M
      CALL OPTIMF (X,U,PHI,PSI,NCONS,NEQUS)
      K1=K1+1
      IF(U.LT.UO)GOTO18
      IF(K1.LE.NSMAX)GOTO11
C     IF NO IMPROVEMENT AFTER NSMAX TRIES WITH THE MINIMUM RANGE (M=7)
C     THEN AN OPTIMUM IS ASSUMED
      IF(M.GE.7)GOTO45
C     INCREASE M TO EFFECTIVELY DECREASE THE STEP SIZE
      M=M+2
      K1=0
      GOTO11
   18 K1=0
      M=1
      DO 20 I=1,N
      AVE(I)=AVE(I)+(X(I)-XO(I))
   20 XO(I)=X(I)
      UO=U
      NCOUNT=NCOUNT+1
C     FIVE RANDOM NUMBERS ARE GENERATED
C     THE AVERAGE OF THE FIVE VALUES IS THEN OBTAINED
      IF(NCOUNT.LT.5) GO TO 11
      NCOUNT=0
      DO 25 I=1,N
 25   AVE(I)=AVE(I)/5.
C     PATTERN SEARCH
C     K2 - IS A COUNT OF THE CYCLES MADE WITHIN THE PATTERN SEARCH
      K2=0
 50   DO 30 I=1,N
 30   X(I)=XO(I)+AVE(I)
      CALL OPTIMF (X,U,PHI,PSI,NCONS,NEQUS)
      IF(U.GE.UO) GO TO 42
      DO 40 I=1,N
      AVE(I)=AVE(I)*1.2
 40   XO(I)=X(I)
      UO=U
      K2=K2+1
C     DO NOT MAKE MORE THAN 50 PATTERN MOVES WITHOUT RECALCULATING THE
C     BEST DIRECTION BY THE RANDOM SEARCH STRATEGY ABOVE
      IF(K2.GT.50) GO TO 42
      GOTO50
 41   KO=1
      WRITE(6,65)KOUNT
      GOTO100
 42   KON3=KON3+1
      DO 12 I=1,N
 12   X(I)=XO(I)
```

```
      IF(IPRINT.EQ.0.OR.KON3.NE.IPRINT)GOTO46
      IF(KOUNT.EQ.IPRINT)WRITE(6,48)
      CALL UREAL(X,UU)
      WRITE(6,44)KOUNT,UU,(X(I),I=1,N)
      KON3=0
   46 KOUNT=KOUNT+1
      IF(KOUNT.GT.MAXM)GOTO41
      GO TO 8
 45   DO 13 I=1,N
 13   X(I)=XO(I)
      KOUNT=KOUNT+1
      CALL UREAL(X,UU)
      IF(NCOUNT.GT.0)WRITE(6,44)KOUNT,UU,(X(I),I=1,N)
  100 CALL ANSWER(U,X,PHI,PSI,N,NCONS,NEQUS)
   43 FORMAT(1H1,49HOPTIMIZATION USING ADAPTIVE RANDOM SEARCH  ADRANS//)
   44 FORMAT(1HO,I4,3X,5E16.8/(24X,4E16.8))
   48 FORMAT(1H-,15X,1HU,25X,23HINDEPENDENT VARIABLES X//)
   65 FORMAT(1H-,20HNO CONVERGENCE AFTER,I5,7H  MOVES/)
   66 FORMAT(1H-,38HSTARTING POINT FOUND BY METHOD IS  U =,E16.8,11H  AT
     1 X(I) =,//(1X,E15.8,4E16.8))
   67 FORMAT(1H+,81X,12H(INFEASIBLE)/)
      RETURN
      END
```

REFERENCES

1. Garvin, W.W., "Introduction to Linear Programming", McGraw-Hill, 1960.

2. Siddall J.N., McDonald J.F., et al., "OPTIPAC - Volume I: Users' Information", Department of Mechanicl Engineering, McMaster University, October, 1969.

3. Siddall, J.N., McDonald J.F., et al., "OPTIPAC - Volume II: Programmers' Information", Department of Mechanical Engineering, McMaster University, October, 1969.

4. Duffin, R.J., Peterson E.L. and C.M. Zener, "Geometric Programming", John Wiley & Sons, 1967.

5. Griffith, R.E. and R.A. Stewart, "A Nonlinear Programming Technique For The Optimization of Continuous Processing Systems", Management Science, Vol. 7, 1961, pages 379-392.

6. Gurunathan, U., "Vibration Analysis and Design Optimization Studies of Space Frames-Optimization Studies", Master's Thesis, Department of Mechanical Engineering, McMaster University, May 1968.

7. Hooke, R. and T.A. Jeeves, "Direct Search Solution of Numerical and Statistical Problems", Westinghouse Research Laboratories Scientific Paper, 1960.

8. Flood, M.M. and A. Leon, "Direct Search Code For The Estimation of Parameters in Stochastic Learning Models", Mental Health Research Institute, University of Michigan, May 1963.

9. Flood, M.M. and A. Leon, "A Generalized Direct Search Code For Optimization", Mental Health Research Institute, University of Michigan, June 1964.

10. Fiacco, A.V. and G.P. McCormick, "Extensions of SUMT For Nonlinear Programming: Equality Constraints and Extrapolation", Management Science, Vol. 12, July 1966, pages 816-828.

11. Fiacco, A.V. and G.P. McCormick, "Nonlinear Programming: Sequential Unconstrained Minimization Techniques", Wiley, 1968.

12. Gall, D.A. and E. Krokosky, "A Generalized Procedure For Automated Optimal Design", Proc. Fourth National Conference on Engineering Design, Hanover, New Hampshire, July, 1967.

13. Gallagher, P.J., "MOP-1, An Optimizing Routine For The IBM 650", Canadian General Electric, Civilian Atomic Power Dept. Report No. R60CAP35, 1960.

14. McArthur, D.S., "Strategy in Research-Alternative Methods For Design of Experiments", IRE Trans. on Engrg. Management, Vol. EM-8, March 1961, pages 34-40.

15. Frank, C.J., "Transformer Design Using Geometric Programming", Westinghouse Research Laboratories Memo 64-7C4-361-M1, December 1964.

16. Beightler, C.S., Lo, T. and H.G. Rylander, "Optimal Design by Geometric Programming", ASME Paper No. 69-WA/DE-7, July, 1969.

17. Siddall, J.N., "Theory of Engineering Design - Part II", Department of Mechanical Engineering, McMaster University, 1967.

18. Hassitt, A., "Computer Programming and Computer Systems", Academic Press, New York, 1967.