MATHEMATICAL PROGRAMMING AS AN AID TO ENGINEERING DESIGN

A user's guide to available MP computer codes and to NELC's capabilities in numerically solving MP problems

D. C. McCall
Research and Development

11 August 1971
This report is a user's guide to available mathematical programming (MP) computer codes and to NELC's capabilities in numerically solving MP problems. It defines the general MP problem; lists and evaluates the MP codes operational on the NELC IBM 360/65 computers; and provides guidelines for modifying the MP problem when, in its first form, it is cumbersome; or there is not information enough to start computation; or the available codes do not yield all the needed information.
<table>
<thead>
<tr>
<th>KEY WORDS</th>
<th>LINK A</th>
<th>LINK B</th>
<th>LINK C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLE</td>
<td>WT</td>
<td>ROLE</td>
<td>WT</td>
</tr>
<tr>
<td>ROLE</td>
<td>WT</td>
<td>ROLE</td>
<td>WT</td>
</tr>
</tbody>
</table>

Computer programming

Mathematical programming
PROBLEM

Provide analysis and synthesis of Navy design problems. Specifically, develop a capability at NELC for using mathematical programming (MP) as an aid to engineering design.

RESULTS

1. The overall software capabilities at NELC for numerically solving various types of MP problems — initiated or developed under this problem — are discussed in the report proper. Applications of integer programming are given in Appendix 1. User's guides and FORTRAN codes for solving some classes of MP problems are given in Appendix 2.

2. Practical guidelines are given for applying MP methods.

RECOMMENDATIONS

1. Maintain a continued effort to keep the mathematical programming software current. Monitor research literature for new developments in non-linear programming and integer programming.

2. Conduct ongoing seminars or in-house classes to inform practicing scientists and engineers of the utility of MP.

3. Review Navy engineering design problems for possible application of MP techniques.

ADMINISTRATIVE INFORMATION

Work was performed under ZFXX.212.001 (NELC Z223) by the Decision and Control Technology Division. The report covers work done from July 1968 to March 1970 and was approved for publication 11 August 1971.

The author thanks W. J. Dejka, Advanced Modular Concepts Division, who, as the initial principal investigator of Z223, set the goals and general trends of the project, for assistance and comments on this report; and San Diego State College Foundation students, Gail Grotke and Dale Klamer, and C. M. Becker, Applications Software Division, for programming assistance in preparing Appendix 2.
CONTENTS

INTRODUCTION: SCOPE OF REPORT ... page 3
THE MATHEMATICAL PROGRAMMING PROBLEM ... 4
MATHEMATICAL PROGRAMMING CAPABILITIES AT NELC ... 5
  Unconstrained problems ... 5
  Constrained problems ... 8
DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID ... 19
  Formulating a mathematical programming problem ... 19
  SUMT and constraint transformation ... 22
  Gradient approximation ... 27
  Duality in MP ... 33
  Lagrange multipliers as a design aid ... 43
  Initial point and scaling ... 50
SUMMARY ... 52
APPENDIX 1: APPLICATIONS OF INTEGER PROGRAMMING TO
  ENGINEERING DESIGN ... 53
APPENDIX 2: USER INFORMATION ... 59
APPENDIX 3: REFERENCES ... 123

ILLUSTRATIONS

1 Constraint set ... page 5
2 Examples of convex and nonconvex sets ... 12
3 Disconnected constraint set ... 15
4 Magnitudes of currently solvable MP problems ... 18
5 Dual circuits ... 33
6 Kuhn-Tucker conditions ... 45
7 Integer constraint set ... 54
8 Backplane grid ... 56

TABLE

1 Listing of FORTRAN code of test problem ... 29
INTRODUCTION: SCOPE OF REPORT

There are several problems involved in the development of faithful mathematical models of real-world processes. The processes are in general nonlinear. The modeling equations are frequently incomplete. Conditions are known only within limits. Often the best approach to these problems is via mathematical programming (MP).\(^1\)\(^-\)\(^5\)

MP is a distinct discipline — it exists independently of computer programming. Prior to the age of the high-speed computer, however, some of the original algorithms for the solution of MP problems were too cumbersome to be of real use. The advent of the modern digital computer has made the solution of many types of MP problems feasible and has stimulated the search for better algorithms.

This report is chiefly concerned with solving MP problems — with the computational stage of MP. It is intended as a guide for the usage and application of available MP computer codes.

THE MATHEMATICAL PROGRAMMING PROBLEM defines the general MP problem.

MATHEMATICAL PROGRAMMING CAPABILITIES AT NELC lists and evaluates the MP codes operational on the NELC IBM 360/65 computer, and will be of interest to the user who has an MP problem in final form, ready to solve. He can choose the appropriate code from the list and obtain the card deck and user's guide from the NELC program library, Computer Sciences Department.

DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID provides guidelines for modifying the MP problem when, in its first form, it is cumbersome; or when there is not information enough to start computation (for example, an initial feasible point is lacking); or when the available codes do not yield all the needed information (for example, postoptimal analysis).

\(^1\) See APPENDIX 3: REFERENCES.
THE MATHEMATICAL PROGRAMMING PROBLEM

The basic problem of MP is to develop an algorithm for finding the minimum of a scalar-valued function of n real variables that satisfies a set of auxiliary conditions called constraints. Stated in mathematical terms, the problem becomes:

Let \( f(x_1, \ldots, x_n) \) and \( g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n) \) be scalar-valued functions of the n variables \( x_1, \ldots, x_n \). Then we wish to find variables which minimize \( f(x_1, x_2, \ldots, x_n) \)

subject to

\[
\begin{align*}
g_1(x_1, x_2, \ldots, x_n) & \geq 0 \\
\vdots & \\
g_m(x_1, x_2, \ldots, x_n) & \geq 0
\end{align*}
\]

The above problem is known variously as the 'general mathematical programming problem,' the 'constrained optimization problem,' and the 'nonlinear programming problem.' For the sake of convenience, we call it problem (1).

In problem (1), \( f \) is called the objective or cost function and the \( g_i \) are called the constraints. We also refer to \( f \) and \( g_i \) as the problem functions.

We denote the vector \((x_1, x_2, \ldots, x_n)\) by \( x^T \) (where \( T \) denotes the matrix transpose) and call the set of all vectors \( x \) which satisfy \( g_i(x) \geq 0 \) for all \( i = 1, \ldots, m \), the constraint set or feasible set. The problem is said to be consistently posed if the constraint set is nonempty. We note that finding the maximum of a function \( f \) is equivalent to finding the minimum of \(-f\).

Consider the following example (fig. 1):

Minimize \( f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 \)

subject to

\[
\begin{align*}
g_1(x_1, x_2) &= x_2 - x_1^2 \geq 0 \\
g_2(x_1, x_2) &= -x_1 - x_2 + 2 \geq 0
\end{align*}
\]

Since \( f(x_1, x_2) \) is the sum of squares, the minimum occurs at \( x_1 = 2, x_2 = 1 \). However, the point \((2,1)\) is not in the constraint set defined by \( g_1 \) and \( g_2 \).

The obvious (in this example) candidate is the point \((1,1)\), which in this straightforward problem is the constrained minimum. As \( m \) and \( n \) get larger, the problem becomes significantly more difficult to solve.
We now turn to a discussion of computer codes which can solve MP problems, for different classes of problem functions.

**MATHEMATICAL PROGRAMMING CAPABILITIES AT NELC**

**UNCONSTRAINED PROBLEMS**

The unconstrained MP problem is stated as:

\[
\text{Minimize } f(x_1, x_2, \ldots, x_n)
\]

where \( f \) is a scalar-valued function of the \( n \) variables \( (x_1, x_2, \ldots, x_n)^T = x \).

Gradient methods and direct search techniques are the two basic approaches to numerically solving the unconstrained problem. The gradient of \( f \) at \( x \), denoted by \( \nabla f(x) \), is defined to be the following vector

\[
\nabla f(x) = (\partial f(x)/\partial x_1, \ldots, \partial f(x)/\partial x_n)^T
\]

Gradient methods use, in some way, the following facts:

1. At the minimum \( x^* \) of \( f \), we have \( \nabla f(x^*) = 0 \).
2. If \( \nabla f(x) \neq 0 \), then \( -\nabla f(x) \) points in the direction of steepest descent.

\( \)A finer classification would be direct methods, gradient methods, and those methods involving the matrix of second derivatives. We feel that the first two are the most useful for applications.
This analytic information makes the gradient methods fast and enables the computer codes to compute meaningful error information. To use gradient methods, we must have the gradient of f available analytically or have a numerical way to compute it. Direct search methods eliminate this need for the gradient and rely only on the behavior of the objective function in seeking out the minimum. Typically, direct methods evaluate the cost function many more times than gradient methods in minimizing the same test function. In minimizing the Rosenbrock test function (see Appendix 2), the direct search routine ZANGWL requires 325 function evaluations, while the gradient method CONJGT requires only 71 combined function and gradient evaluations, in finding the optimum to within the same accuracy. This is a trade-off a user must make if he can choose between a gradient method and a direct method.

Direct methods do not rest on so firm a mathematical foundation as gradient methods do, and most direct methods are proved to converge for only special functions. However, they have been useful in practice, since the objective function in many applications is complicated or its gradient is not available. It is generally simpler to code a problem for a direct method, which allows for faster implementation on the computer.

We present the following example of posing a two-point boundary value problem (TPBVP) as an unconstrained MP problem, to illustrate both an application of MP and the need for good direct search methods. The problem is to find an n-dimensional vector function \( y(t) \) which satisfies

\[
\dot{y} = h(t, y) \quad (a \leq t \leq b)
\]

with

\[
y_i(a) = q_{ia} \quad (i=1,2,\ldots,j<n)
\]

and

\[
y_i(b) = q_{ib} \quad (i=j+1,\ldots,n)
\]

In general, this problem has no closed-form solution, and in some cases no solution at all. However, since it frequently arises in applications, either a numerical estimate of the solution is desired or, if no solution exists, a function which comes close (in some sense) to solving the problem is desired. With this in mind we pose the following MP problem.\(^{\dagger}\)

\[
\text{Minimize } f(x_1, x_2, \ldots, x_{n-j}) = \left( \sum_{i=j+1}^{n} (y_i(b) - q_{ib})^2 \right)^{\frac{1}{2}} \]  

\(^{\dagger}\)Rosen\(^{6}\) discusses the same problem and obtains approximate solutions using linear programming techniques. His approach requires a great deal of equation manipulation before the linear programming techniques can be applied. Unfortunately, no comparison between the two methods has been made.
where the numbers \( y_i(b), i=j+1, \ldots, n \) are numerically computed as follows. For a given \((x_1, \ldots, x_{n-j})\), solve the following initial-value problem over the interval \([a,b]\).

\[
\dot{y} = h(t,y) \quad (a \leq t \leq b)
\]

where

\[
y(a) = (q_{1a}, \ldots, q_{ja}, x_1, \ldots, x_{n-j})
\]

the last \(n-j\) components of the solution obtained at \(t=b\) of this problem are used in the objective function for \(y_i(b), i=j+1, \ldots, n\).

In this problem there is no analytic expression for the objective function \(f\), from which \(\nabla f\) can be derived. Thus, to numerically solve this unconstrained problem (3), either a direct search method must be used or \(\nabla f\) must be numerically calculated via a differencing routine. We recommend the former as reliable and easy to use, and discuss some of the drawbacks of the latter in *DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID*, Gradient approximation. Dejka\(^7\) discusses a similar TpBVP and uses a direct search method to solve a related MP problem.

We briefly discuss computer routines available from the NELC program library for solving the unconstrained problem. The user's guides for these routines provide ample background information and details for implementation.

Gradient methods from the library are FP, CONJGT, SOREN, FMFP, and FMCG. FP and CNJGAT originally were programmed and used by Winterbauer\(^8\) to solve a parameter selection problem for a sonar signal equation, but they are general-purpose, unconstrained, MP codes. FP and CNJGAT are based on the methods of Fletcher and Powell\(^9\) and Fletcher and Reeves,\(^10\) respectively. SOREN is a modification of CNJGAT which has converged faster for some test functions. FMFP and FMCG are available from the IBM Scientific Subroutine Package.\(^11\) we have not tested these last two routines extensively or compared them with the other gradient methods.

ZANGWL, DIRECT, and UNIVAR are direct search codes in the library. ZANGWL is discussed in Appendix 2, and DIRECT and UNIVAR are presented in reference 12. These three routines are based on methods presented in references 13-15, respectively, and were programmed at NELC. ZANGWL has a mathematical basis for convergence similar to that of CNJGAT, and of the three direct methods, it is the most efficient in terms of the total number of function evaluations required to minimize a function. In minimizing the Rosenbrock function, to the same accuracy and from the same initial point, the number of function evaluations were ZANGWL(325), DIRFCT(705), and UNIVAR(2303). The extraordinary number of function evaluations clearly makes UNIVAR unacceptable, but ZANGWL should not be selected over DIRECT. DIRECT makes intermediate searching moves in a much more
cautious manner than ZANGWL, which makes it better for some applications. This is discussed in DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID. SUMT and Constraint Transformation.

Before moving onto the constrained problem, we give a word of caution on all minimization routines. Each method, be it a constrained or an unconstrained code, is capable of finding only a local minimum and not a global minimum. We define local and global minima in the next lines. Let the objective function \( f(x) \) be defined on a set \( G \) in an \( n \)-dimensional vector space, denoted by \( \mathbb{E}^n \). Then we say that \( f \) has a global minimum at \( x^* \) (in \( G \)) if \( f(x^*) \leq f(x) \) for all \( x \) in \( G \). Note that we do not exclude \( G \equiv \mathbb{E}^n \). A \( \hat{x} \) in \( G \) is called a local minimum, if for all \( x \) sufficiently close (with respect to some norm) to \( \hat{x} \), and also in \( G \), we have \( f(\hat{x}) \leq f(x) \).

Local minima can occur in the gradient methods because the condition that \( \nabla f(x^*) = 0 \) is only necessary and not sufficient for a global minimum. In direct methods, only local information about the surface defined by the objective function is available to the routine. This characteristic makes direct methods susceptible to stopping at a local minimum. For reasonable certainty that a global optimum has been reached, it is wise to restart the problem from different initial points. In many applications, if a local minimum gives a satisfactory value of the objective function, no further processing is necessary.

Work is continuing in the area of unconstrained minimization algorithms, with refinements to the above methods and new methods appearing regularly in the literature. The most fruitful and accessible sources of articles on the subject are The Computer Journal, Communications of the Association of Computing Machinery, SIAM Review, and the SIAM journals on control, numerical analysis, and applied mathematics. The above sources, together with Management Science and Operations Research, contain many articles on the constrained problem.

CONstrained PROBLEMS

We return to the discussion of problem (1) for various classes of problem functions. The following types of mathematical programming problems are discussed: linear, quadratic, convex, nonlinear and nonconvex, and integer. The methods for solving these problems make explicit use of the properties of the problem functions.

When an unconstrained problem is solved, the codes require only an initial point for which the objective function is defined. This requirement is more demanding in the constrained problem. Depending on the type of problem under consideration, the user can be required to provide an initial feasible point, as a starting point for the computation. In many applications an initial feasible point is known from the engineering knowledge of the problem. However, if the constraints are numerous or complicated, such a point will not be
obvious and a preliminary step must be taken prior to solving the problem.
A method for obtaining an initial feasible point is treated in DESIGNING WITH
MATHEMATICAL PROGRAMMING AS AN AID, Initial Points and Scaling,
so we assume that one is at hand in the following discussion.

Each class of constrained MP problem is described, together with
computer programs which can solve it. Since each routine to be discussed has
an associated user’s guide, we confine our remarks to the following points:

1. Is an initial feasible point required?
2. Does the code find a global optimum?
3. Can information be saved for possible restarts or postoptimal
   analysis?
4. Is the routine easy to use?
5. What error messages are given if the routine fails to converge?

LINEAR PROGRAMMING (LP)

In the standard linear programming problem all the functions in
question are linear and the problem variables are constrained to be non-
negative. We write:

Minimize \( f(x) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \)
subject to

\[
\begin{align*}
g_1(x) &= a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1 \\
&\vdots \\
g_m(x) &= a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m
\end{align*}
\]

This LP problem can be written in matrix notation as:

Minimize \( z = c^T x \)
subject to

\[
\begin{align*}
Ax &\leq b \\
x_i &\geq 0
\end{align*}
\]

where the \( \leq \) means that the corresponding components of the vectors are
"less than or equal to." If the constraints are consistent, then the simplex
method of linear programming guarantees that a global optimum\(^\dagger\) can be
found in a finite number of steps. The simplex method is an iterative proce-
dure and generates "basic feasible solutions" at each iteration, which decrease
\( z \). To produce these solutions, a "basis inverse" matrix is calculated. The

\(^\dagger\)An unbounded solution can also be detected in a finite number of steps.
preceding brief comments serve only to associate the terms “basic feasible solution” and “basis inverse” with the simplex method; references 16 and 17 treat the simplex method.

The most complete code for numerically solving the LP problem is the IBM Mathematical Programming System\(^{18}\) (MPS/360). MPS/360 is based on a modification of the simplex method and will either solve the LP problem or indicate that no solution exists. This code does not require that the problem variables be nonnegative, and treats upper and lower bounds on the variables as special constraints. Separable programming problems (a special nonlinear MP problem) can be solved with this routine. MPS/360 is capable of solving problems of up to 4095 constraints and “virtually an unlimited number of columns.”\(^{18}\) It is currently stored on disk pack NELC05 at the NELC Computer Center.

No initial point is required to begin the computation; however, the option exists to start the problem from a user-supplied basis inverse. MPS/360 has its own control language, which provides a variety of capabilities. The user is afforded several postoptimal analysis procedures and can access the current basis inverse for future restarts. This control language is straightforward to use and provides some looping and branching capability. A variety of messages are output to the user in the course of computation. They are fully explained in the message manual.\(^{19}\)

The chief drawback of this program is the format of the input data. It requires each element of the arrays \(c,b,\) and \(A\) to have a “row name” and a “column name” for identification. This has proved cumbersome for scientific and engineering work. A FORTRAN program, DATAPREP, is available to reduce the data arranged in compact matrix notation to a format acceptable to MPS/360.

In many applications a linear programming problem must be solved repeatedly as part of a larger problem. The READCOMM\(^{20}\) facility of MPS/360 allows the main program to be used in an iterative fashion as a subroutine. READCOMM enables the user to supplement the standard control language with FORTRAN procedures; for example, DATA PREP. Rosen\(^{6}\) and Griffith and Stewart\(^{21}\) have examples of using a linear programming code in an iterative way.

Previous large-scale, efficient LP codes were geared to commercial applications and required a great deal of modification for efficient scientific and engineering use. The READCOMM facility has made a powerful program easily available for a wide range of specialized applications.

**QUADRATIC PROGRAMMING (QP)**

This type of problem is the next order of difficulty. A quadratic cost function is minimized subject to linear constraints:
Minimize \( f(x_1, x_2, \ldots, x_n) \)
subject to
\[
a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b_1
\]
\[
a_mx_1 + a_mx_2 + \cdots + a_mx_n \leq b_m
\]
where \( f \) has one of two forms –
\[
f(x_1, x_2, \ldots, x_n) = c^T x + x^T B x
\]
or
\[
f(x_1, x_2, \ldots, x_n) = H x - e
\]
B and \( H \) are \( n \)-by-\( n \) and \( k \)-by-\( n \) matrices, respectively, and \( c \) and \( e \) are \( n \)-, and \( k \)-dimensional vectors, respectively. The norm of a vector \( y \), denoted by \( \|y\| \), is given by
\[
\|y\| = \left( \sum_{i=1}^{n} y_i^2 \right)^{1/2}
\]
At present only the minimum norm problem (equation 6) can be solved at NLCLC. The program which does this is QPHANSON. This routine was written by R. J. Hanson, of the Jet Propulsion Laboratory, Pasadena, and uses a numerically stable\(^\dagger\) version of Rosen's\(^{22}\) gradient projection algorithm.
The method guarantees that a global optimum will be found for a consistent problem. The routine is reported to have worked well on examples from the areas of curve fitting and approximation of solutions to linear integral equations.\(^{23}\)
The routine is described in reference 23, and an NELC user's guide is in preparation. The code is operational, but it is still unfinished in respect to user-oriented input and output. QPHANSON does not require an initial feasible point nor does it have an option to accept a good approximation of the optimum. The code treats equality constraints and inequality constraints separately. It can solve problems of up to 60 constraints (equality and inequality combined) and 30 variables. No experiments have been done to determine whether this is a hard and fast upper bound on the problem size.
The program suffers from lack of good error and timing messages. If the routine fails to converge, no messages are given as to the possible cause. Also, no provisions are made to identify inconsistent quadratic programs. The user is on his own with QPHANSON.

\(^\dagger\)An algorithm is numerically stable if the errors in the input data are approximately equal to the round-off errors generated by the computations of \( \phi \).
If a QP problem occurs with equation (5) as the cost function, and the matrix $B$ is positive-definite or positive-semidefinite, then Hanson presents a method for transforming this QP problem into a minimum-norm QP problem. This minimum-norm problem is solved with QPHANSON and then an inverse transformation is made. Presently this must be done by the user. QPHANSON is currently being modified to make this transformation automatically.

Codes to solve quadratic programs are not as well polished or as highly developed as those for LP. Unless the demand increases for good QP routines, the user will have to write his own code or be content with the experimental models.

**CONVEX PROGRAMMING**

Before turning to the convex programming problem, we make some preliminary definitions.

**CONVEX SET.** A set $G$ in $\mathbb{R}^n$ is said to be convex (fig. 2) if for any two points $x_1$ and $x_2$ contained in $G$ we have $\lambda x_1 + (1-\lambda)x_2$ contained in $G$, for all $\lambda$ in $(0,1)$.

---

A symmetric matrix $B$ is positive-definite (semidefinite) if for every $x \neq 0$ we have $x^T B x > 0$, ($\geq 0$).
CONVEX FUNCTION. A scalar-valued function $f$ defined on a convex set $G$ in $E^n$ is said to be convex if for any two points $x_1$ and $x_2$ in $G$

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$$

for all $\lambda$ in $(0,1)$.

Linear functions, and the quadratic cost function (equation 6) with $B$ positive-semidefinite, are examples of convex functions. A theorem of interest states that if the constraint set of an MP problem is defined by convex functions, then it, too, is convex. More precisely, if $g_1(x), \ldots, g_m(x)$ are convex functions, then the set of all $x$ for which $g_1(x) \leq 0, \ldots, g_m(x) \leq 0$ holds simultaneously is a convex set. The constraint sets of linear and quadratic programs are convex.

With these facts in hand we state the convex programming problem.

Minimize $f(x)$

subject to

$$g_i(x) \leq 0 \quad (i=1, \ldots, m)$$

where the functions $f$ and $g_i$ are convex functions of $x = (x_1, \ldots, x_n)^T$.

A great deal of work has been done with convex programming and the theory can guarantee convergence for some computational methods. Each method is valid for specific requirements on the problem functions. In this section we assume that the gradients of all functions exist and are continuous. The central problem in solving convex programs is not so much theoretical difficulty but rather the obtaining of rapid convergence of numerical schemes. Even though some QP problems are convex, it may be more efficient to use a routine like QPHANSO to solve them rather than treat them as convex programs. Another practical difficulty with convex programming is the identification of convex functions. If the function has a complicated analytic expression, it can be difficult to classify it as convex. The methods for solving convex programs will not in general completely hang-up if the data are not convex, but the significance of such results should be judged in terms of the user’s problem formulation.

The first library routine which solves the convex programming problem is the subroutine CONVEX, which was developed by Hartley and Hocking at Texas A&M. The routine makes a linear approximation to the functions in question and then uses a simplex-like procedure to move to the optimum. In making the linear approximations, the routine requires a user-supplied subroutine which computes the gradients of the cost function and the nonlinear constraints.

CONVEX does not require an initial feasible point; however, the option does exist to start from a given point. In addition, CONVEX produces
a current feasible point and a basis inverse at the end of each iteration for possible restarts. The format of the input data is straightforward and suitable to scientific and engineering work; however, care should be exercised in the organization of the data for any upper and lower bounds on the problem variables. CONVEX requires that the constraint data be input in three groups – the upper and lower bounds on the variables, the linear inequalities, and the nonlinear convex constraints. This feature makes it possible to conveniently solve quadratic programs with convex cost functions. No comparisons between QPHANSON and CONVEX have been made on solving quadratic programming problems.

CONVEX suffers from the lack of good error messages and analysis in the event of an inconsistent problem or any numerical difficulties. No investigation of the numerical stability of the method or of timing or accuracy benchmarks for large problems has been reported. A convex problem with 60 constraints and 60 variables is the largest which can be solved without program modifications. Because the linear constraints are treated separately, it is likely that larger problems can be solved if the number of nonlinear constraints is not too great. Future work should investigate this possibility.

The second routine for solving the convex programming problem is Experimental SUMT. This method is theoretically convergent for convex data, but, since it also has provisions for nonconvex programs, we postpone discussion of it until the next section.

There is a special subclass of convex programs for which a global optimum can be found with the linear programming code MPS/360. These are separable programming problems, which are defined as follows:

\[
\text{Minimize } z = \sum_{j=1}^{n} f_j(x_j) \\
\text{subject to } \sum_{j=1}^{n} g_{ij}(x_j) \leq b_i \quad (i=1, \ldots, m)
\]

Note that the objective function and the constraints are sums of functions of the single variables \(x_j\); that is, there are no "cross product" terms. This allows each nonlinear function to be replaced by a polygonal approximation, and reduces problem (7) to a form which can be solved by MPS/360. The MPS/360 user's manual gives the appropriate details and examples of solving separable programming problems. If the separable functions are convex, then MPS/360 will find a global optimum to the approximation problem. The use of successive approximations causes the global optima of the approximation problems to converge to the optimum of the original
problem. The method can tolerate some nonconvex functions but may stop at a local optimum.

This technique can also be used for separable convex programs too large for CONVEX. (CONVEX may be more efficient if the problem is not too large — unfortunately, no experimental evidence is available to aid in making the selection.)

NONLINEAR, NONCONVEX PROGRAMMING

This last class of MP problems is composed of all the problems which are not necessarily linear or necessarily convex. In the statement of problem (1), no requirements were made on the functions in question other than the assumptions that the objective function would be defined for all feasible \( x \) and the \( g_i \) would be defined for all \( x \). This statement of problem (1) is much too general to be of use. To have any hope of obtaining a solution, we must put some restrictions on the problem functions. The three computer codes which we discuss require that the gradients of all the functions in problem (1) exist and be continuous. Although these conditions are stringent from a mathematical point of view, \(^1\) they do not provide a base for an MP algorithm.

The following example illustrates one of the difficulties of nonlinear, nonconvex programming. Since the problem functions are possibly nonconvex, complicated constraint sets can be generated (fig. 3).

\[ g_1(x) = 0 \]
\[ g_2(x) = 0 \]

Figure 3. Disconnected constraint set.

In this example, if the initial feasible point \( x^0 \) is in one component of the constraint set and the optimum \( x^* \) in another component, then the routine cannot move from \( x^0 \) to \( x^* \) while keeping intermediate points feasible. Thus, the most that general MP computer routines guarantee is a local minimum.

One of the most sophisticated routines at NELC is the Ricochet Gradient\(^26\) method. This is an IBM SHARE routine which requires that the

\(^1\) If the constraint set is bounded, then these conditions are sufficient for a global optimum to problem (1) to exist. This existence theorem gives no method for finding the optimum, which can be difficult in the general problem.
gradients of both the objective function and the constraints exist and be continuous. The method requires an initial feasible point and begins by moving down the gradient of the objective function until a constraint is reached. The program "ricochets" and traverses on the objective function surface across the feasibility region to the opposite constraint. A triangle is then constructed with this traverse line as its base and its apex in the direction of the gradient of $f$. The next step is made along the line from the base to the apex. The method terminates either on a small step size or when no ricochet is possible.

The user must supply codes to calculate the cost function and the constraints and their gradients. An initial feasible point must also be provided. The program has no options for restarts or postoptimal analysis; in such cases the problem is simply rerun — the known best point is used as initial data.

This routine is capable of producing a tremendous amount of output information. Once the method for controlling this output has been mastered, the user has access to a variety of information, which can be of great use in solving a nonlinear problem. The accompanying user's guide provides detailed documentation on the code and the underlying method. A supplemental user's guide (Appendix 2) reports the results of some test examples and gives a sample deck setup for output control. This program has proved reliable and, after a bit of experience, easy to use.

A second computer code for the general problem is NELC FESDIR. This program is not as sophisticated as the Ricochet Gradient routine, but it can easily be modified for special applications. The user's guide, complete with results on test examples, appears in Appendix 2.

The final code available for solving the nonlinear problem is Experimental SUMT, which was written at Research Analysis Corporation by G. P. McCormick, et al. SUMT is not a production code and is primarily used as a research tool in MP. The experimental nature does not lessen its accuracy, but only its efficiency and speed. The code has modular structure, which allows for easy user modification and adaptation. The user's guide is complete with test examples, although some handwritten corrections and deletions are not too clear. The user has the option of providing an initial point himself or allowing the program to find one. The option also exists of having SUMT compute the gradients by a differencing scheme; the code can also check user-supplied gradients for errors. SUMT provides timing information and allows for user-controlled output. However, the output can be confusing, with the values of different variables appearing under the same headings. The only error message other than incorrectly entered data is a warning during computation that certain estimates indicated the problem functions to be not convex. The theoretical background for SUMT is described in Fiacco and McCormick.

In DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID, SUMT and Constraint Transformation, we discuss methods for transforming
a constrained problem into a sequence of unconstrained problems (also called SUMT). This requires more work on the part of the user but can also give him more control in solving the problem and perhaps more insight as to what is happening. In solving the associated unconstrained problems, the user has the option of selecting a direct search method, thereby eliminating the need for differentiable or, in some problems, continuous functions. Special-purpose MP routines can be closely tailored to fit special applications via these techniques.

**INTEGER PROGRAMMING**

In addition to the standard linear programming problem, there are several special programs under the heading of integer linear programming (ILP). The problem statement is similar except for constraints placed on the variables:

Minimize \( f(x) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \)

subject to

\[
\begin{align*}
g_1(x) &= a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1 \\
g_m(x) &= a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m \\
x_i \geq 0 \text{ and } x_i \text{ is an integer.}
\end{align*}
\]

There are further distinctions within ILP—pure integer, mixed integer, and \([0,1]\). The above problem is a pure integer problem; if \(x_1, \ldots, x_k\) are required to be integral and \(x_{k+1}, \ldots, x_n\) are not necessarily integral, then we have a mixed integer problem; and finally if \(x_i\) can equal only zero or one, we have a \([0,1]\) ILP. Integer programming is in its infancy, and some methods, although they theoretically exhibit finite convergence, have not been computationally successful. The three ILP routines available at NELC are “Zero-One Integer Programming with Heuristics,” \(^{28}\) BBMIP, \(^{29}\) and OPTALG. Zero-one and BBMIP are SHARE routines which have been checked out on the 360/65 but not tested extensively. BBMIP (a mixed integer routine) has been tried on a series of test problems\(^{30}\) and compared with other routines; however, the other codes are machine-dependent, so the comparison is not meaningful.

OPTALG is a bound and scan pure integer programming code which has solved large problems successfully. It was developed at Stanford by F. S. Hillier, \(^{31}\) and is currently operational at NELC. The routine requires 336k of core and a solution to the associated linear programming problem; that is, we simply drop the restriction that \(x_i\) be an integer. This solution, together
with the LP basis matrix and a guess at an initial feasible solution are then used as data for OPTALG. If the problem is large (a maximum of 61 rows and 61 columns), then this data preparation can be tedious if done by hand. The procedure has been somewhat automated; the exact details are in the user’s guide (Appendix 2).

The user should be cautious when attempting to solve an ILP with an integer programming code, since it is possible for a routine to solve some problems and not others, even if they are the same dimension and fairly similar. Matching the routine to the problem is still an art. Progress is being made in this area, but it will be some time before methods are available to solve a general ILP. A selection of engineering applications is presented in Appendix 1.

The following graph (fig. 4) gives an idea of the sizes of mathematical programming problems that can currently be solved. The abscissa represents the sum of both the number of constraints and the number of variables; integer programming methods are still too inconsistent to include.

---

Figure 4. Magnitudes of currently solvable MP problems.
DESIGNING WITH MATHEMATICAL PROGRAMMING AS AN AID

The engineer can effect a better design in some areas in less time and at lower cost with mathematical programming techniques than with classical methods. The key to success is in the word "aid," for, in using the methods effectively, the engineer must have command of his own field and understand some basic principles of MP techniques. It is recognized that typically the user of MP techniques is concerned with obtaining results quickly without lengthy excursions into numerical analysis or mathematical programming. Commercial routines – ECAP, MATCH, etc. – are geared to such a user; however, blindly accepting the output of such routines can be disastrous. Also, these off-the-shelf routines are procrustean – they generally do not lend themselves to modification for a special case. General mathematical programming computer codes can be modified for particular requirements. For example, if filter design by MP techniques is a frequent task, then the computer code can be modified to incorporate the automatic scaling of variables.

In these remaining sections, we discuss how an engineer would proceed from start to finish in using MP as a design aid. We also discuss some special topics, such as duality, that do not apply in all instances, but, if used properly, can save time and money, both in setup analysis and in obtaining a numerical solution.

FORMULATING A MATHEMATICAL PROGRAMMING PROBLEM

In many design problem statements there is, instead of a single goal, a collection of specifications to be satisfied. This gives the engineer several degrees of freedom in formulating an associated MP problem. He has the option of merely satisfying all the design specifications (this is equivalent to finding an initial feasible point) or of singling out one distinguished requirement and using it as the objective function. For example, in reference 1, a tunable bandpass filter was to be designed which would satisfy the following (among other) specifications:

1. At the tuned frequency F, the "insertion loss" of the filter was to be less than 2 dB.
2. At 10% on either side of F, the "roll off" was to be at least 40 dB.

Either of the above options for formulating an MP problem would have been satisfactory. The latter method was chosen. The insertion loss of the filter was selected to be minimized and the roll off requirements were treated as constraints.

To properly formulate an MP problem, the user must explicitly determine the following:
1. The objective (performance, tolerance, etc.) that is to be accomplished.

2. The mathematical relations that govern the interaction between the independent design variables.

3. The bounds and limitations on the values of the components that guarantee a realizable design.

With this information in hand the designer can select which MP approach to use. However, since care is required in choosing the objective function and providing a code for its numerical evaluation, we present some general examples and guidelines.

The objective function can be defined as a measure of the merit or the desirability of a solution to a problem, and its magnitude typically represents cost, profit, performance, quality, etc., or a combination of these. The case of the single well defined objective function generally poses no problems; it is the combination of goals which can lead to difficulties. The following examples illustrate various treatments of multiple goals.

Suppose that it is desired to minimize both the insertion loss of a network denoted by \( f_1(x) \) and the cost of the components denoted by \( f_2(x) \). Then one formulation of an objective function \( f \) would be

\[
\text{Minimize } f(x) = f_1(x) + r f_2(x) \quad (9)
\]

where \( r \) is an appropriately chosen scaling factor. Another possible choice of \( f \) would be

\[
\text{Minimize } f(x) = -f_2(x)/f_1(x) \quad (10)
\]

Note that no scaling factor is required and the dimensions of the objective function are

\[
\text{cost($)/(unit power loss) = cost($)/(unit power gain)} \quad (11)
\]

The next example illustrates treating secondary design goals as constraints. Suppose that high reliability \( f_3(x) \) is desired but the primary goal is a design of minimum costs \( f_2(x) \). A constrained formulation would be

\[
\text{Minimize } f_2(x) \quad (12)
\]

subject to

\[-f_3(x) + \alpha \leq 0\]

where \( \alpha \) is a tolerance on reliability (mean time to failure).

A possible numerical pitfall is combining design goals in a haphazard manner. Suppose we have two performance indicators \( u_1(x) \) and \( u_2(x) \), and we desire to maximize \( u_1 \) and minimize \( u_2 \) in the same design. Since maximizing \( u_1 \) is equivalent to minimizing \(-u_1\), a possible objective function
would be

\[ \text{Minimize } f(x) = w_2^* u_2(x) - w_1^* u_1(x) \]  \hspace{1cm} (13) 

where the constants \( w_1 \) and \( w_2 \) are required to make the function dimensionally correct. These weights are important, since the magnitudes of \(-u_1\) and \(u_2\) at the optimum \( x^* \) can be quite different. For example, if \(-u_1(x^*) \approx 0\) and \(u_2(x^*) \approx 1000\), then the effect of \(u_1\) is obliterated by \(u_2\).

The above discussion of objective functions is intended to be general. Aoki\(^{36}\) presents many detailed examples of engineering applications together with ample background material.

Some care should be taken in the numerical evaluation of the problem functions and their gradients. Coding the functions offers an opportunity for considerable analysis and clever programming. This task is done only once, against the many times that the functions are evaluated during the optimization. A seemingly innocent equation or a naive way of combining terms can lead to poor numerical results. The objective function can be complicated, and a single evaluation for a set of parameters can involve:

1. A solution of a system of differential equations
2. Inverting a matrix
3. Table lookups or interpolation
4. All of the above

Gear\(^{37}\) and Calahan\(^{38}\) discuss similar numerical problems and some useful techniques in applying MP to engineering design. One of the goals of mathematics as applied to computer-aided design is to free the applications-oriented user from the standard numerical worries. For example, the routine should be able to analyze the problem and choose the best method for integrating a system of nonlinear differential equations. State-of-the-art techniques do not meet this goal; thus, it is still up to the user to make the proper selection of numerical methods.

Duality is a case in which careful analysis can pay off generously. Full details with examples are presented in the section on duality, but we present this idea here to point out some analysis and numerical considerations. In some applications of linear programming, problems occur which have a much larger number of constraints than variables. To be specific, we may have 50 constraints and 10 variables. To obtain a solution to the LP problem posed in this way, an LP code would essentially invert a 50-by-50 matrix. This inversion can be time-consuming and perhaps inaccurate for such a large matrix. We show in Duality in MP how to cast this problem as a dual linear programming problem which would require only a 10-by-10 matrix to be inverted. Duality is an excellent example of a little analysis saving a great deal of time and effort.
Suppose that the design problem is clearly stated as an MP problem. In order to solve the resulting problem, it may be necessary to transform it into an equivalent MP problem. The possible modifications can occur in the light of the following questions:

1. Is the necessary computer code available?
2. Would duality aid in obtaining numerical results?
3. Is an initial point available?
4. Can the derivatives be easily calculated?
5. Would a postoptimal analysis be useful?
6. Can significant benefit be gained by scaling the variables or clever coding techniques?

It does little good to cast a design problem as an MP problem, if we lack the numerical means to solve it. Thus, the user should be prepared to transform his problem into a solvable form or into a more useful form. In the ensuing pages we address ourselves to problem modifications which we found useful in practice.

SUMT AND CONSTRAINT TRANSFORMATION

The most common mismatch is that of a constrained problem to be solved and a routine for solving only unconstrained problems. The sequential unconstrained minimization techniques (SUMT) developed by Fiacco and McCormick\(^2\) transform a constrained MP problem into an equivalent sequence (in terms of having the same solution) of unconstrained problems. The transformation takes place with the aid of an unconstrained auxiliary function which has the following form:

\[
\phi(x, r_k) \equiv f(x) + p(r_k) \sum_{i=1}^{m} G(g_i(x))
\]  

where \(r_k\) is a parameter, \(G(y)\) is a monotonic function of \(y\) that behaves in some well chosen manner at \(y = 0\), and \(p(r)\) is a function of \(r\) which depends on the choice of \(G\). Typical choices require \(G(y) > 0\) for \(y > 0\) and \(G(y) = 0\) for \(y < 0\), or require that \(G(y)\) approach \(-\infty\) as \(y\) approaches 0 through values less than zero. The first choice of \(G\) is usually associated with procedures that are not concerned with constraint satisfaction except at the solution (exterior methods); and the second choice of \(G\) is associated with procedures which enforce constraint satisfaction throughout the minimization (interior methods). The basic idea of SUMT is the following. Let \(x^*\) be a solution to problem 1; that is, we assume \(x^*\) minimizes \(f(x)\), subject to \(g_i(x) < 0\) for \(i=1, \ldots, m\). Then under appropriate conditions\(^2\) on the problem functions the following theorem holds.
If \( x_k \) is a sequence of points, each of which minimizes \( \phi(x, r_k) \) where \( r_k \) is a sequence of points tending to zero, then we have the limit \( x_k \rightarrow x^* \), or in some cases limit \( f(x_k) \rightarrow f(x^*) \). So a constrained problem is replaced by an equivalent sequence (in the sense of having a common solution) of unconstrained problems. Common choices for \( G(y) \) are \(-1/y\), \((\min \{y, 0\})^{1+\epsilon}, \epsilon > 0\), and \(-\log (-y)\) with \( p(r) = r, 1/r, r \) respectively. For the first form of \( G(y) \), \( \phi(x, r_k) \) becomes
\[
\phi(x, r_k) = f(x) + r_k \sum_{j=1}^{m} \left\{ -1/g_j(x) \right\}
\]
then if we have a feasible point \( x^0 \) and seek to minimize \( \phi(x, r_k) \), the term \( \sum_{j=1}^{m} 1/g_j(x) \) keeps intermediate test points in the constraint set. For, if the routine attempts to leave the feasible set, it must cross the boundary; this causes \( \sum_{j=1}^{m} 1/g_j(x) \) to approach \(+\infty\), and, since we are minimizing, the routine automatically avoids points which yield large values of \( \phi(x, r_k) \). As \( r_k \) approaches zero, \( x_k \) is approaching \( x^* \), and since \( x_k \) is used as the initial point to find \( x_{k+1} \), each successive minimization requires fewer iterations. In Lagrange Multipliers as a Design Aid we give some examples.

Variations of the above techniques can be applied in many situations where there is a mismatch between computer code and mathematical programming problem. For example, Rosen's gradient projection method solves the following problem:

\[
\text{Minimize } f(x) \quad (16)
\]
Subject to
\[
\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i = 1, 2, \ldots, m
\]
Now suppose the MP problem we have on hand has some nonlinear constraints \( g_i(x) \leq 0 \), \( i = 1, \ldots, k \), as well as some linear ones. We modify the problem as follows:

\[
\text{Minimize } f(x) - r_k \sum_{i=1}^{g} 1/g_i(x) \quad (17)
\]
subject to
\[
\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i = 1, \ldots, m
\]
Thus, we take advantage of a good routine which our problem does not quite fit.
Another method which allows us to make use of an unconstrained computer code is a straight constant transformation. Many times in Problem 1 the constraints are fairly simple—for example, $a_i \leq x_i \leq b_i$ or $c_i \leq a_i x_i + b_i \leq d_i$—and the objective function is very complicated with a difficult gradient to compute. The simple nature of the above constraints allows an initial feasible point to be obtained immediately, and the constraint transformation keeps the intermediate points feasible. This transformation (as well as SUMT) allows a direct search method to be used for optimizing complicated objective functions. However, as the examples show, there is a point of diminishing return in trying to transform all the constraints, since, as the constraints become complicated, it can be impossible to find a well-mannered constraint transformation to do the job.

If we require $a \leq x \leq b$, then the following transformations keep $x$ within this range:

1. $x = \frac{b+a}{2} + \frac{(b-a)}{2} \sin y$
2. $x = a + (b-a) \sin^2 y$
3. $x = a + (b-a) \frac{e^y}{e^y + e^{-y}}$ for $y$ unconstrained.

For one-sided boundaries—i.e., $a \leq x$—we have:

4. $x = a + e^y$
5. $x = a + |y|$
6. $x = a + y^2$

The next types of constraints are linear inequalities—e.g., $d < a x + b < c$. However, this is equivalent to a boundary inequality; that is,

$$\frac{c-b}{a} \leq x \leq \frac{c-b}{a} \text{ for } a > 0.$$  

If we have two linear inequalities in two unknowns, a transformation is still possible:

$$a \leq b_1 x_1 + b_2 x_2 \leq c$$
$$e \leq b_1 x_1 + b_2 x_2 \leq f$$

then let

$$y_1 = b_{11} x_1 + b_{12} x_2$$
$$y_2 = b_{21} x_1 + b_{22} x_2$$

then

$$x_1 = \frac{(b_{22} y_1 - b_{12} y_2)}{D}$$
$$x_2 = \frac{(b_{21} y_1 - b_{11} y_2)}{D}$$
where \( D = b_{22} b_{11} - b_{12} b_{21} \)

then let

\[
\begin{align*}
    y_1 &= a + (c-a) \sin^2 z_1 \\
    y_2 &= e + (f-e) \sin^2 z_2
\end{align*}
\]

Thus, we have made \( x_1, x_2 \) functions of the unrestricted variables \( z_1, z_2 \), and they still satisfy the inequality constraints.

**OTHER EXAMPLES**

1. Suppose we require \( 0 < x_1 < x_2 < x_3 \)

   then consider

   \[
   \begin{align*}
       x_1 &= y_1^2 \\
       x_2 &= y_1^2 + y_2^2 \\
       x_3 &= y_1^2 + y_2^2 + y_3^2
   \end{align*}
   \]

2. This example was presented in Box 39

   maximize \( f = [9 - (x_1 - 3)^2] x_2^3/27 \sqrt{3} \)

   subject to

   \[
   \begin{align*}
       0 &< x_1 \\
       0 &< x_2 < x_1/\sqrt{3} \\
       0 &< x_1 + \sqrt{3} (x_2) < 6
   \end{align*}
   \]

   Initial point

   \[
   \begin{align*}
       x_1 &= 1, x_2 = .5 \quad & f = .01336 \\
       \text{optimum} & \quad & x_1 = 3, x_2 = \sqrt{3} \quad & f = 1.0
   \end{align*}
   \]

**TRANSFORMATION OF CONSTRAINTS**

\[
\begin{align*}
    y_1 &= x_1 + \sqrt{3} \cdot x_2 \\
    y_2 &= x_2 - x_1/\sqrt{3}
\end{align*}
\]

Then if \( x_2 = x_1/\sqrt{3} \), we have \( 0 < x_1 + x_1 < 6 \) or \( x_1 <= 3 \). This then gives bounds for the variables \( y_1 \) and \( y_2 \):

\[
\begin{align*}
    -x_1/\sqrt{3} &< x_2 - x_1/\sqrt{3} < 0 \\
    -\sqrt{3} &< -3/\sqrt{3} < y_2 < 0. \quad \text{Thus we have}
\end{align*}
\]
This implies
\[ x_1 = \frac{1}{2} (y_1 - \sqrt{3}y_2) \]
\[ x_2 = \frac{1}{2} (y_1/\sqrt{3} + y_2) \]

Thus, we have the unconstrained problem

\[
\text{maximize } f(x_1(z_1, z_2), x_2(z_1, z_2))
\]
where
\[ x_1 = \frac{1}{2} (y_1 - \sqrt{3}y_2) \]
\[ x_2 = \frac{1}{2} (y_1/\sqrt{3} + y_2) \]
\[ y_1 = 6 \sin^2 z_1 \]
\[ y_2 = -\sqrt{3} + \sqrt{3} \sin^2 z_2 \]

The necessary FORTRAN code to evaluate this objective function, if we were using NELC DIRECT to solve the minimization problem, is as follows:

```fortran
SUBROUTINE FN(Z, N, F)
    DIMENSION Z(2)
    Y1 = 6.0*(SIN(Z(1)))**2
    Y2 = -2.0*SQRT(3)*(1-(SIN(Z(2)))**2)
    X1 = .5*(Y1-SQRT(3)*Y2)
    X2 = .5*(Y1/SQRT(3)+Y2)
    F = (9-(X1-3.)*X2)/(27.0*SQRT(3))
RETURN
END
```

The best transformation to use depends both on the problem and the minimization scheme used. For example, transformation 5 is not differentiable at \( y = 0 \); hence, a gradient method would not be valid at this point. Method 3 would make coding the gradient of the objective function lengthy and time-consuming to debug. These problems do not occur if a direct search method is used.
Let us now suppose the problem is matched to the routine and further suppose the computer code is a gradient method. The user must supply a subroutine or function that will compute the gradient; that is, he must provide the following vector:

$$\nabla f(x) = \begin{pmatrix} \frac{df}{dx_1} & \frac{df}{dx_2} & \cdots & \frac{df}{dx_n} \end{pmatrix}$$

(18)

Often, the complexity of the objective function $f(x)$ or of some of the constraint function is such that the gradient cannot be computed, or the time required to derive $\nabla f$ analytically is excessive. Then, the designer, in order to use the gradient methods, can decide to approximate the gradient. In the process, the function must be evaluated several times in the vicinity of a point $x$ for each approximation. Thus, a trade-off situation arises, since a more accurate estimate will require many function evaluations which may ultimately be more costly (in computer time) than direct calculation of the gradient (in manhours). The problem becomes one of selecting the most accurate approximation for the least number of function evaluations.

We list some of the standard schemes for approximating the derivative of a function together with the corresponding error estimates (Hildebrand40). Let

$$f_1 = f(x_1, x_2, \ldots, x_i + \Delta, \ldots, x_n)$$

(19)

$$f_0 = f(x_1, x_2, \ldots, x_i, \ldots, x_n)$$

(20)

$$f_{-1} = f(x_1, \ldots, x_i - \Delta, \ldots, x_n)$$

(21)

$$e = \frac{\partial^3 f}{\partial x_i^3} (x_1, x_2, \ldots, x_i + \xi, \ldots, x_n) \text{ where } |\xi| < \Delta$$

(22)

Then

$$\frac{\partial f_1}{\partial x_i} = \frac{1}{2\Delta}(-3f_{-1} + 4f_0 - f_1) + \frac{e\Delta^2}{3}$$

(23)

$$\frac{\partial f_0}{\partial x_i} = \frac{1}{2\Delta}(f_{-1} + f_1) - \frac{e\Delta^2}{6}$$

(24)

$$\frac{\partial f_{-1}}{\partial x_i} = \frac{1}{2\Delta}(f_{-1} - 4f_0 + 3f_1) + \frac{e\Delta^2}{3}$$

(25)

We note that the best estimate for $\frac{\partial f}{\partial x_i}$ is equation (29), for if $|\partial^3 f/\partial x_i^3| < M_3$ as $\xi$ varies over $|\xi| < \Delta$ we have the maximum error $|e| \text{ max } = M_3/6$.

It is interesting that the point in question does not appear in the formula, even though $f_0$ is generally available. This additional information does not improve the estimate. The next question is, now should $\Delta$ be chosen?
Hildebrand gives an optimal $\Delta$ which is derived analytically via $\partial^3 f(x)/\partial x^3$. Unfortunately, this expression is generally not available when we are attempting to approximate $\partial f(x)/\partial x_i$. Also, this optimal $\Delta$ is a function of the test point $x$, which changes many times in the course of a minimization. If this approach is taken, then an educated guess will have to be made for $\Delta$. Experiments have indicated that for the most efficient operation, $\Delta$ should be changed automatically by the code. A method along this line is Stewart’s modification of Davidson’s minization routine.

Steward’s method is dependent on the information generated by the minimization part of the routine. Using this information, he is able to select a good $\Delta_i$ and initially estimate the $i$th component of the gradient by the formula $(f_1 - f_0)/\Delta_i$. When this simple scheme begins to fail, the routine automatically switches to the central difference method for a more accurate estimate. $\Delta_i$ is also suitably modified. His method required 163 function evaluations to find the minimum of the Rosenbrock function to within five decimal places. This compares with 325 for the direct search method (ZANGWL) and 71 for the gradient method (CNJGAT), to minimize to approximately the same accuracy. Unfortunately, Steward’s method was coded in a language for the CDC 1604 and has not been modified for the NELC IBM 360.

The final technique for general numerical differentiation which we discuss, is a code from the IBM Scientific Subroutine Package, called DDCAR. DDCAR uses an extrapolation technique to obtain highly accurate estimates of the gradient. This method does not need an optimal $\Delta$ to give good results. However, we must pay the price of using a larger number of function evaluations than the central difference method would require, for each estimate of $\nabla f$. A code and results of using DDCAR to compute the gradient in minimizing the Rosenbrock function with the gradient method NELC CNJGAT are presented in table 1. 1850 function evaluations were required to obtain these results.

The methods presented above are general and can be used for a wide class of functions. Some applications may lend themselves to special methods for estimating the gradient. For example, Calahan discusses methods for numerically evaluating the gradient in a network optimization scheme. The method is highly specialized for this type of problem. He uses a “calculus of variations” approach together with numerical integration. The section on gradient calculation is a good example of the success of careful analysis.
IMPLICIT REAL*8 (A-H,O-Z)
COMMON KOUNT
DIMENSION X(30), H(30,30), G(30), S(30), SIGMA(30), XX(20)
EXTERNAL FGBOX
X(1)=-1.2000
X(2)=-1.0000
KOUNT= 0
NMAX = 30
MPRNT = 1
N= 2
ISTART = N + 1
EPSLON = 1.0D-10
FEST = 0.0
ITERBD = 80
SBOUND = 1.0D-09
ICNT = 0
!REST = 0
ITER = 0
NMI = N - 1
DO 30 I = 1, NMI
H(I,1) = 1.0
IP1 = I + 1
DO 30 J = IP1, N
H(J,1) = 0.0
30
H(I,J) = 0.0
H(N,N) = 1.0
DO 20 J=1,20
20
S(J) = 0.
CALL CNJGAT(FGBOX,N,NMAX,X,ITERBD,EPSLON,FEST,MPRNT,ISTART,ITE F,G,S)
PRINT 999,KOUNT
999 FORMAT(IX,'KOUNT',IS)
END

FUNCTION USERF (X,N)
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION X(N)
T1=X(2)-X(1)*X(1)*X(1)
T2=X(1)-1.0D00
USERF=100.0D00*T1*T1 + T2*T2
RETURN
END

FUNCTION FUNC(H)
IMPLICIT REAL*8 (A-H,O-Z)
COMMON KOUNT,GRAD/XX,IVAR,NN
DIMENSION XTEMP(20), XX(20)
DO 30 J = 1,NN
IF (J .EQ. IVAR) GO TO 29
XTEMP(J) = XX(J)
30
...
GO TO 30
29 XTEMP (IVAR) = XX(IVAR) + H
30 CONTINUE
FUNC=USEQ(XTEMP,NN)
KOUNT = KOUNT + 1
RETURN
END

SUBROUTINE FG PXI( X,F,G,N)
IMPLICIT REAL*(A-H,O-Z)
EXTERNAL FUNC
DIMENSION X(1),G(1),XX(2G)
COMMON/GRAD/XX,IVAR,NN
NN = N
DD 10 J=1,N
10 XX(J) = X(J)
IVAP = 1
F = FUNC(G)
DD 20 K=1,N
IVAR = K
WZ = .001
CALL DUCAR(C*,WZ,1,FUNC,Y)
20 G(K) = Y
RETURN
END
<table>
<thead>
<tr>
<th>Iteration</th>
<th>F</th>
<th>X(1)</th>
<th>X(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.57838400D 02</td>
<td>-0.10117724D 01</td>
<td>-0.10432685D 01</td>
</tr>
<tr>
<td>2</td>
<td>0.40513033D 01</td>
<td>-0.10125637D 01</td>
<td>-0.10412920D 01</td>
</tr>
<tr>
<td>3</td>
<td>0.38829400D 01</td>
<td>-0.93251385D 00</td>
<td>-0.77233367D 00</td>
</tr>
<tr>
<td>4</td>
<td>0.36809335D 01</td>
<td>-0.91520878D 00</td>
<td>-0.77747070D 00</td>
</tr>
<tr>
<td>5</td>
<td>0.36807309D 01</td>
<td>-0.91847047D 00</td>
<td>-0.77339670D 00</td>
</tr>
<tr>
<td>6</td>
<td>0.36788232D 01</td>
<td>-0.91715735D 00</td>
<td>-0.76572077D 00</td>
</tr>
<tr>
<td>7</td>
<td>0.36621503D 01</td>
<td>-0.78236780D 00</td>
<td>-0.43965944D 00</td>
</tr>
<tr>
<td>8</td>
<td>0.33307413D 01</td>
<td>-0.73334700D 00</td>
<td>-0.33842677D 00</td>
</tr>
<tr>
<td>9</td>
<td>0.33177071D 01</td>
<td>-0.6999652D 00</td>
<td>-0.35578328D 00</td>
</tr>
<tr>
<td>10</td>
<td>0.29063527D 01</td>
<td>-0.7004484D 00</td>
<td>-0.34461374D 00</td>
</tr>
<tr>
<td>11</td>
<td>0.28919187D 01</td>
<td>-0.51999518D 00</td>
<td>-0.11405523D 00</td>
</tr>
<tr>
<td>12</td>
<td>0.23808990D 01</td>
<td>-0.48818580D 00</td>
<td>-0.13704749D 00</td>
</tr>
<tr>
<td>13</td>
<td>0.22575477D 01</td>
<td>-0.46833270D 00</td>
<td>-0.98170996D-01</td>
</tr>
<tr>
<td>14</td>
<td>0.21580722D 01</td>
<td>0.64254454D 00</td>
<td>0.24119918D 00</td>
</tr>
<tr>
<td>15</td>
<td>0.18577830D 00</td>
<td>0.64207173D 00</td>
<td>0.25080394D 00</td>
</tr>
<tr>
<td>16</td>
<td>0.13566572D 00</td>
<td>0.70499570D 00</td>
<td>0.31316119D 00</td>
</tr>
<tr>
<td>17</td>
<td>0.98975412D-01</td>
<td>0.70499570D 00</td>
<td>0.31316119D 00</td>
</tr>
</tbody>
</table>

31
18
0.70097588D-01
\(X(1) = 0.79903323D\ 00\)
\(X(2) = 0.49290947D\ 00\)
19
0.43186973D-01
\(X(1) = 0.79235939D\ 00\)
\(X(2) = 0.64854209D\ 00\)
20
0.37085901D-01
\(X(1) = 0.87194316D\ 00\)
\(X(2) = 0.64854209D\ 00\)
21
0.37085901D-01
\(X(1) = 0.87194316D\ 00\)
\(X(2) = 0.64854209D\ 00\)
22
0.17549333D-01
\(X(1) = 0.86684297D\ 00\)
\(X(2) = 0.65086901D\ 00\)
23
0.10469106D-01
\(X(1) = 0.92094261D\ 00\)
\(X(2) = 0.77458511D\ 00\)
24
0.10043283D-01
\(X(1) = 0.9375552D\ 00\)
\(X(2) = 0.8066209D\ 00\)
25
0.47209678D-02
\(X(1) = 0.9313289D\ 00\)
\(X(2) = 0.8075809D\ 00\)
26
0.19935622D-02
\(X(1) = 0.9632844D\ 00\)
\(X(2) = 0.8913062D\ 00\)
27
0.18329659D-03
\(X(1) = 0.99500316D\ 00\)
\(X(2) = 0.9838259D\ 00\)
28
0.28885366D-04
\(X(1) = 0.9966279D\ 00\)
\(X(2) = 0.9838259D\ 00\)
29
0.50345410D-06
\(X(1) = 0.9998186D\ 00\)
\(X(2) = 0.999889D\ 00\)
30
0.12440066D-07
\(X(1) = 0.9998892D\ 00\)
\(X(2) = 0.9996682D\ 00\)
31
0.12228058D-07
\(X(1) = 0.9998894D\ 00\)
\(X(2) = 0.9996681D\ 00\)
32
0.13578763D-12
\(X(1) = 0.10000000D\ 01\)
\(X(2) = 0.9999999D\ 00\)
33
0.87952785D-18
\(X(1) = 0.10000000D\ 01\)
\(X(2) = 0.10000000D\ 01\)
34
0.87854680D-18
\(X(1) = 0.10000000D\ 01\)
\(X(2) = 0.10000000D\ 01\)
35
0.24394397D-23
\(X(1) = 0.10000000D\ 01\)
\(X(2) = 0.10000000D\ 01\)
36
0.59070910D-27
\(X(1) = 0.10000000D\ 01\)
\(X(2) = 0.10000000D\ 01\)
DUALITY IN MP

In linear programming and certain SUMT methods with convex functions, the possibility of using duality to gain computational efficiency should be considered.

Duality occurs in many areas – mathematics, engineering, economics, physics, etc. In a mathematical or engineering context it implies that two concepts or systems have a specific mathematical relationship. We give an example from circuit theory before proceeding with duality and MP.

Consider the following series RCL network (fig. 5).

![Series RCL network](image)

**Figure 5. Dual circuits.**

The mathematical equation representing (A) is

\[
L \frac{di}{dt} + Ri + \frac{1}{C} \int_0^t idt = e(t)
\]  
(26)

If we interchange e and i and L and C and replace R by G = 1/R, we have

\[
C \frac{de}{dt} + Ge + \frac{1}{L} \int_0^t edt = i(t)
\]  
(27)

which is the equation which models the parallel network (B). Thus, an equation of the form

\[
a \frac{dx}{dt} + bx + c \int_0^t xdt = y(t)
\]

can represent either circuit (A) or circuit (B) depending on the parameters a, b, c; that is, it has a dual function, and we say that (A) is the dual of (B).

In linear programming, we have the most straightforward application and complete theory of duality. Let x and c be vectors of length n, b a vector of length m, and A an m-by-n matrix. Then the linear programming problem seeks to find the vector x which produces...
a minimum\[z = c^T x\]
subject to\[
Ax \leq b \\
x > 0.
\]

To this linear programming problem there corresponds an associated problem.

Maximize \[v = b^T w\]
subject to\[
A^T w \geq c \\
w \geq 0
\]

where \(w\) is an \(m\) vector. Problem (28) is called the primal problem while problem (29) is referred to as the dual problem.

In the primal formulation we have \(m\) constraints and \(n\) variables, and just the opposite in the dual formulation. Some facts about these corresponding problems which are pertinent to this section are:

1. The primal problem has a bounded solution \(x^*\) if and only if the dual has a bounded solution \(w^*\).
2. \(w^*\) can be obtained from \(x^*\) and vice versa.
3. The dual of the dual is the primal.
4. In the case of bounded solutions we have \(z^* = v^*\).

The efficiency of linear programming computer codes decreases as the number of constraints increases. It is this trait which makes the study of the duality relationship worthwhile from a computational standpoint. The following example shows how this loss of efficiency can be lessened by judicious use of the dual.

Consider the following programming problem:

Minimize \[z = -4x_1 - 3x_2\]
subject to\[
x_1 \leq 6 \\
x_2 \leq 8 \\
x_1 + x_2 \leq 7 \\
3x_1 + x_2 \leq 15 \tag{30}
\]
Note that $x_2$ can be negative. To pose this as a standard LP problem, we replace the variable $x_2$ by the difference of two positive variables; that is, we let $x_2 = x'_2 - x''_2$. Now the problem can be written in standard form.

Minimize $-4x_1 - 3(x'_2 - x''_2)$

subject to

$$-x_2 \leq 1$$

$$x_1 \geq 0,$$

Now the problem can be written in standard form.

$$x_1 \leq 6$$

$$x'_2 - x''_2 \leq 8$$

$$x_1 + x'_2 - x''_2 \leq 7$$

$$3x_1 + x'_2 - x''_2 \leq 15$$

$$-x'_2 + x''_2 \leq 1$$

$$x_1, x'_2, x''_2 \geq 0$$

Writing the constraints in matrix notation, we have

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 1 & 1 & -1 \\ 3 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x'_2 \\ x''_2 \end{bmatrix} \leq \begin{bmatrix} 6 \\ 8 \\ 7 \\ 15 \\ 1 \end{bmatrix}$$

In solving this LP problem, we add another positive variable $x_{s1}$, called a slack variable, to each row, to make each inequality an equality. The problem then becomes:

MPS/360 accepts unrestricted variables and automatically makes this transformation.
Minimize 

\[-4x_1 - 3x_1' + 3x_2'' + 0x_{s1} + \ldots + 0x_{s5}\]

subject to

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 1 & 0 & 0 \\
1 & 1 & -1 & 0 & 0 & 1 & 0 \\
3 & 1 & -1 & 0 & 0 & 0 & 1 \\
0 & -1 & 1 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_2' \\
x_{s1} \\
x_{s2} \\
x_{s3} \\
x_{s4} \\
x_{s5}
\end{bmatrix}
\begin{bmatrix}
6 \\
8 \\
7 \\
15 \\
1
\end{bmatrix} = \begin{bmatrix}
0 \\
-1 \\
0 \\
1
\end{bmatrix}
\tag{32}
\]

Note that each slack variable appears in the cost row with zero for a coefficient.

The simplex method of LP performs an iterative process on five selected columns of this augmented matrix to obtain a basis inverse matrix. This matrix is then used to obtain the solution. It is this inversion process which causes the efficiency of LP codes to decrease as \( m \) increases. We now cast problem (31) in its dual formulation, which will reduce the number of rows.

Maximize

\[v = 6w_1 + 8w_2 + 7w_3 + 15w_4 + w_5\]

subject to

\[
A^T w = \begin{bmatrix}
1 & 0 & 1 & 3 & 0 \\
0 & 1 & 1 & 1 & -1 \\
0 & -1 & -1 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4 \\
w_5
\end{bmatrix}
\begin{bmatrix}
-4 \\
-3 \\
3
\end{bmatrix} \tag{33}
\]

Now from each row we subtract a positive variable \( w_{si} \) and write the constraint matrix as:

\[
\begin{bmatrix}
1 & 0 & 1 & 3 & 0 & -1 & 0 & 0 \\
0 & 1 & 1 & 1 & -1 & 0 & -1 & 0 \\
0 & -1 & -1 & -1 & 1 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4 \\
w_5 \\
w_{s1} \\
w_{s2} \\
w_{s3}
\end{bmatrix}
\begin{bmatrix}
-4 \\
-3 \\
3
\end{bmatrix}
\]
We now have a formulation similar to problem (33); however, to solve this problem, it is necessary to "invert" only a 3-by-3 matrix. Even though the number of variables is the same, the smaller number of constraints makes this formulation more efficient. If we were to solve the dual formulation using MPS/360, since the dual of the dual is the primal, the solution to problem (31) would appear under the DUAL ACTIVITY heading.

There is another form of the primal-dual relationship which has both formulative and computational advantages; it is the unsymmetric form.

The primal problem can also be stated as: find a column vector \( x^* \) which

\[
\text{minimize } z = c^T x
\]

subject to

\[
Ax = b
\]
\[
x \geq 0
\]

The original LP problem (28) can be put in this form by adding slack variables to transform the inequality constraints into equality constraints. Then the unsymmetric dual to (34) is

\[
\text{Maximize } v = b^T w
\]

subject to

\[
A^Tw \leq c
\]

We notice in (35) that there is no restriction on the sign of \( w \). This is most useful in using linear programming as an analysis tool, and to condense the problem size. The three previous properties for the symmetric form of the dual also hold for the unsymmetric form.

We return to problem (30) to give a simple example of the usefulness of this unsymmetric form. The problem is written as

\[
\text{Maximize } v = 4w_1 + 3w_2
\]

subject to

\[
A^Tw = \begin{bmatrix}
1 & 0 \\
0 & 1 \\
1 & 1 \\
3 & 1 \\
0 & -1 \\
-1 & 0
\end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \leq \begin{bmatrix} 6 \\ 8 \\ 7 \\ 15 \\ 1 \\ 0 \end{bmatrix}
\]

\[
(36)
\]
Note that $w_2$ is unrestricted in sign and that the last row of $A^T$ keeps $w_1$ nonnegative. This is the unsymmetric dual formulation of problem (30). Since $w_2$ is unrestricted in sign, to solve this problem via a standard LP code, we should have to replace $w_2$ by $w_2 - w_2^*$, as before. But if we consider the associated primal problem, this requirement disappears; i.e.,

Minimize $6x_1 + 8x_2 + 7x_3 + 15x_4 + x_5$

subject to

$$A x = \begin{bmatrix} 1 & 0 & 1 & 3 & 0 & -1 \\ 0 & 1 & 1 & 1 & -1 & 0 \\ \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ \end{bmatrix} \quad (37)$$

To solve this primal problem we must invert only a 2-by-2 matrix rather than a 5-by-5 as in the dual case. When we solve this program using MPS/360, the optimal $w$ will appear as a DUAL ACTIVITY.

This method accomplishes two things:

1. There is no increase in the number of variables.
2. If $m$ is much larger than $n$, then the primal has fewer constraints and will generally be faster to solve.

Of course, the form we decide to use will depend on the relative sizes of $m$ and $n$.

We present another example which will illustrate both the use of the unsymmetric dual and the utility of LP in the area of applied mathematics; viz., linear boundary value problems. Consider the following problem. Find a solution, $y$, to

$$L[y] = r(x) \text{ over } [a, b],$$

where

$$L[y] = \sum_{j=0}^{n} f_j(x)y^{(j)} = f_0(x)y + f_1(x)y^{(1)} + \ldots + f_n(x)y^{(n)} \quad (38)$$

with boundary conditions $V_j[y] = \Gamma_j, j = 1, 2, \ldots, n$

$$V_j[y] = \sum_{k=0}^{n-1} (\alpha_{j,k}y^{(k)}(a) + \beta_{j,k}y^{(k)}(b)) \quad (39)$$

where $y^{(i)}$ is the $i$th derivative of $y(x)$ with respect to $x$. 

38
Some boundary value problems do not have a closed form solution or even a
solution in the limit; however, we should still like some information and in
most instances an approximate solution is sufficient. The approximation is
made in the min-max sense; i.e., if \( f^* \) is the theoretical solution to, say, a
differential equation over an interval \([a, b]\), then an approximation \( f_\alpha \) is
sought to

\[
\text{minimize } \max_{x \in [a, b]} |f_\alpha(x) - f^*(x)|
\]

This formulation lends itself to an application of mathematical programming.
We approximate the solution by a sum of functions over the interval \([a, b]\).
To do this, we partition \([a, b]\) as follows:

\[
a < x_1 < x_2 < \ldots < x_m < b
\]

which minimize

\[
\max_{1 \leq i \leq m} |L(y^*(x_i)) - r(x_i)|
\]

where

\[
y^*(x) = y_0(x) + \sum_{j=1}^{p} a_j y_j(x),
\]

\[y_0(x) \text{ satisfies } V_r(y_0) = \Gamma_r, \quad r = 1, \ldots, n\]

and \( y_j(x) \) satisfy \( V_r(y_j) = 0, \quad j = 1, \ldots, p \)

Substituting equation (42) into equation (41), and introducing an additional
variable \( \epsilon \), we wish to find \( a_1, \ldots, a_p, \epsilon \) which minimizes \( \epsilon \) subject to

\[
|L(y_0(x_i) + \sum_{j=1}^{p} a_j y_j(x_i)) - r(x_i)| < \epsilon
\]

for \( i = 1, 2, \ldots m \)

For LP to be applied, the constraints must be linear; thus, we have

Minimize \( \epsilon \)

\((a_1, a_2, \ldots, a_p, \epsilon)\)

subject to

\[
-\epsilon \leq \sum_{i=1}^{p} a_i L[y_i(x_j)] + L[y_0(x_j)] - r(x_j) \leq \epsilon \quad j = 1, 2, \ldots, m.
\]
or, rewriting as two single inequalities,

\[
\begin{align*}
- \sum_{i=1}^{p} a_i L(y_i(x_j)) - L(y_o(x_j)) + r(x_j) & \leq e \\
\sum_{i=1}^{p} a_i L(y_i(x_j)) + L(y_o(x_j)) - r(x_j) & \leq e
\end{align*}
\] (43)

Collecting all the parameters on the left side of the inequality yields

\[
\begin{align*}
- \sum_{i=1}^{p} a_i L(y_i(x_j)) - e & \geq L(y_o(x_j)) - r(x_j) \\
\sum_{i=1}^{p} a_i L(y_i(x_j)) - e & \geq r(x_j) - L(y_o(x_j))
\end{align*}
\] (44)

for \( j = 1, 2, \ldots m \)

where

\[
a = x_1 < x_2 \ldots < x_m = b \text{ is a partition of } [a, b].
\]

Form the \( p \)-by-\( m \) matrix:

\[
G = \begin{bmatrix} L[y_1(x_1)] & \ldots & L[y_1(x_m)] \\ \vdots & \ddots & \vdots \\ L[y_p(x_1)] & \ldots & L[y_p(x_m)] \end{bmatrix}
\]

the 2m vector

\[
c^T = \begin{bmatrix} L(y_o(x_1) - r(x_1)), \ldots, L(y_o(x_m) - r(x_m)), r(x_1) - L(y_o(x_1)), \ldots, \\ r(x_m) - L(y_o(x_m)) \end{bmatrix}
\]

the \((p + 1)\) vectors

\[
\alpha^T = (a_1, \ldots, a_p, \epsilon) \\
b^T = (0 \ldots 0, -1)
\]
and the matrix

\[
A = \begin{bmatrix}
G & -G \\
-1, \ldots, -1 & -1, \ldots, -1
\end{bmatrix}
\]

Since there is no restriction on the sign of the parameters \(a_i (i=1, \ldots, p)\), we cast the programming problem in the dual formulation; i.e.,

\[
\text{Maximize } -\epsilon = a^T \cdot b \\
\begin{array}{c}
(a_1, a_2, \ldots, a_p, \epsilon) \dagger
\end{array}
\]

subject to

\[
\begin{bmatrix}
G^T & -1 \\
& \ddots & -1 \\
& & \ddots & -1 \\
-G^T & & & -1
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_p \\
\epsilon
\end{bmatrix}
\leq
\begin{bmatrix}
L(y_0(x_1)) - r(x_1) \\
\vdots \\
L(y_0(x_m)) - r(x_m) \\
\vdots \\
L(y_0(x_m)) - L(y_0(x_1))
\end{bmatrix}
\]

or more succinctly as

\[
\text{Maximize } a^T b \\
\text{(dual)}
\]

\[
A^T a \leq c
\]

then, since the dual of the dual is the primal, we have

\[
\text{Minimize } c^T x \\
\text{(primal)}
\]

subject to

\[
A x = b \\
x \geq 0
\]

where \(x\) is a 2m vector.

This primal problem is then solved numerically and the optimal \(\alpha\) vector appears as the DUAL ACTIVITY vector.

\[\dagger\text{Note inequalities (43) and (44) keep } \epsilon > 0.\]
In summary, the basic advantages are as follows: We wish parameters $\varepsilon_1, \ldots, \varepsilon_p$, which give the best estimate in a min-max sense of the solution to the linear boundary value problem. The parameters are unrestricted in $[a, b]$ and the number of points $m$ in the interval of solution is large. To cast this problem as a primal linear programming problem would have required $2p + 1$ variables, since the $a_i$ are unrestricted, and $2m$ constraints. By first writing the approximation problem as a dual linear programming problem (45), we need only $p + 1$ variables; then transforming to the primal reduces the number of constraints from $2m$ to $p + 1$. We give two examples which use the above method. Example 1: A homogeneous equation with inhomogeneous boundary conditions

$$L(y) = xy''' - (x+1)y' - 2(x-1)y = 0; y(0) = 1, y(1) = 0$$

on $[0, 1]$,

$$y^*(x) = y_0(x) + \sum_{j=1}^{p} a_jy_j(x)$$

with

$$y_0(x) = 1-x; y_j(x) = x^j(1-x)$$

$j = 1, 2, \ldots, p$

then

$$L[y_0] = 2x^2 - 3x + 3$$

$$L[y_j] = 2x^{j+2} + (j-3)x^{j+1} - (j+2)x^j + j(j-2)x^{j-1}$$

$$x_i = (i-1)(0.05), \quad i = 1, 2, \ldots, 21$$

for $p = 3$, $a_1 = 3.0374706$

$a_2 = -0.7865970$

$a_3 = 0.8839133$

$\varepsilon = 0.0374706$, and the maximum error occurred at $x = 0$.

Example 2: Inhomogeneous case with homogeneous boundary conditions

$$L[y] = y''' + (1+x^2)y = -1, y(1) = y(-1) = 0$$

on $[-1, 1]$,

$$y_j(x) = 1 - x^2j$$

$$L[y_j] = -x^2(j+1) - x^2j - 2j(2j-1)x^{2j-1} + x^2 + i$$
Let
\[ z = x^2, \quad L[y_j] = -z^{j+1} - z^j - 2j(2j - 1)z^{j-1} + z + 1 \]
\[ z_i = (i-1)(0.05) \quad i = 1, 2, \ldots, 21 \]
for \( p = 3, \quad a_1 = 0.9675, \quad a_2 = \ldots \quad a_3 = -0.0285 \]
\( e = 0.0029 \)

The preceding paragraphs have pointed out some of the numerical advantages of duality. Duality concepts also have application in nonlinear programming; however, so far these have been limited to SUMT methods involving convex functions. In this case, the duality theory provides a lower bound on the value of the unconstrained problem to test for convergence; see reference 24. Its use in nonlinear problems is not widespread.

LAGRANGE MULTIPLIERS AS A DESIGN AID

Another influence on choice of routine or method to solve an MP problem is postoptimal analysis.

When a constrained optimization problem is solved via sequential unconstrained minimization techniques (SUMT), additional information is available to the designer for a sensitivity analysis; i.e., postoptimal analysis. Suppose \( x^* \) is the solution to the following MP problem.

Minimize \( f(x) \)
subject to
\[ h_i(x) \leq b_i, \quad i = 1, 2, \ldots, m \]

The engineer wishes information as to how \( f(x^*) \) will change if \( b \) is changed a little; i.e., if the design requirements are changed, how will the performance be affected? The interesting result is:
\[ \frac{\partial f(x^*(b))}{\partial b_i} = \lambda_i \]
\[ (47) \]
where \( \lambda_i \) is a generalization of the classic Lagrange multiplier for finding the extrema with side conditions. The proof of (47) is contained in reference 44.

We recall from advanced calculus that an extrema problem with side conditions was: find the minimum (maximum) of \( p(x_1, x_2, \ldots, x_n) \) subject to \( q_1(x_1, \ldots, x_n) = \ldots = q_m(x_1, \ldots, x_n) = 0 \). To solve this problem we
formed the Lagrangian $L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = p(x) + \sum_{i=1}^{m} \lambda_i q_i(x)$

then solved the $(n+m)$ system of equations:

\[
\frac{\partial L(x, \lambda)}{\partial x_i} = 0 \quad i = 1, \ldots, n \\
q_i(x) = 0 \quad i = 1, \ldots, m
\]

for $(x_1, \ldots, x_n)$ with the $\lambda_i$'s being introduced to help find the $x_i$. The Kuhn-Tucker theorem allows us to define a meaningful and useful Lagrangian for inequality constraints.

We discuss the Kuhn-Tucker theorem to allow a generalization of Lagrange multiplier, and then discuss SUMT methods to illustrate obtaining the $\lambda_i$'s numerically.

Rewriting the constraints to problem (46) as $g_i(x) h_i(x) - b_i \leq 0$, we obtain problem (1):

Minimize $f(x)$

subject to

$g_i(x) \leq 0 \quad i = 1, \ldots, m.$

Then the Kuhn-Tucker theorem says essentially the following:

Let $x^*$ be a solution to the above problem and assume the boundary of the constraint set has no "cusps," then the following conditions hold.

1. There exist multipliers $\lambda_i \geq 0, i = 1, \ldots, m$ such that

$\lambda_i g_i(x^*) = 0, i = 1, 2, \ldots, m$

2. $x^* + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) = 0$

The following example illustrates the Kuhn-Tucker condition:

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

subject to

$g_1(x_1, x_2) = -x_2 + x_1^2 \leq 0$

$g_2(x_1, x_2) = -2 + x_1 + x_2 \leq 0$
The global solution to the above is \((x_1^*, x_2^*) = (1, 1)\). The gradients at the optimum are

\[
\nabla g_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \nabla g_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \nabla f = \begin{bmatrix} -2 \\ 0 \end{bmatrix}
\]

and the multipliers are \(\lambda_1 = \lambda_2 = 2/3\). Finally we can write:

\[-\nabla f = +2/3 \ nabla g_1 + 2/3 \ nabla g_2.\]

Geometrically we have the following (fig. 6):

Figure 6. Kuhn-Tucker conditions.
Now that we have seen a need for knowing the Lagrange multipliers and have seen the geometrical interpretation, we turn to SUMT and the numerical calculation of the multipliers. The constrained optimization problem is transformed into an unconstrained one through the use of an auxiliary function, \( \theta(x) \). \( \theta(x) \) does one of two things, \( \theta(x) \to \infty \) as \( x \to 0 \) or \( \theta(x) = 0 \) for \( x \) feasible and \( \theta(x) \) positive for \( x \) infeasible. The three forms of \( \theta(x) \) we discuss are:

- **Barrier:**
  \[
  B(x) = -\sum_{i=1}^{m} \frac{1}{g_i(x)}
  \]  
  \( (49) \)

- **Penalty:**
  \[
  P(x) = \sum_{i=1}^{m} (\text{Max}(g_i(x), 0))^{1+\epsilon}
  \]  
  \( (50) \)

- **Logarithmic:**
  \[
  L(x) = \sum_{i=1}^{m} \log(-g_i(x))
  \]  
  \( (51) \)

SUMT works as follows: Using \( \theta(x) = B(x), P(x) \) or \( L(x) \), we transform the constrained problem into a sequence of unconstrained problems, as in SUMT and Constraint Transformation.

Minimize \( D(x, r_k) = f(x) + p(r_k) \theta(x) \) \( (52) \)

for \( r_k > 0 \). Then if \( x^* \) is the optimum for the constrained problem (48) and \( x(r_k) = x_k \), the optimum for problem (52), it can be shown that \( \lim_{k \to \infty} x_k = x^* \); i.e., \( (r_k \to 0) \). Thus, we replace the constrained problem by a sequence of unconstrained problems.

In the barrier and logarithmic cases as \( g_i(x) \to 0 \) – i.e., \( x \) attempts to leave the constraint set – \( g_i(x) \) must approach 0, and this causes \( B(x) \) and \( L(x) \) to increase rapidly. Since the routine wishes to minimize \( D(x, r) \), and we have an initial feasible point, intermediate \( x \)'s are chosen feasible.

In the penalty function approach the test points are allowed to leave the constraint set; however, when they do so, a positive amount is added to the objective function. Again, since we seek to minimize \( D(x, r) \), points are selected within the constraint set. When \( B(x) \) or \( L(x) \) is used, we have an interior point method; and when \( P(x) \) is used, we have an exterior point method. With each type of \( \theta(x) \) a slightly different technique is used to recover the \( \lambda_i \).
Barrier Method:

Form the function

\[ D(x, r) = f(x) + rB(x) \]

\[ = f(x) + r \sum_{i=1}^{m} -1/g_i(x) \quad (53) \]

for \( r > 0 \). Let \( r_k \) be a monotonic decreasing sequence converging to zero. If \( x_k \) minimizes \( D(x, r_k) \), then

\[ 0 = \nabla_x D(x_k, r_k) = \nabla f(x_k) + r_k \sum_{i=1}^{m} \frac{1}{2g_i^2(x_k)} \nabla g_i(x_k) \quad (54) \]

Letting

\[ \lambda_i^{(k)} = \frac{r_k}{g_i^2(x_k)} \quad (55) \]

then the Lagrangian becomes

\[ L(x_k, \lambda_k) = f(x_k) + \sum_{i=1}^{m} \lambda_i^{(k)} g_i(x_k) \]

\[ = f(x_k) + r_k \sum_{i=1}^{m} -1 \frac{g_i(x_k)}{g_i^2(x_k)} \quad (56) \]

The multipliers can be computed by equation (55).

We note if \( g_i(x^*) > 0 \), then \( \lambda_i^{(k)} \) approaches 0 as \( k \to \infty \), since

\[ r_k \to 0. \]

If \( g_i(x^*) = 0 \), then by the Kuhn-Tucker theorem \( \lambda_i \geq 0 \); thus the limit \( \frac{r_k}{g_i^2(x_k)} \) must be taken strictly as a quotient.

Logarithmic Penalty: Form \( C(x, r) = f(x) - r \sum_{i=1}^{m} \log(-g_i(x)) \quad (57) \)

then at the optimum, \( x_k \) for \( C(x, r_k) \) we have,

\[ 0 = \nabla_x C(x_k, r_k) = \nabla f(x_k) + r_k \sum_{i=1}^{m} \frac{-1}{g_i(x_k)} \nabla g_i(x_k) \quad (58) \]
Now, in a similar argument, let
\[
\lambda_{i}^{(k)} = \frac{r_{k}}{-g_{i}(x_{k})}
\]  
(59)

Since \((-g_{i}(x^{*})) > 0\) in the feasible set, we have \(\lambda_{i}^{(k)} > 0\). Thus, we can form the Lagrangian as before and equation (59) generates the multipliers.

We next give an example to illustrate this. Lootsma solves the following problem

Minimize \(f(x) = x_{1}^3 - 6x_{1}^2 + 11x_{1} + x_{3}\)

subject to
\[
g_{1}(x) = x_{1}^2 + x_{2}^2 - x_{3}^2 \leq 0
\]
(60)
\[
g_{2}(x) = -x_{1}^2 - x_{2}^2 - x_{3}^2 - 4 \leq 0
\]
\[
g_{3}(x) = +x_{3} - 5 \leq 0
\]
\[
g_{4}(x) = -x_{1} \leq 0
\]
\[
g_{5}(x) = -x_{2} \leq 0
\]
\[
g_{6}(x) = -x_{3} \leq 0
\]

The optimum is \(x^{*} = (0, \sqrt{2}, \sqrt{2}, \sqrt{2})\) with \(g_{1}(x^{*}) = g_{2}(x^{*}) = g_{4}(x^{*}) = 0\). The theoretical multipliers are
\[
\lambda_{1} = \lambda_{2} = \sqrt{2}/8 \approx 0.1777766
\]
\[
\lambda_{3} = \lambda_{4} = \lambda_{5} = \lambda_{6} = 0
\]

For \(r_{k} = 10^{-6}\), the numerical multipliers are
\[
\lambda_{1}^{(k)} = \frac{r_{k}}{-g_{1}(x_{k})} = \frac{10^{-6}}{5.66046 \times 10^{-6}} = 0.176664
\]
\[
\lambda_{2}^{(k)} = \frac{r_{k}}{-g_{2}(x_{k})} = \frac{10^{-6}}{6.65324 \times 10^{-6}} = 0.15030
\]
\[
\lambda_{4}^{(k)} = \frac{r_{k}}{-g_{4}(x_{k})} = \frac{10^{-6}}{9.2007 \times 10^{-8}} = 10.8687
\]
\[
x_{k} = (9.207 \times 10^{-8}, 1.4121, 1.41422)
\]
The final example is the Penalty Function method.

Let

$$C(x, r) = f(x) + \frac{1}{r} P(x),$$

where

$$P(x) = \sum_{i=1}^{m} (\max (g_i(x), 0))^{1+\varepsilon} \quad \varepsilon > 0.$$  

Then at the optimum,

$$\nabla_x C(x_k, r_k) = 0 \text{ which implies } \nabla f(x) + \frac{1}{r} \nabla P(x) = 0 \quad (61)$$

where

$$\nabla P(x) = \sum \epsilon(1+\epsilon) [\max (g_i(x), 0)]^\varepsilon \nabla g_i(x)$$

If we let

$$\lambda_i^{(k)} = \frac{(1+\epsilon)}{r_k} \left[ \max (g_i(x_k), 0) \right]$$

then $$\lambda_i^{(k)} > 0$$, since we have an exterior point method.

Thus, (61) becomes $$\nabla f(x) + \Sigma \lambda_i \nabla g_i(x) = 0$$,

and, letting $$x_k$$ denote the optimum for $$r_k$$, we have

$$\lambda_i = \lim_{x_k \to \infty} \lambda_i^{(k)} = \lim_{x_k \to \infty} \frac{(1+\epsilon)}{r_k} \left[ \max (g_i(x_k), 0) \right]^\varepsilon$$

In this way we can obtain the $$\lambda_i$$'s.

In solving the Lootsma problem by the penalty method with $$\varepsilon = 1$$, we obtained the following results for $$r_k = 1.0 \times 10^{-4}$$:

$$x_k = (1.443 \times 10^{-16}, 1.414, \ldots , 1.414, \ldots)$$

$$g_1(x_k) = 0.879711 \times 10^{-6}$$; $$g_2(x_k) = 0.8797477 \times 10^{-6}$$

$$g_4(x_k) = 0.388414 \times 10^{-14}$$

$$\lambda_1^{(k)} = \frac{(1+\epsilon)}{r_k} \left[ \max (g_1(x_k), 0) \right]^\varepsilon = 2 \left[ \frac{0.879711 \times 10^{-6}}{1.0 \times 10^{-5}} \right] = 0.1759422$$

$$\lambda_2^{(k)} = \frac{(1+\epsilon)}{r_k} \left[ \max (g_2(x_k), 0) \right]^\varepsilon = 2 \left[ \frac{0.8797479 \times 10^{-6}}{1.0 \times 10^{-5}} \right] = 0.1759495580$$

49
\[ \lambda_4^{(k)} = \frac{(1+\varepsilon) \cdot \text{max.} (g_4, 0))}{f_k} = 2 \left\{ \frac{0.388414 \cdot 10^{-14}}{10^{-5}} \right\} = 0.7768 \cdot 10^{-11} \neq 11 \]

The last four constraints of problem (60) were transformed by the following equation and not included as penalties:

\[
\begin{align*}
  x_1 &= y_1^2 \\
  x_2 &= y_2^2 \\
  x_3 &= 5 \sin^2 (y(3))
\end{align*}
\]

When these transformations are used, \( \lambda_4^{(k)} \) cannot be obtained by the above formula.

**INITIAL POINT AND SCALING**

Two final topics in solution strategy depend a great deal on the specific problem: initial point and scaling of variables. If the constraints are numerous and complicated, then finding an initial feasible point can be an additional problem in itself.

Zoutendijk\(^4^7\) gives a "simple" trick for a transformation which replaces the original problem by an equivalent problem for which an initial feasible point can readily be found. Suppose we are given:

Minimize \( f(x) \)

subject to

\[ g_i(x) \leq 0 \quad i = 1, 2, \ldots, m \]

and no initial point \( x^0 \) can be found by inspection or engineering knowledge such that \( g_i(x^0) < 0 \) for \( i = 1, 2, \ldots, m \). We now form the following problem:

Minimize \( f(x) + \mu \xi \)

subject to

\[ g_i(x) - \rho_i \xi \leq 0 \]

\[ \rho_i \geq 0 \quad i = 1, 2, \ldots, m \]

where \( \mu \) is a large number, \( \xi \) an additional variable, and \( \rho_i = 1 \) if \( g_i(x^0) > 0 \) and \( \rho_i = 0 \) if \( g_i(x^0) < 0 \). If \( \xi \geq \max \left\{ g_i(x^0) \right\} \) for all \( i \) such that \( g_i(x^0) > 0 \), then the point \( (x^0, \xi) \) is feasible for the modified problem. In reference 17 Zoutendijk proves that, for \( \mu \) sufficiently large, the modified problem will
have the same solution as the original. Note that in the attempt to minimize 
\( f(x) + \mu \xi \), \( \xi \) is driven to zero, but to satisfy the constraints \( x \) must be 
simultaneously chosen feasible (for the original problem). This is the part of 
mathematical programming in which the user’s previous engineering and 
design experience pays off, since the better the initial point, the faster the 
routine will converge.

Proper scaling of variables is still an art. Most computer routines for 
solving mathematical programming problems are designed to follow steep 
curved valleys and sharp ridges in locating an optimum. For a one-shot 
problem, scaling the variables is not worth the time; however, if a code is to 
be written to solve a large class of similar problems, then scaling may be 
worthwhile in long-term savings of machine time. We give an example, 
Pierre which transforms a poorly shaped objective function into a “nice” 
bowl-shaped function which can quickly be minimized. Unfortunately, 
scaling is somewhat limited to unconstrained problems; in a constrained 
problem attempting to make the objective function easy to minimize, the 
scaling might destroy any useful properties the constraints enjoy.

Example:

If \( f(x_1, x_2) = x_1^2 + 10x_1x_2 + 100x_2^2 \), then \( f \) has a narrow valley and 
a minimum at \( x_1 = x_2 = 0 \). To transform \( f \) into a more desirable shape, we 
eliminate the cross product \( 10x_1x_2 \) by letting \( x_1 = z_1 - abz_2 \) and \( x_2 = bz_2 \). 
Then \( f \) becomes

\[
 f = z_1^2 + (-2ab + 10b)z_1z_2 + (-10ab^2 + a^2b^2 + 100b^2)z_2^2.
\]

Letting \( a = 5 \) makes \(-2ab + 10b = 0\) and \( b = (1/75)^{1/2} \) solves \(-50b^2 + 100b^2 + 25b^2 = 1\); thus, \( f = z_1^2 + z_2^2 \), which is easily minimized by almost any 
routine.

It is highly recommended, when the variables are bounded or the 
range of the variable is known, that the variable be normalized. For example, 
if the variable \( x_i \) lies in the range:

\[
 L_i \leq x_i \leq U_i,
\]

then an appropriate normalization is given by:

\[
 0 \leq y_i \leq 1.0
\]

where
Normalization is particularly useful in direct search algorithms for which we must arbitrarily choose a step size search increment, $\Delta$. It can readily be 1% of 1.0 (0.01), which allows each variable to be changed proportionally, or any other realistic choice depending on our knowledge of the problem.

In summary, scaling an objective function is generally not practical, and we must rely on the properties of the algorithms to find the optimum; however, normalization is recommended wherever possible.

**SUMMARY**

Mathematical programming is a broad subject with many varied applications. Not all topics in MP are applicable to engineering design. We have tried to delineate those areas which are useful and the corresponding capabilities at NELC. The computer codes described have proved reliable on a wide range of problems; however, MP is an expanding area, and the new algorithms constantly being developed could render some of these codes obsolete. The applications-oriented user should not accept this list as complete or give up if his particular problem does not match an available routine. Perhaps a literature search will yield the appropriate method. The high-speed digital computer has provided an impetus to develop algorithms to solve MP problems which previously were too large to be handled. It is recommended that a continued effort be maintained to keep NELC up to date in the area of solving MP problems numerically. Logically, this should be a function of either a Center-wide computer users’ group or of Computer Sciences Department. Other installations maintain a library of computer codes readily available to users; NELC should do the same.

In the applications area, mathematical programming has been a proved design tool. In reference 1, it was shown to be applicable to typical NELC problems. The second part of this report discussed some topics which were useful in obtaining actual solutions to design-related MP problems. The techniques have been used with routines available at NELC. We have tried to show that MP can be an aid to the design engineer and not his replacement. In fact to use MP effectively requires that the engineer be skillful in his field and be able to generate an accurate mathematical model of his design problem. We further recommend that a short course or continuing seminar be offered to NELC personnel to familiarize them with MP techniques. Such a course was given in-house in the fall of 1968 and was well received.
APPENDIX 1: APPLICATIONS OF INTEGER PROGRAMMING TO ENGINEERING DESIGN

In this appendix we report on the results of a literature search to determine the feasibility of using integer programming (IP) as a practical design aid in an ongoing NELC task - BAMS (Benchmarks for Applications of Microelectronics to Systems). The results were disappointing. Meaningful applications are still in the experimental stage, with results limited to relatively small test problems. The main hindrance to successful applications is not formulation difficulties, but the lack of reliable computer codes for solving the resulting IP problems. General-purpose IP codes are severely limited in the size problem they can solve (a maximum of 60 constraints and 60 variables at NELC), and they are sometimes unreliable. The majority of the applications of IP have been in the business world, and many algorithms for solving special IP problems have been developed; e.g., aircraft crew scheduling and warehouse placement. These methods rely on the special structure of the IP problem under consideration and have worked well. If the design engineer is lucky, his IP problem may fit one of these special methods (it is still an art to match the computer code to the posed IP problem); otherwise he must rely on the general IP codes. Here, application is ahead of theory.

We give a brief example which illustrates the computational difficulties of IP, a review of the state of the art of IP as related to BAMS, and finally a detailed example in which IP is used to solve a backboard wiring problem. Most of the IP work has been done in the linear case; i.e.,

Minimize \( f(x) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \)

subject to \( g_1(x) = a_{11} x_1 + a_{12} x_1 + \ldots + a_{1n} x_n \leq b_1 \)

\[
\vdots
\]

\( g_m(x) = a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m \)

\( x_i \geq 0 \) and \( x_i \) an integer

Thus, the integer linear programming (ILP) problem is just the LP problem with the additional constraint that the solution be integral. Gomory\(^49\) has developed an algorithm which theoretically solves the above problem in a finite number of steps \( N \); however, \( N \) can be a large number and hence impractical for some problems. Since this area of mathematics is so useful, many methods (some heuristic in nature) for solving special ILP problems have been reported which work well. One method, which appears obvious, is to solve the associated LP problem, then round to the nearest (in some sense) integer-valued vector and use that for the solution. Many times this
will work fine, if it is not too critical to have the optimum. If it is necessary
to have the optimum and know that it is the optimum, then other methods
must be used. The following example points out some difficulties which
can arise.

\[
\text{Minimize } z = f(x_1, x_2) = -x_1 - 4x_2
\]

subject to

\[
0 \leq x_1 \leq 4.55
\]

\[
0 \leq x_2 \leq 4.0
\]

\[
.5x_1 + x_2 \leq 5.2 \quad x_1, x_2 \text{ integers}
\]

The constraint set (denoted by +) looks as follows (fig. 7):

![Figure 7. Integer constraint set.](image)

When the associated LP problem is solved (by inspection), we obtain (1.3, 4.55). Rounding to the nearest vector with integer components yields (1,5), but (1,5) is not feasible. If we take the closest feasible point to (1.3, 4.55) in the Euclidean norm sense, then we have (1,4) for a solution, which yields \( z = f(1,4) = -17 \). However, another feasible point (2,4) yields \( z = f(2,4) = -18 \) and this point is the solution. As the number of variables and constraints increases, it becomes difficult to find a feasible point to the ILP near the LP optimum.

The field of digital systems design has made the most engineering use of IP, but again designers have had only limited success. Logic designers have used ILP as a theoretical tool and have solved small problems numerically. Muroga\(^5\) perhaps has the most recent application (see references 52 and 53 also). He discusses designing optimal networks of the "feedforward" type by ILP and defines a generalized gate called a threshold gate. His ILP formulation allows a wide choice of objective functions, depending on the application,
as well as the inclusion of any design constraints. If \( R \) is the number of gates in the design, then his associated ILP problem has \( R^2 \) variables. For any typical size in a logic design problem, the resulting ILP problem would be intractable. Presently, work in this area is of academic interest only. Also, with the abundance of off-the-shelf MSI and LSI components, little design is done at the gate level (except by those manufacturing the aforementioned items). Thus, the extra effort to formulate the logic design problem as an ILP does not appear to be cost-effective.

The next pertinent application area is that of actual component and circuit layout. Kodres\(^5\) develops the theory for solving the circuit layout problem, which he defines as follows:

"The circuit layout problem is viewed as a sequence of four subproblems.

1. The determination of standard replaceable modules.
2. The partitioning of circuits into groups subject to input-output restrictions.
3. The selection of replaceable modules.
4. The circuit placement and the interconnection problem."

Kodres uses graph theory, combinatorics, and integer programming to formulate the problem. The actual casting of parts 3 and 4 as an IP problem is partially in terms of some graph theoretical concepts and requires more background than we can present here. This paper points the way for future work in this area; again, the ideas are far ahead of practical methods for implementing them. Breuer in reference 55 poses part 4 as a single ILP problem, in straightforward terms, which we outline in the following paragraphs. Other applications are in coding theory and satellite communications network design; see references 56 - 58.

**BREUER'S PLACEMENT AND INTERCONNECTION IP FORMULATION**

The backboard wiring problem consists of three subproblems: the placement problem, the connection problem, and the routing and installation problem; each is dependent on the other two. We discuss the first two and pose them as a single ILP problem which, when solved, will simultaneously solve both problems.

**PLACEMENT PROBLEM**

Given \( B \) objects, connect each object to a subset of the remaining \((B-1)\) objects. The objects are constrained to lie on grid points which represent the backboard of a computer, or any digital system hookup. The object is to place all the modules so that the hook up wire is of minimal length.
CONNECTION PROBLEM

Given $S$ fixed objects which are to be made electrically common, connect the objects so that the total interconnection length is minimal.

INTEGER PROGRAMMING FORMULATION

Given $B$ objects to be placed at the intersections of the rectangular grid (fig. 8).

![Rectangular grid diagram]

Figure 8. Back plane grid.

Let $x_i$, $i = 0, 1, \ldots, n-1$ be the $x$ coordinate of the $i$th object

$y_i$, $i = 0, 1, \ldots, m-1$ be the $y$ coordinate of the $i$th object

and $B \leq mn$.

The interconnection distance between the $i$th and $j$th object is defined to be

$$d_{ij} = |x_i - x_j| + k |y_i - y_j|.$$ 

We note that no two objects can occupy the same spot at the same time.

Also given is a list of the desired connections. We wish to uniquely position the set of $B$ objects at the intersections and determine which objects should be directly connected together, in a manner such that total interconnection distance of those objects, directly connected, is minimal. The hard parts are making sure no two objects occupy the same spot and getting a linear relation for $d_{ij}$. 

56
The constraint that no two objects lie at the same point requires that if \( x_i = x_j \), then \( y_i \neq y_j \) for \( i \neq j \); or if \( y_i = y_j \), then \( x_i \neq x_j \) for \( i \neq j \).

We list the required constraints for the placement problem and then explain how they meet the conditions of the problem.

\[
\begin{align*}
\text{(A1)} & \quad x_i - x_j \leq n \delta_{ij} \\
\text{(A2)} & \quad x_i - x_j + 1 \leq n(1 - \delta_{ij}) \\
\text{(A3)} & \quad \delta_{ij} \leq 1 \quad \text{i.e.} \quad \delta_{ij} = 0 \text{ or } 1 \\
\text{(A4)} & \quad \alpha_{ij} \leq n \delta_{ij} \\
\text{(A5)} & \quad x_i - x_j \leq \alpha_{ij} \leq x_i - x_j + n(1 - \delta_{ij}) \\
\text{(A6)} & \quad \alpha_{ij} \leq n(1 - \delta_{ij}) \\
\text{(A7)} & \quad x_j - x_i \leq \alpha_{ij} \leq x_j - x_i + n \delta_{ij}
\end{align*}
\]

for all \( i > j, j = 1, 2, \ldots, B-1 \)

From inequalities (A1) and (A2) we find that if \( x_i > x_j \), then \( \delta_{ij} = 1 \) and \( \delta_{ij} = 0 \), if \( x_i > x_j \). Inequalities (A4)-(A7) give a representation for \( |x_i - x_j| \) as follows. Since (A4) and (A5) yield \( 0 < \alpha_{ij} = x_i - x_j \leq n - i \), if \( x_i > x_j \) and \( \alpha_{ij} = 0 \) otherwise. In a similar fashion \( 0 < \alpha_{ij} = x_j - x_i \), if \( x_j > x_i \); thus, we have \( \alpha_{ij} + \alpha_{ji} = |x_i - x_j| \).

Now let \( \alpha \) and \( \beta \) play analogous roles for \( y \); then \( \beta_{ij} + \beta_{ji} = |y_i - y_j| \). Thus, \( d_{ij} \) can be stated in terms of these auxiliary variables as

\[
d_{ij} = \alpha_{ij} + \alpha_{ji} + k(\beta_{ij} + \beta_{ji})
\]

Some additional constraints guarantee realizability.

\[
\begin{align*}
\text{(A8)} & \quad \alpha_{ij} + \alpha_{ji} + k(\beta_{ij} + \beta_{ji}) \geq 1 \\
\text{(A9)} & \quad x_i \leq n-1 \quad i = 1, 2, \ldots, B \\
\text{(A10)} & \quad y_i \leq m-1 \quad i = 1, 2, \ldots, B
\end{align*}
\]

Inequality (A8) guarantees that two objects do not occupy the same grid intersection. All \( x_i \) and \( y_i \) are nonnegative integers. Note that the above inequalities do not depend on how the objects are connected, but only guarantee that specific necessary conditions will be satisfied. Inequalities (A1) through (A10) represent the constraints for the placement problem. We now turn to the problem of optimally interconnecting the \( B \) objects.

Assume that there are \( C \) independent circuits where the \( j \)th circuit can be connected in \( P_j \) different acceptable ways. Let \( f_{ij}(d) \) be an expression for the total length of wire in the \( i \)th way of connecting the \( j \)th circuit. For example, if the sixth circuit consists of three objects \( (1, 2, 3) \) and the first
may have only one connection to it, then

\[ f_{16} = d_{21} + d_{32} \quad \text{and} \quad f_{26} = d_{31} + d_{32} \]

The two final constraints which relate to the connection problem are:

\[ V_j \geq f_{ij}(d) + (\gamma_{ij} - 1)Q \quad \text{(A11)} \]

and

\[ \sum_{i=1}^{P_i} \gamma_{ij} = 1 \quad \text{for} \quad j = 1 \ldots C \quad \text{(A12)} \]

Equation (A12) implies that each circuit is connected and \( \gamma_{ij} \) is 0 or 1.

In (A11) \( Q = \max_i f_{ij}(d) \) for \( i = 1, 2, \ldots, P_j; j = 1, 2, \ldots, C \). Then the resulting objective function is

\[ \min Z = \sum_{j=1}^{C} V_j \]

The objective function and inequalities (A1) - (A11) form the ILP representing the combined placement and connection problem. Although the formulation seems straightforward, the resulting ILP problem can be large, even for small values of \( B \). Breuer gives the following relations between the number of variables \( I \), the number of constraints \( W \), and \( B \) for the placement problem: i.e., constraints (A1) - (A10).

\[ I(B) = (B/2)(19B-15) \]
\[ W(B) = B(3B-1) \]

Thus, for \( B=5 \) (which could be manually positioned quickly and most likely optimally), we have \( I(5) = 200, W(5) = 70 \); for \( B=10 \) (still not too large), \( I(10) = 875, W(10) = 290 \). These ILP problems are very large and are beyond the capabilities of today’s methods.

It seems that, unless newer formulation techniques are developed which lead to smaller IP problems, or the computational capabilities solving IP increase rapidly, this approach will remain a theoretical tool with no practical applications. The following quote (Glover, p. 1) sums up the current state of affairs in integer programming: “Since its inception integer linear programming has, paradoxically, been a source of both promise and disappointment. Promise because there are manifold and compelling opportunities for its application; disappointment because it has made only the most dubious progress in spite of these opportunities.”
APPENDIX 2: USER INFORMATION

H1 – H2 OPTALG

CATALOG IDENTIFICATION:
H1 – H2 OPTALG

PROGRAMMER:
F. S. Hillier, Stanford University, adapted for NELC by D. Klamer, Decision and Control Technology Division.

PURPOSE:
An algorithm for solving the pure integer linear programming problem.

Maximize \( x_0 = \sum_{j=1}^{n} c_j x_j \)

subject to

(i) \( \sum_{j=1}^{n} a_{ij} x_j \leq b_j \) \( (i=1,2,\ldots,m) \)  

(ii) \( x_j \geq 0 \) \( (j=1,2,\ldots,n) \)  

(iii) \( x_j \) is an integer \( (j=1,\ldots,n) \)

RESTRICTIONS AND LIMITATIONS:
The dimensions of the constraint matrix \( A(I,J) = a_{ij} \) have to be less than or equal to 61 \( \times \) 61; i.e., \( m \leq 61 \) and \( n \leq 61 \).

LANGUAGE:
FORTRAN IV

COMPUTER CONFIGURATION:
Go step REGION = 336k
IBM 360/65

METHOD:
An initial noninteger solution must be obtained from the related linear programming problem (i.e., \( x_j \) is not necessarily an integer) as well as the resulting basis inverse. To accomplish this, we have chosen the linear programming routine MPS/360. Data are taken directly from MPS/360 and input directly into Hillier's program without user intervention, in one multistep computer run.

The advantage of using this modified version of Hillier's program is the time saved from obtaining the basis inverse and optimal solution; the time
spent for punching these input cards is also saved. The data needed are exactly
the same as the first three card groups of Hillier's program. These are:

Card group 1 Any alphanumeric characters to identify the problem in 20A4 Format
Card group 2 m, n, KL in 315 Format where A(I,J) is of size mxm
and KL = 1
Card group 3 the arrays A, b, c in 15 F5.0 Format. A(I,J) is the
constraint matrix, B(I) is the right-hand side, and
C(J) is the objective function. The A is read in one
row at a time.

FORTRAN DECK 1 Setup data for MPS/360.
DATA Constraint matrix, right-hand side, and objective
function.
FORTRAN DECK 2 Using READCOM from MPS/360 obtains basis inverse,
JPM(I), optimal solution, and starting integer solution.
Stores information on disk.
MPS/360 Computes basis inverse and optimal noninteger solution.
HILLIER'S PROGRAM Computes the optimal integer solution.
FORTRAN DECK 1

This deck sets up the data for MPS/360, since the format for MPS/360 is long and cumbersome. The constraint matrix $A(I,J)$ is normalized, as is the right-hand side $B(I)$; this is done one row at a time. The subroutine XPUNCH places the data into a disk file in proper format for MPS/360, from which MPS/360 reads the data. A printout is given of the data that are placed on disk. (Note: This data set is placed into a disk file called FT01F001. The data are in normalized form, and, since MPS/360 is designed to find the minimum, the signs of the cost coefficients (objective function) are changed.) The constraint matrix $A(I,J)$, the right-hand side $B(I)$, and the cost coefficients $C(J)$ are also stored on the disk file called FT02F001.

FORTRAN DECK 2 (DATAHILL)

This deck is a temporary update that is concatenated onto MPS/360, under the name DATAHILL. It uses READCOM, which is a subroutine designed to augment MPS/360 with procedures written in FORTRAN language. DATAHILL retrieves from MPS/360 the basis inverse, the order of the basic variables, the optimal noninteger solution, and a starting feasible solution to the integer linear programming problem. These data are then added to the data from the first FORTRAN deck on the disk file FT02F001.

The starting feasible solution is a lower bound on the value of the objective function. To obtain this feasible solution, we have chosen to do the following: If the cost coefficient is positive, round the corresponding variable of the optimal noninteger solution down to the next largest integer. If the cost coefficient is negative, round up to the next smallest integer. (Note: The cost coefficients are placed into the disk file called FT03F001 in the first FORTRAN deck and are read by DATAHILL; the constraint matrix and right-hand side are also passed.) This rounded solution is checked for feasibility. If the solution is feasible, then proceed to the next step. If the solution is not feasible, then try a rounding procedure to satisfy the constraint violated. (User may set the maximum number of iterations or changes in the solution.)

In order for Hillier's program to be executed, a feasible integer solution must be found; if such a solution is not found, then Hillier's program is skipped and all the data are punched out on cards. This information includes the name, constraint matrix $A(I,J)$, right-hand side $B(I)$, cost coefficients $C(J)$, the basis inverse, the order of the basic variables, and the optimal noninteger solution. The user may then supply his own starting integer solution and run the problem directly from LLOAD, using the punched data produced in the last step. (See Part II.)

If a feasible integer solution is obtained and the user wishes to have the above data also, then either of two methods may be used. First, there
are comment cards in the program to punch out each of the groups of data. All that is required is to remove the "C" from the cards in the program corresponding to which groups are to be punched out. The second method requires two changes in the JCL cards. See the JCL listing at the beginning of the program.

**MPS/360**

MPS/360 is an IBM supplied application program, “Mathematical Programming System/360.” MPS/360 obtains an optimal solution (non-integer) from the related linear programming problem and finds the inverse of the basis.

**HILLIER’S PROGRAM**

Hillier’s program resides on LLOAD (a partitioned data set on NELC’s 360/65 disk storage) under the name OPTALG.
This section covers the necessary input for Hillier's program when run without the adapted program to generate the data. The following is the necessary input:

Card input 1 Any alphanumeric characters to identify the problem in 20A4 Format; e.g., "Thompson Number 8"

Card group 2 m, n, KL in 315 Format (m \leq 61, n \leq 61) where m is the number of rows of the constraint matrix A(I,J) and n is the number of columns

\[
KL = \begin{cases} 
1 & \text{if the basis inverse is in normalized form} \\
0 & \text{otherwise} 
\end{cases}
\]

Card group 3 This group of cards contains the arrays A, b, c: the Format is 15F5.0. A(I,J) is the constraint matrix, B(I) is the right-hand side, and C(J) is the row matrix of the cost coefficients (or objective function). The A array is read in one row at a time. (For example, if m=2, n=16, the cards would be:

\[
a_{1,j}(j=1,2,\ldots,15) \text{ in the first 75 columns;}
\]

\[
a_{1,16} \text{ in the first 5 columns;}
\]

\[
a_{2,j}(j=1,2,\ldots,15) \text{ in the first 75 columns;}
\]

\[
2_{2,16} \text{ in the first 5 columns;}
\]

\[
b_1, b_2 \text{ in the first 10 columns;}
\]

\[
c_j(j=1,2,\ldots,15) \text{ in the first 75 columns;}
\]

\[
c_{16} \text{ in the first 5 columns.}
\]

Card Group 4 This group of cards contains the basis inverse in 6F13.5 Format. The rows are read in sequentially. In the example listing,

\[
BB_{1,1}, BB_{1,2}, BB_{1,3}, \ldots, BB_{1,6}
\]

\[
BB_{1,7}, BB_{1,8}, BB_{2,1}, \ldots, BB_{2,4}
\]

\[
BB_{2,5}, BB_{2,6}, BB_{2,7}, \ldots, BB_{3,2}
\]

Card Group 5 JPM(i), i=1,2,\ldots, in 15I4 Format, where JPM(i) is the index of the i \text{th} basic variable (including slack variables) from the simplex code.

Card Group 6 The optimal solution to the related linear programming problem, x(i), in 6F13.5 Format.
Card Group 7  A starting optimal solution to the Integer Linear Programming problem, \( XF(j) \), in 6F13.5 Format.

See the sample problem following. The correspondence of cards with the above card groups is as follows:

<table>
<thead>
<tr>
<th>Card Group</th>
<th>Card in Sample Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3-12</td>
</tr>
<tr>
<td>4</td>
<td>13-23</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>25-26</td>
</tr>
<tr>
<td>7</td>
<td>27-28</td>
</tr>
</tbody>
</table>

(Note: The first three card groups are the same as the three card groups for the adapted version of Hillier's program.)
SAMPLE PROBLEM

//PRELIM JOB 1055369,6202044,F,D,5,5,10000,DIKAMER,
// MSGLEVEL=1,CLASS=L
// MESSAGE 000
//S1 EXEC FORTRANL,TIME=1,REGION.GO=64K
//FORTSYSIN DD *
DOUBLE PRECISION IA(61),BUFFER(61)
DIMENSION A(61), B(61), C(61), NAME(20), SUM(61)

READ(5,574) (NAME(I),I=1,20)
WRITE(6,575) (NAME(I),I=1,20)
FORMAT(315)
READ(5,201) M,N,KL
WRITE(2,201) M,N,KL
WRITE(6,503)
DO 183 I=1,M
READ (5,2001) (IA(J),J=1,N)
WRITE(2,2001) (IA(J),J=1,N)
WRITE(3,2001) (IA(J),J=1,N)
CALL CORE(BUFFER,488)
WRITE(8,2002) (IA(J),J=1,N)
CALL CORE(BUFFER,488)
READ(8,2003) (A(I,J),J=1,N)
WRITE(6,500) (A(I,J),J=1,N)
CONINUE

WRITE(6,504)
READ (5,2001) (IA(J),J=1,M)
WRITE(2,2001) (IA(J),J=1,M)
WRITE(3,2001) (IA(J),J=1,M)
CALL CORE(BUFFER,488)
WRITE(8,2002) (IA(J),J=1,M)
CALL CORE(BUFFER,488)
READ(8,2003) (B(J),J=1,M)
WRITE(6,500) (B(J),J=1,M)
WRITE(6,505)
READ (5,2001) (IA(J),J=1,N)
WRITE(2,2001) (IA(J),J=1,N)
WRITE(3,2001) (IA(J),J=1,N)
CALL CORE(BUFFER,488)
WRITE(8,2002) (IA(J),J=1,N)
CALL CORE(BUFFER,488)
READ(8,2063) (C(J),J=1,N)
WRITE(6,500) (C(J),J=1,N)

2001 FORMAT(15A5)
2002 FORMAT(6A5)
2003 FORMAT(61F5.0)
570 FORMAT(1H,15F8.2)
501 FORMAT(1H,15F8.4)
555 FORMAT(1H,10F12.5)
567 FORMAT(1H,15F8.4)
FORMAT(28H CONSTRAINT MATRIX A(I, J) IS)

FORMAT(24H RIGHT HAND SIDE B(I) IS)

FORMAT(27H COST COEFFICIENTS C(J) ARE)

FORMAT(43H THE NORMALIZED CONSTRAINT MATRIX A(I, J) IS)

FORMAT(3QH THE NORMALIZED RIGHT HAND SIDE B(I) IS)

K=0
DO 1111 I=1,M
K=K+1
SUM(K)=C(J)
D:2 2222 J=1,N
SUM(K)=SUM(K)+A(I, J)**2
2222 CONTINUE
SUM(K)=SQRT(SUM(K))
1111 CONTINUE
DO 3333 I=1,M
DO 4444 J=1,N
A(I, J)=A(I, J)/SUM(I)
4444 CONTINUE
3333 CONTINUE
WRITE(6, 506)
DO 5555 I=1,M
5555 WRITE(6, 567)(A(I, J), J=1,N)
DO 7777 I=1,M
7777 B(I)=B(I)/SUM(I)
WRITE(6, 507)
WRITE(6, 5011)(B(I), I=1,M)
508 FORMAT('1 THE FOLLOWING DATA HAS BEEN STORED ON DISK *)
WRITE(6, 508)
CALL XPUNCH (A, B, C, M, N)
STOP
END

SUBROUTINE XPUNCH (A, B, C, M, N)
DIMENSION A(61, 61), B(61), C(61)
WRITE (1, 1111)
PRINT 110
WRITE (1, 1113)
PRINT 112
WRITE (1, 1151)
PRINT 114
DO 1 K=1,M
KROW10 = KROW + 10
WRITE (1, 107) KROW10
PRINT 100, KROW10
1 CONTINUE
WRITE (1, 109)
PRINT 109
M1 = M+1
DO 10 J=1,N
K = K1
10 I = C
CONTINUE
K1 = K
K2 = K+1
L = J+10
I = I + 1
IF (K2 .GE. M1 + 10) GO TO 11
WRITE (1,100) L,K1,A(I,J),K2,A(I+1,J)
PRINT 100,L,K1,A(I,J),K2,A(I+1,J)
K = K+2
I = I + 1
GO TO 5
11 IF (K .EQ. M1 +10) GO TO 12
C(J) = - C(J)
WRITE (1,101) L,K1,A(I,J),C(J)
PRINT 101,L,K1,A(I,J),C(J)
GO TO 10
12 C(J) = - C(J)
WRITE (1,102) L,C(J)
PRINT 102,L,C(J)
CONTINUE
WRITE (1,105)
PRINT 105
DO 20 I = 1,M
IR = I + 10
WRITE (1,103) IR,B(I)
PRINT 103,IR,B(I)
CONTINUE
WRITE (1,117)
PRINT 117
100 FORMAT(4X,'C',12,7X,'R',12,7X,F12.5,3X,'R',12,7X,F12.5)
101 FORMAT(4X,'C',12,7X,'R',12,7X,F12.5,3X,'C',9X,F12.5)
102 FORMAT(4X,'C',12,7X,'C',9X,F12.5)
103 FORMAT(4X,'C',12,7X,'C',9X,F12.5)
104 FORMAT(4X,'RHS')
105 FORMAT(4X,'RHS')
106 FORMAT(4X,'L',I2)
107 FORMAT(4X,'L',I2)
108 FORMAT(4X,'COLUMNS')
109 FORMAT(4X,'COLUMNS')
110 FORMAT(4X,'NAME OPTBAS')
111 FORMAT(4X,'NAME OPTBAS')
112 FORMAT(4X,'ROWS')
113 FORMAT(4X,'ROWS')
114 FORMAT(4X,'N O')
115 FORMAT(4X,'N O')
116 FORMAT(4X,'ENDATA')
117 FORMAT(4X,'ENDATA')
RETURN
END
//STEPONE EXEC FORTGCL
//FORT.SYSLIN DD DISP=(NEW, PASS)
//FORT.SYSIN DD *

INTEGER DIFF, FILE
INTEGER*2 COMP(61), R
DIMENSION BASE(61, 61), NROW(61), NCOLM(61), JPM(61), IKOW(61),
1 BUFFFR(61), CC(61), XF(61)
DIMENSION A(61, 61), B(61),
1 IDX1(61), IDX2(61), XFMIN(61)
REAL*8 NAMEXLIST(30), OUT1(62), OUT2(3721), OUT:61
DATA BASE /3721*0.0/
LK = 0
FILE = 4

C TO OBTAIN THE NUMBER OF ROWS AND COLUMNS OF THE C
CONSTRAINT MATRIX (INCLUDING THE OBJECTIVE FUNCTION)
C
CALL ARRAY(FILE, INDIC, NAME)
CALL VECTOR(FILE, INDIC, XLIST)
MROWS = IFIX(SNGL(XLIST(0)))
NCOLMN = IFIX(SNGL(XLIST(10)))
MNWS1 = MROWS - 1
MN = MNWS1 * NCOLMN

C TO OBTAIN THE BASIS
C
CALL ARRAY(FILE, INDIC, NAME)
DO 10 J = 1, MN
CALL VECTOR(FILE, INDIC, XLIST)
IF(J.GT.MROWS) GO TO 12
OUT1(J) = XLIST(J)
10 CONTINUE
12 IF(MOD(J, MROWS) .EQ. 0) GO TO 11

68
LK = LK + 1
OUT2(LK) = XLIST(2)

IF(INDIC-1) 20, 20, 10
CONTINUE
20 CONTINUE

C TO OBTAIN THE OPTIMAL SOLUTION

CALL ARRAY(FILE, INDIC, NAME)
CALL VECTOR(FILE, INDIC, XLIST)
CALL ARRAY(FILE, INDIC, NAME)
CALL VECTOR(FILE, INDIC, XLIST)
CALL ARRAY(FILE, INDIC, NAME)
DO 201 J=1, NCOLMN
CALL VECTOR(FILE, INDIC, XLIST)
OUT(J) = XLIST(3)
IF(INDIC-1) 200, 200, 201
201 CONTINUE

C TO PRINT OUT THE OPTIMAL SOLUTION

101 FORMAT(6F13.5)
102 FORMAT(1X, 6F13.5)
103 FORMAT(' THE OPTIMAL NONINTEGER SOLUTION IS ')
WRITE(6, 103)
WRITE(6, 102) (OUT(J), J=1, NCOLMN)

C TO DETERMINE WHICH VARIABLES ARE ACTIVE

CALL CORE(BUFFER, 240)
WRITE(8, 104) (OUT1(J), J=1, MROWS)
104 FORMAT(61A4)
CALL CORE(BUFFER, 2, C)
READ(8, 105) (COMP(J), IROW(J), J=1, MROWS)
105 FORMAT(61(A1, 12, 1X))
DATA R/R 'R /C/ C '
NOROW = 0
NCOLM = 0
DO 500 J=1, MROWS
IF(COMP(J).EQ.R) NOROW = NOROW + 1
IF(COMP(J).EQ.C) NCOLM = NCOLM + 1
500 IF((NOROW.EQ.0).AND.(NCOLM.EQ.0)) GOTO 100
IF(NOROW.EQ.0) GOTO 600
IF(NCOLM.EQ.0) GOTO 650

C TO COMPUTE THE JTH BASIC VARIABLE USING MPS/360 INFORMAT

DO 56 J=1, NOROW
56 NOROW(J) = IROW(J)
I = 0
NTOT = NOROW + NCOLM

69
Nie = NUROW + 1
DO 57 J=1,NTOT
   L = L+1
57   NCOLM(I) = IROW(J)
   GOTO 150
C
C BASIS INVERSE IS BASIS
C
600 DO 601 J=1,NCOLM
601 JPM(J) = J
   WRITE(6,105)
   DO 602 I=1,MROWS1
602 WRITE(6,101) ((OUT2(I+(J-1)*MROWS1),I=1,NCOLMN),J=1,NCOLMN),I=1,MRCWS1)
C
C IF YOU WANT THE BASIS INVERSE PINCHED OUT
C (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C REMOVE THE 'C' FROM THE FOLLOWING CARD.
C
WRITE(7,101) ((OUT2(I+(J-1)*MROWS1),J=1,NCOLMN),I=1,MRCWS1)
C
GOTO 202
C
C BASIS INVERSE IS IDENTITY MATRIX
C
650 DO 651 J=1,NOROW
651 BASE(J,J) = 1.0
   WRITE(6,106)
   WRITE(6,101) ((BASE(I,J),J=1,MROWS1),I=1,NCOLMN),I=1,MROWS1)
   WRITE(2,101) ((BASE(I,J),J=1,MROWS1),I=1,NCOLMN),I=1,MROWS1)
C
C IF YOU WANT THE BASIS INVERSE PINCHED OUT
C (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C REMOVE THE 'C' FROM THE FOLLOWING CARD.
C
WRITE(7,101) ((BASE(I,J),J=1,MROWS1),I=1,MCWS1)
C
GOTO 202
150 CONTINUE
   L=1
   KL=1
   NORM1 = NOROW - 1
   IF(NOROW(1)-11) 100,30,22
30   JPM(1) = NOROW(1) + NCOLMN - 10
   L = L+1
   GOTO 47
22   DIFF = NOROW(1) - 11
   DO 46 K=1,DIFF
   JPM(L) = NCOLM(KL) - 10
   KL = KL+1
   L = L+1
46   CONTINUE
   JPM(L) = NOROW(1) + NCOLMN - 10
   L = L + 1
47   CONTINUE
   DO 50 I=1,NORM1
DIFF = NROW(I+1) - (NROW(I)+1)
IF (DIFF) 100,35,40
35 JPM(L) = NROW(I+1) + NCOLMN - 10
L = L+1
GOTO 50
40 DO 45 K=1,DIFF
   JPM(L) = NCOLM(KL) - 10
   KL = KL+1
   L = L+1
45 CONTINUE
   JPM(L) = NROW(I+1) + NCOLMN - 10
   L = L+1
50 CONTINUE
   DIFF = 10 + NTOT - NROW(NOROW)
   IF (DIFF) 100,55,60
   JPM(NTOT) = NROW(NOROW) + NCOLMN - 10
55 GOTO 65
60 DO 65 K=1,DIFF
   JPM(L) = NCOLM(KL) - 10
   KL = KL+1
   L = L+1
65 CONTINUE
C
C TO OBTAIN THE BASIS INVERSE
C
J1 = 0
J2 = 1
DO 300 JK=1,NTOT
   IF(JPM(JK).GT.NCOLMN) BASE(J2,JK) = 1.0
   IF(JPM(JK).LE.NCOLMN) GOTO 310
   J2 = J2 + 1
300 GOTO 300
310 DO 311 J=1,MROWS1
   J1 = J1 + 1
311 BASE(J,J1) = OUT2(J1)
300 CONTINUE
C
C THE FOLLOWING IS FOR OUTPUT.
C
WRITE(6,105)
WRITE(6,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
WRITE(2,101) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
C IF YOU WANT THE BASIS INVERSE PUNCHED OUT
C (THERE ARE THREE (3) CARDS FOR THE BASIS INVERSE)
C REMOVE THE C* FROM THE FOLLOWING CARD.
C
WRITE(7,104) ((BASE(I,J),J=1,MROWS1),I=1,MROWS1)
C
200 CONTINUE
WRITE(6,107)
WRITE(6,108) (JPM(JK), JK=1,NTOT)
WRITE(2,108) (JPM(JK), JK=1,NTOT)
C
C IF YOU WANT THE ORDER OF THE ITH BASIC VARIABLE, JPM(I),
PUNCHED OUT
C REMOVE THE 'C' FROM THE FOLLOWING CARD.
C WRITE(7,108) (JPM(JK), JK=1,NTOT)
C WRITE(2,101) (OUT(J),J=1,NCOLMN)
C IF YOU WANT THE OPTIMA: INTEGERT SOLUTION PUNCHED OUT
C REMOVE THE 'C' FROM THE FOLLOWING CARD.
C WRITE(7,101) (OUT(J),J=1,NCOLMN)
C IF YOU WANT TO SUPPLY YOUR OWN FEASIBLE INTEGER SOLUTION
C REMOVE THE 'C' FROM THE FOLLOWING CARD.
C GOTO 151
C TO OBTAIN THE STARTING INTEGER SOLUTION TO THE ILP
C DO 110 I=1,MROWS1
READ( 3,111) (A(I,J),J=1,NCOLMN)
110 CONTINUE
READ( 3,111) (B(I),I=1,MROWS1)
READ(3,111)(CC(J),J=1,NCOLMN)
111 FORMAT(15F5,C)
DO 1000 I=1,NCOLMN
XF(I) = SNGL(OUT(I))
IF(CC(J)) 1001,1000,1000
1001 IF (XF(I).EQ.AINT(XF(I))) GOTO 1000
XF(I) = XF(I) + 1.
1000 CONTINUE
DO 1020 I=1,NCOLMN
1020 XF(I) = AINT(XF(I))
IEST = 0
C SET THE MAXIMUM NUMBER OF ITERATIONS.
C IF(10 -IEST) 1100,1C22,1022
1022 IEST = IEST + 1
C CHECK TO SEE IF THE POINT IS FEASIBLE.
C DO 1030 I=1,MROWS:
SUM = 0.0
DO 1025 J=1,NCOLMN
SUM = SUM + A(I,J) * XF(J)
1025 CONTINUE
IF(B(I)-SUM) 1026,1030,1030
1026 ITER = I
C IF POINT IS NOT FEASIBLE, WRITE THE CONSTRAINT VIOLATED AND
THE POINT THAT VIOLATED THE CONSTRAINT.
C
WRITE(6,1027) I,B(I),SUM,(XF(I,J),I,J=1,NCOLMN)
1027 FORMAT(10X,10H B(I2,2I1),SUM /
15X, 2F12.2 // 14H THE POINT IS /4(15F8.2/)///////)
GOTO 1035
1030 CONTINUE
GOTO 1120
1035 L = 0
N1 = 0
C
C FIND ALL OF THE NONZERO VALUES OF THE VARIABLES.
C
IDX1 STORES ALL OF THE ZERO VALUED VARIABLES.
C
IDXF STORES ALL OF THE NONZERO VALUED VARIABLES.
C
DO 1040 J=1,NCOLMN
IF( XF(J) ) 1036,1037,1038
1036 XF(J) = 0.0
1037 L = L+1
IDX1(L) = J
GOTO 1040
1038 N1 = N1 + 1
XFMIN(N1) = XF(J)
IDXF(N1) = J
1040 CONTINUE
IF(L.LT.L) GOTO 1042
C C TACK ON TO THE END OF THE INDIES OF THE NONZERO VARIABLES THE
C INDEX OF THE ZERO VARIABLES.
C
DO 1041 I=1,L
IDXF(N1+I) = IDX1(I)
C
C IF THERE IS AT LEAST ONE NONZERO VARIABLE,
C THEN ARRANGE THE NONZERO VARIABLES FROM
C MINIMUM TO MAXIMUM VALUE.
C
IDX2 IS THE INDEX OF THE REARRANGED VARIABLES.
C
1042 IF(N1.GT.0) GOTO 1045
DO 1044 I=1,L
1044 XFMIN(I)=0.0
GOTO 1046
1045 CALL SORTI(XFMIN,IDX2,1,N1)
1046 IF(N1.EQ.NCOLMN)GOTO 1048
N1P1 = N1 + 1
DO 1047 I=N1P1,NCOLMN
1047 IDX2(I) = IDXF(I)
C
C ROUNING PROCEDURE.
C
TAKE THE SMALLEST NONZERO VARIABLE,
C
IF THE CORRESPONDING CONSTRAINT COEIFFICIENTS IS
C
POSITIVE -- ROUND DOWN
C
NEGATIVE -- ROUND UP.
C
DO 1060 KL=1,NCOL\*MN
   IV = IDXF(IDX2(KL))
   LCK = 0
1049 IF( A(ITER,IV) ) 1050,1060,1051
1050 XF(IV) = XF(IV) + 1.0
   GOTO 1055
1051 XF(IV) = XF(IV) - 1.0
   IF(XF(IV),LT,0.0) XF(IV) = 0.0
1055 SUM = 0.0
   DO 1056 I=1,NCOLMN
   SUM = SUM + A(ITER,I) * XF(I)
1056 CONTINUE
C
C IF THIS DOES NOT SATISFY THE CONSTRAINT,
C THEN TAKE THE NEXT SMALLEST VARIABLE
C AND REPEAT THE PROCEDURE.
C
C IF THE CONSTRAINT IS SATISFIED, THEN USE THIS POINT TO
C CHECK ALL OF THE OTHER CONSTRAINTS.
C
C IF(B(ITER) - SUM) 1057,1021,1021
1057 LCK = LCK + 1
C
C SET THE MAXIMUM NUMBER OF CHANGES FOR ONE VARIABLE FOR ONE
ITERATION.
C CHANGE THIS IF THERE IS AN OSCILATION BACK AND FORTH BETWEEN
C TWO POINTS.
C
IF(LCK.LT.1) GOTO 1049
1060 CONTINUE
GOTO 1021
1100 CONTINUE
WRITE(6,1105) ITEST
1105 FORMAT(47H NO FEASIBLE STARTING INTEGER SOLUTION HAS BEEN
1 26H FOUND AT THIS POINT AFTER 15,6H TRYS.)
   GOTO 1150
1120 CONTINUE
   OBJ = 0.0
   DO 1122 I=1,NCOLMN
      OBJ = OBJ + XF(I)*CC(I)
   WRITE(6,1125) (XF(I),I=1,NCOLMN)
1125 FORMAT(53H A FEASIBLE STARTING INTEGER SOLUTION HAS BEEN
1 FOUND. /
14(15F8.7/))
   WRITE(6,1126) OBJ
1126 FORMAT( 4OH THE VALUE OF THE OBJECTIVE FUNCTION IS
1 F12.4 )
   WRITE(2,101) (XF(J),J=1,NCOLMN)
C
C IF YOU WANT THE OPTIMAL 'FEASIBLE' INTEGER SOLUTION PUNCHED OUT

74
REMOVE THE 'C' FROM THE FOLLOWING CARD.

WRITE(7,101) (XF(J),J=1,NCOLMN)

GOTO 151

100 WRITE(6,106)

106 FORMAT(* AN ERROR HAS OCCURED*)

107 FORMAT(* JPM(I) IN THE ORDER THEY SHOULD OCCUR AND IS THE
INDEX OF I THE ITH BASIC VARIABLE*)

108 FORMAT(1514)

109 FORMAT(//// THE BASIS INVERSE IS ////)

1150 CONTINUE

I = 10

WRITE(10,1151) I

1151 CONTINUE

I = 0

WRITE(10,1151) I

200 RETURN

END

SUBROUTINE SORTI(A,I0,II,JJ)

DIMENSION A(1),IU(30),IL(30),ID(1)

INTEGER T1,T2

M=1

I=II

J=JJ

DO 1 IS=I,J

1 ID(IS)=IS

5 IF(I .GE. J) GOTO 70

10 K=1

IJ=(J+I)/2

I=A(IJ)

T1=ID(IJ)

IF(A(I) .LE. T) GOTO 20

A(IJ)=A(I)

ID(IJ)=ID(I)

A(I)=T

ID(I)=T1

T=A(IJ)

T1=ID(IJ)

20 L=J

IF(A(J) .GE. T) GOTO 40

A(IJ)=A(J)

ID(IJ)=ID(J)

A(J)=T

ID(J)=T1

T=A(IJ)

T1=ID(IJ)

IF(A(I) .LE. T) GOTO 40

A(IJ)=A(I)

ID(IJ)=ID(I)

A(I)=T

ID(I)=T1
T = A(IJ)
T' = ID(IJ)
GOTO 40

30
A(L) = A(K)
ID(L) = ID(K)
A(K) = T
ID(K) = T'

40
L = L - 1
IF(A(L) .GT. T) GOTO 40
T = A(L)
T' = ID(L)

50
K = K + 1
IF(A(K) .LT. T) GOTO 50
IF(K .LE. L) GOTO 30
IF(L - 1 .LE. J - K) GOTO 60

60
IL(M) = K
IU(M) = J
M = M + 1
GOTO 80

70
M = M - 1
IF(M .EQ. 0) RETURN
I = IL(M)
J = IU(M)

80
IF(J - I .GE. 11) GOTO 10
IF(I .EQ. 11) GOTO 5
I = I + 1

90
IF(I .EQ. J) GOTO 70
T = A(I + 1)
T' = ID(I + 1)
IF(A(I) .LE. T) GOTO 90
K = I
ID(K) = ID(I)

100
A(K + 1) = A(K)
ID(K + 1) = ID(K)
K = K - 1
IF(T .LT. A(K)) GOTO 100
A(K + 1) = T
ID(K + 1) = T'
GOTO 90
END

//LKED SYSLIB DD DSN=LOAD,DISP=(SHR,KEEP)
// DD DSN=SYS1.FORTLIB,DISP=SHR
//LKED SYSLMOD DD DSN=EMCCAL,DISP=(NEW,PASS),UNIT=SYSDA,
SPACE=(CYL,(1,1,10)),DCB=(DSORG=PO,RECFM=U,BLKSIZE=3625)
/LKED.SY.IN DD *
INSE2,READCOMM
ENTRY MAIN
NAME DATAHILL(R)

// CPC EXEC PGM=COMPILER
// STEPLIB DD DISP=(SHR,KEEP),DSNAME=LLCAD
// SCRATCH1 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
// SCRATCH2 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
// SCRATCH3 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
// SCRATCH4 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
// SYSLCP DD UNIT=SYSODA,SPACE=(TRK,(2,1)),DISP=(NEW,PASS)
// SYSPRINT DD SYSOUT=A
// SYSPUNCH DD SYSOUT A
// SYSIN DD *  "PROGRAM ('IND')"
  "INITIAL"
  "MOVE(XDATA,'OPTBAS')"
  "MOVE(XPBNMAME,'MYFILE')"
  "MOVE(XOBJ,'O')"
  "MOVE(XRHS,'CONST')"
  "ASSIGN ('COMMFMT','FT04F001','COM')"
  "PREPOUT ('COMMFMT')"
  "CONVERT('SUMMARY')"
  "BCDOUT"
  "SETUP (1)"
  "PRIMAL"
  "SOLUTION"
  "TRANCOL('ENTIRE','INVERSE')"
  "TRANCOL('ENTIRE')"
  "TRANCOL('FILE','COMMFMT','ENTIRE')"
  "SOLUTION('FILE','COMMFMT')"
  "DATAHILL"
  "EXIT"
  "PEND"

// EXEC EXEC PGM=EXECUTOR,COND=(C,NE,CPC),REGION=220K,TIME=3
// STEPLIB DD DISP=(SHR,KEEP),DSNAME=LLCAD
// DD DSN=GMCCAL,DISP=(OLD,PASS),UNIT=SYSODA
// SCRATCH1 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
// SCRATCH2 DD UNIT=SYSODA,DISP=(NEW,DELETE),SPACE=(CYL,(1,1))
// PROBFIL DD UNIT=SYSODA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
// ETIL DD UNIT=SYSODA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
// MATRIX1 DD UNIT=SYSODA,SPACE=(CYL,(1,1)),DISP=(NEW,DELETE)
//SYSMLCP DD UNIT=SYSDA,DSNAME=*,CPC.,SYSMLCP,DISP=(OLD,DELETE)
//SYSPRT INT DD SYSOUT=A
//EXEC.FT02FOO1 DD DSN=*,S1.GO,F T02FOO1,DISP=(MOD,PASS)
//EXEC.FT03FOO1 DD DSN=*,S1.GO,F T03FOO1,DISP=(MOD,DELETE)
//EXEC.FT06FOO1 DD SYSOUT=A
//EXEC.FT07FOO1 DD SYSOUT=B
//EXEC.FT04FOO1 DD UNIT=SYSDA,SPACE=(CYL,{1,1})
//SYSPUNCH DD SYSOUT=B
//EXEC.FT10FOO1 DD UNIT=SYSDA,SPACE=(TRK,{1,1}),RLSE).
// DISP=(1,PASS),
// DBCS=(RECFM=FB,RECL=80,LBLKSIZE=880)
//SYSIN DD DSN=*,S1.GO,F T01FOO1,DISP=(OLD,DELETE)
//CHKF EXEC FORTGCLG,REGION.GO=42K,TIME=1
//FURT,SYSLIN DD DISP=(NEW,PASS)
//FURT,SYSLIN DD *
  READ(10,10C) I
100  FORMAT(15)
    IF(I) 20,20,10
10  STOP 10
20 CONTINUE
   STOP
END
//GU.FT10FOO1 DD DSN=*.EXEC.FT10FOO1,DISP=(OLD,DELETE)

//TWO EXEC PGM=OPTALG,COND=(2,LT,CHKF.GO),REGION=(,336K),TIME=2
//STEPLIB DD DSN=LOAD,DISP=SHR
//FTC6FOO1 DD SYSOUT=A
//FTO5FOO1 DD DSN=*,S1.GO,F T02FOO1,DISP=(OLD,DELETE)

//SFAIL EXEC FORTGCLG,COND.FORT=(0,EQ,CHKF.GO),
// COND.LKED=(10,EQ,CHKF.GO),(4,LT,CHKF.GO)
// REGION.GO=(10,EQ,CHKF.GO),(4,LT,EXTENT),(4,LT,LUAD)
// REGION.GO=42K,TIME=1
//FURT,SYSLIN DD DISP=(NEW,PASS)
//FURT,SYSLIN DD *
  REAL X(80)
5  READ(5,10,END=100) X
10  FORMAT(80A)
  WRITE(7,10) X
  GOTO 5
100  STOP
END
//GU.FT05FOO1 DD DSN=*,S1.GO,F T02FOO1,DISP=(OLD,DELETE)
NELC ZANGWL

CATALOG IDENTIFICATION:
E4 NELC ZANGWL

PROGRAMMERS:
D. C. McCALL, Decision and Control Technology Division, and
C. M. BECKER, Applications Software Division

PURPOSE:
To compute the minimum of a function \( f(x_1, \ldots, x_n) \), of \( n \) real variables

RESTRICTIONS AND LIMITATIONS:
A maximum of 20 variables can be handled.

LANGUAGE:
FORTRAN IV

COMPUTER CONFIGURATION:
IBM 360/65
Core storage: 19086 bytes

ENTRY POINTS:
ZANGWL

SUBPROGRAMS AND WHERE REFERENCED:
User-supplied programs
FUNC called by ZANGWL, (POWELL)
Programmer-supplied programs
ZGITER called by ZANGWL
POWELL called by ZANGWL

USAGE:
CALL ZANGWL (XI, N, EACCUR, QSTEP, ISTOP, LPRINT, IX,
1PUNCH, XOFT, FF)
For a description of parameters see the listing.

INPUT FORMAT:
All input is through the parameter list except when user-supplied search
directions are desired. Then ZANGWL expects \( N \) vectors of length \( N \) input on
cards in 4(F15.10, 5X) Format, where \( N = n \) is the number of variables.

OUTPUT:
The output depends on the print and punch option. See the listing.

ERROR MESSAGES:
None
PROGRAM DESCRIPTION:

ZANGWL — acts as a driver and convergence monitor for ZGITER. If the vector $x = (x_1, x_2, \ldots, x_n)$ on returning from ZGITER is to within $\varepsilon_{a,c u r}$ of the value on entering, ZANGWL returns.

ZGITFR — keeps track of the directions to be searched and normalizes each newly generated direction.

POWELL — finds the minimum of the objective function along a direction supplied by ZGITER using quadratic interpolation.

MATHEMATICAL METHOD:

ZANGWL is based on a method proposed by W. I. Zangwill in the Computer Journal, Vol. 10, 1967, pp. 293-296. The method is outlined as follows: Let $f(x_1, x_2, \ldots, x_n)$ be the function to be minimized, and $c_r, r = 1, \ldots, n$ be the unit coordinate directions. Assume that an initial point $p_0$ and $n$ normalized directions $\xi_1^1, \xi_2^1, \ldots, \xi_n^1$ are given.

To initialize, calculate $\lambda_0$ to minimize $f(p_0^0 + \lambda_0 \xi_1^1)$, then set $p_1 = p_0^0 + \lambda_0 \xi_1^1, t = 1$ and go to iteration $k$ with $k = 1$.

Iteration $k$: $p_{n+1}, \xi_r^k, r = 1, \ldots, n$ and $t$ are given.

Step (i): Find $\alpha$ to minimize $f(p_{n+1}^k + \alpha \xi_r^k)$. Update $t$ by

$$ t = \begin{cases} t + 1 & \text{if } 1 \leq t < n \\ t & \text{if } t = n \end{cases} $$

If $\alpha \neq 0$, let $p_{n+1}^k = p_{n+1}^{k-1} + \alpha \xi_r^k$. If $\alpha = 0$, repeat step (i). Should step (i) be repeated $n$ times in succession, stop; the point $p_{n+1}^k$ is optimal.

Step (ii): For $r = 1, \ldots, n$ calculate $\lambda_r^k$ to minimize $f(p_r^k + \lambda_r^k \xi_r^k)$ and define $p_r^k = p_r^k + \lambda_r^k \xi_r^k$. Let $\xi_{n+1}^k = (p_n^k - p_{n+1}^k - \lambda_x^k \xi_x^k) / \|p_n^k - p_{n+1}^k\|$

Determine $\lambda_n^k$ to minimize $f(p_n^k + \lambda_n^k \xi_{n+1}^k)$ and set

$$ p_{n+1}^k = p_n^k + \lambda_n^k \xi_{n+1}^k. $$

Define $\xi_r^{k+1} = \xi_{r+1}^k, r = 1, \ldots, n$

Go to iteration $k$ with $k+1$ replacing $k$.\textsuperscript{13}
NUMERICAL EXAMPLES – GENERAL TEST FUNCTIONS

In this section we list the functions for which ZANGWL computed the minimum, and summarize the results. These test functions are selected because the surfaces they define have steep curving valleys or have many known optima. The results are tabulated for each function. This table lists the various initial points, the computed optimum, the minimum value of the objective function, the number of function evaluations, and the elapsed CPU in seconds’ time on the IBM 360/65.

In comparison with minimization routines (FP and CNJGAT) at NELC, the computing times are a bit slower and many function evaluations are needed, but with the speed of the 360/65 this is relatively insignificant compared with the number of man-hours spent in deriving the analytic expression for the gradient.

1. Rosenbrock’s function: \( f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 \).
   This function has a steep valley along the parabola \( x_2 = x_1^2 \) with a minimum at \((1,1)\).

2. Cube: \( f(x_1, x_2) = 100(x_2 - x_1^3)^2 + (x_1 - 1)^2 \). Cube is similar to Rosenbrock’s function except the steep valley follows the curve \( x_2 = x_1^3 \) and has a minimum \((1,1)\).

3. Helical: \( f(x_1, x_2, x_3) = 100(x_3 - 100)^2 + (r - 1)^2 + x_3^2 \) where \( x_1 = r \cos 2\pi \theta, x_2 = r \sin 2\pi \theta, \) and \( r = \sqrt{x_1^2 + x_2^2} \). This function has a steep helical valley with a minimum at \((1,0,0)\).

4. THREE: \( f(x_1, x_2, x_3) = \frac{-1}{1 + (x_1 - x_2)^2} \sin \left( \frac{\pi}{2} x_2 x_3 \right) \exp \left( -\left( \frac{x_1 + x_3}{x_2} \right)^2 \right) \). THREE has minima at \( x_1 = x_2 = x_3 = \pm \sqrt{4n+1}, n \geq 0 \) integral with a minimum value of \(-3\). This function tends to change quickly from the point \((0,1,2)\) and then flattens out until it reaches an optimum. The optimum depends on the starting point \( x_0 \).

5. FOUR: \( f(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \). FOUR has its minimum at \((0,0,0)\).

6. CHEBYQ(UAD): This relatively new function allows testing a routine on a function with an arbitrary number of variables: i.e., CHEBYQ
(x_1, \ldots, x_n), where n is a parameter preset by the user. For n = 1, 2, \ldots, 7, 9 the minimum value of CHEBYQ is zero; however, for other n the minimization is still valid.

Briefly, CHEBYQ does the following: Let \( x = (x_1, \ldots, x_n) \) be a vector (abscissae) whose coordinates are in the range \( 0 < x_i < 1 \). Then, choosing the shifted chebyshev polynomial \( T_j \), we define:

\[
\Delta_j(x) = \int_0^1 T_j(z) \, dz - \frac{1}{n} \sum_{j=1}^{n} T_j(x_j)
\]

Then the function \( f(x) = \sum_{i=1}^{n} (\Delta_i(x))^2 \) has the property that if \( X \) is the vector of abscissae, then \( f = 0 \); otherwise, \( f > 0 \). Although contrived, CHEBYQ is a good example of a complicated objective function that can occur. The FORTRAN IV listing of CHEBYQ follows.
SUBROUTINE CHERYO(F,X,N)
IMPLICIT REAL*8(A-H,O-Z)
LOGICAL IEVEN
DIMENSION Y(2C),TI(2C),TIMIN(20),X(1)

DFLTA = 0.0000
ZERO = 0.0000
ONE = 1.0000
TWO = 2.0000

DO 10 J=1,N
Y(J) = TWO*Y(J)-ONE

10 TIMIN(J) = ONF

IF IEVEN = .FALSE.*

IF 20 I=2,N
IEVEN = .NOT.*IEVEN
DFLTA = ZERO

DO 19 J=1,N
TPLUS = TWO*Y(J)*TI(J)-TIMIN(J)

19 TI(J) = TPLUS

A = ZERO

IF IEVEN A = -ONE/(I*I-ONF)

DFLTA = DFLTA/N-A

DO 20 J=1,N

20 F = F+DFLTA*DFLTA

RETURN

END
ZANGWL· ROSIE

optimum at (1.0, 1.0)

QSTEP = 0.1; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.2</td>
<td>0.1000000000D01</td>
<td>0.16961904D-22</td>
<td>325</td>
<td>0.26</td>
</tr>
<tr>
<td>2 0.1</td>
<td>0.1000000000D01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

QSTEP = 0.01; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.2</td>
<td>0.999999991D00</td>
<td>0.77935877D-14</td>
<td>366</td>
<td>0.19</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.999999992D00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.2</td>
<td>0.999999991D00</td>
<td>0.77935878D-14</td>
<td>370</td>
<td>0.19</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>0.999999992D00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1.2</td>
<td>0.1000000000D01</td>
<td>0.23682822D-14</td>
<td>152</td>
<td>0.12</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>0.1000000000D01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ZANGWL - CUBE**

Optimum at (1.0, 1.0)

QSTEP = 0.1; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.2</td>
<td>0.999999999D00</td>
<td>0.31134541D-21</td>
<td>402</td>
<td>0.28</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.999999999D00</td>
<td>0.999999999D00</td>
<td>399</td>
<td>0.23</td>
</tr>
<tr>
<td>1 -1.2</td>
<td>0.999999999D00</td>
<td>0.31134835D-21</td>
<td>179</td>
<td>0.17</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>0.999999999D00</td>
<td>0.78369023D-21</td>
<td>216</td>
<td>0.12</td>
</tr>
</tbody>
</table>

QSTEP = 0.01; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.2</td>
<td>0.100000000D01</td>
<td>0.37920987D-25</td>
<td>473</td>
<td>0.25</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.100000000D01</td>
<td>0.37920987D-25</td>
<td>470</td>
<td>0.25</td>
</tr>
<tr>
<td>1 -1.2</td>
<td>0.100000000D01</td>
<td>0.27144901D-25</td>
<td>216</td>
<td>0.12</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>0.100000000D01</td>
<td>0.27144901D-25</td>
<td>216</td>
<td>0.12</td>
</tr>
</tbody>
</table>
ZANGWL: THREE

optima at $x_1 = x_2 = x_3 = \pm \sqrt{n+1}$, n integral

$QSTEP = 0.1; EACCUR = 1.0D-04$

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.0</td>
<td>0.10000000D01</td>
<td>-0.30000000D01</td>
<td>288</td>
<td>0.38</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.99999999D00</td>
<td>-0.30000000D01</td>
<td>298</td>
<td>0.30</td>
</tr>
<tr>
<td>3 2.0</td>
<td>0.99999999D00</td>
<td>-0.30000000D01</td>
<td>298</td>
<td>0.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.0</td>
<td>0.40012498D02</td>
<td>-0.30000000D01</td>
<td>5710</td>
<td>4.88</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.40012498D02</td>
<td>-0.30000000D01</td>
<td>5710</td>
<td>4.88</td>
</tr>
<tr>
<td>3 -2.0</td>
<td>0.40012498D02</td>
<td>-0.30000000D01</td>
<td>5710</td>
<td>4.88</td>
</tr>
</tbody>
</table>

$STEP = 0.01; EACCUR = 1.0D-04$

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.0</td>
<td>0.10000000D01</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.28</td>
</tr>
<tr>
<td>2 1.0</td>
<td>0.10000000D01</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.28</td>
</tr>
<tr>
<td>3 2.0</td>
<td>0.99999999D00</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.0</td>
<td>-0.10000000D01</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.35</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>-0.10000000D01</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.35</td>
</tr>
<tr>
<td>3 -2.0</td>
<td>-0.99999999D00</td>
<td>-0.30000000D01</td>
<td>257</td>
<td>0.35</td>
</tr>
</tbody>
</table>
ZANGWL - HELICAL

optimum at (1.0, 0.0, 0.0)

QSTEP = 0.1; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.0</td>
<td>-0.81379053D-15</td>
<td>0.29631116D-30</td>
<td>462</td>
<td>0.56</td>
</tr>
<tr>
<td>2 0.0</td>
<td>0.10000000D01</td>
<td>0.77927466D-32</td>
<td>366</td>
<td>0.38</td>
</tr>
<tr>
<td>3 0.0</td>
<td>-0.34372772D-15</td>
<td>0.18577895D-33</td>
<td>351</td>
<td>0.38</td>
</tr>
</tbody>
</table>

QSTEP = 0.01; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -1.0</td>
<td>-0.81379053D-15</td>
<td>9.14457809D-31</td>
<td>622</td>
<td>0.59</td>
</tr>
<tr>
<td>2 0.0</td>
<td>-0.54445252D-21</td>
<td>0.60518735D-34</td>
<td>375</td>
<td>0.42</td>
</tr>
<tr>
<td>3 0.0</td>
<td>0.10000000D01</td>
<td>0.10674562D-34</td>
<td>337</td>
<td>0.33</td>
</tr>
</tbody>
</table>
ZANGWL - FOUR

optimum at (0.0, 0.0)

QSTEP = 0.1; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Num. of cF Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3.0</td>
<td>-0.137543715D-5</td>
<td>0.68469142D-23</td>
<td>1350</td>
<td>0.95</td>
</tr>
<tr>
<td>2 -1.0</td>
<td>0.137543716D-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0.0</td>
<td>-0.57715785D-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 1.0</td>
<td>-0.57715786D-6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

QSTEP = 0.01; EACCUR = 1.0D-04

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Num. of cF Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3.0</td>
<td>0.58159117D-5</td>
<td>0.79920637D-20</td>
<td>1568</td>
<td>0.97</td>
</tr>
<tr>
<td>2 1.0</td>
<td>-0.58159117D-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 -2.0</td>
<td>0.44312920D-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 -1.0</td>
<td>0.44312921D-5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ZANGWL - CHEBYQUAD

**QSTEP = 0.01; EACCUR = 1.0D-04**

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Evaluations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>n = 6</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0.142</td>
<td>0.66876591D-1</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td>2 0.285</td>
<td>0.36668229D00</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td>3 0.428</td>
<td>0.63331770D00</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td>4 0.571</td>
<td>0.71125932D00</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td>5 0.714</td>
<td>0.93312341D00</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td>6 0.857</td>
<td>0.93312341D00</td>
<td>0.32039347D-17</td>
<td>704</td>
<td>1.81</td>
</tr>
<tr>
<td><strong>n = 8</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0.111</td>
<td>0.43152760D-1</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>2 0.222</td>
<td>0.50000000D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>3 0.333</td>
<td>0.49999999D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>4 0.444</td>
<td>0.73367129D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>5 0.555</td>
<td>0.80690916D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>6 0.666</td>
<td>0.95684724D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>7 0.777</td>
<td>0.95684724D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td>8 0.888</td>
<td>0.95684724D00</td>
<td>0.35168737D-02</td>
<td>209</td>
<td>7.99</td>
</tr>
<tr>
<td><strong>n = 10</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0.090</td>
<td>0.59619900D-1</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>2 0.181</td>
<td>0.16670828D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>3 0.272</td>
<td>0.23917065D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>4 0.363</td>
<td>0.39888429D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>5 0.454</td>
<td>0.39888429D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>6 0.545</td>
<td>0.60111570D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>7 0.636</td>
<td>0.60111570D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>8 0.727</td>
<td>0.76082934D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>9 0.818</td>
<td>0.83329171D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
<tr>
<td>10 0.909</td>
<td>0.94038009D00</td>
<td>0.65039548D-02</td>
<td>2603</td>
<td>14.30</td>
</tr>
</tbody>
</table>
SUBROUTINE ZANGUL

PURPOSE

TO FIND THE MINIMUM OF A REAL VALUED FUNCTION OF N-VARIABLES.

WHOSE VALUES ARE UNCONSTRAINED.

USAGE

CALL ZANGUL (XI, N, EACCUR, QSTEP, ISTOP, LPRINT, IX, LPUNCH, XOPT, FF)

DESCRIPTION OF PARAMETERS

XI - THE INITIAL GUESS OF THE OPTIMUM

N - THE NUMBER OF VARIABLES

EACCUR - THE INITIAL ACCURACY DESIRED. FOR BEST RESULTS SET

LESS THAN QSTEP**2.

QSTEP - THE INITIAL STEP SIZE FOR THE ONE DIMENSIONAL SEARCH

ROUTINE POWELL. QSTEP EQUAL .1 WORKS WELL.

ISTOP - QSTEP REDUCTION CODE

1 AFTER FINDING AN MINIMUM TO WITHIN EACCUR, QSTEP IS

SET TO EACCUR/EACCUR**2 AND THE ROUTINE DOES ONE

FINAL MINIMIZATION TO WITHIN EACCUR.

LPRINT - PRINT CODE

1 DO NOTHING

2 EACH N-DIMENSIONAL ITERATION ONLY

3 FINAL RESULTS ONLY

4 PLUS THE POINT ON ENTERING AND LEAVING POWELL.

IX - CHOICE OF USER SUPPLIED DIRECTIONS. IF. EQ. 0, CO-ORDINATE

DIRECTIONS ARE USED.

LPUNCH - A PUNCH OPTION, IF DIFFERENT FROM C, ZANGUL WILL PUNCH

OUTPUT FOR THE MINIMUM POINT.

XOPT - A LIST OF LENGTH N CONTAINING THE MINIMUM.

FF - FF = FUNC (XOPT,N)

NOTE ALL PARAMETERS ARE DOUBLE PRECISION.

SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED

USER MUST SUPPLY THE FUNCTION SUBPROGRAM FUNC(X,N)

METHOD

THIS ROUTINE IS BASED ON A METHOD PROPOSED BY W.I. ZANGWILL,
COMPUTER JOUR., VOL. 17, 1967, P293-296.

--------------------------------------------------------------------------------

SUBROUTINE ZANGUL (N, EACCUR, QSTEP, ISTOP, LPRINT, IX, LPUNCH, XOPT, FF)

IMPLICIT REAL*8 (A-H, O-Z)

DIMENSION NF(N), OL(20), XI(20, 21), Q(20), PI(20), XIHY(20)

DIMENSION P(20), XM(20, 21), XOPT(1)

INTEGER I, J, N, COUNT

REAL LAMDA, NORM, MINF

COMMON /MAIN/ EPSILN, Q, IF, IPWEL, OBJFN, LIST, COUNT
IP=0
Q = QSTEP
E = EACCUR
LIST = LPRINT
IHMCRO= IPUNCH
OBJFN=0
IPDWE=2
EPSLN=1.00-15
C IP=STOP FLAG TERMINATING SIMULTANEOUS REDUCTION OF BOTH Q AND E
C K=NUMBER OF CURRENT ITERATION
C OBJFN=CURRENT QUANTITY OF OBJECTIVE FUNCTION EVALUATIONS
C OLD(IC)=IC-TH COMPONENT OF ENTER POINT FOR ZGITER
C NEW(IC)=IC-TH COMPONENT OF INITIAL POINT AND POINT COMPUTED IN
C ZGITER
C R(IC)=OLD(IC) AT START OF ZGITER
C XIC(IDC)=IC-TH COMPONENT OF ID-TH NORMALIZED NONCOORDINATE
C DIRECTION
C XJ(IDC)=XIC(IDC) AT START OF ZGITER
C XIN=NEW(IDC)=IC-TH COMPONENT OF (N+1)-ST ('EXTRA')
C NORMALIZED NONCOORDINATE DIRECTION
C PT(IDC)=IC-TH COMPONENT OF MINIMUM POINT OF N-DIMENSIONAL
C MINIMIZATION
C DI(IDC)=IC-TH COMPONENT OF NORMALIZED NONCOORDINATE DIRECTION OF
C ONE DIMENSIONAL MINIMIZATION
C ALPHA=MINIMUM STEP LENGTH FOR N-DIMENSIONAL MINIMIZATION
C LAMDA=MINIMUM STEP LENGTH FOR ONE DIMENSIONAL MINIMIZATION
C MINFN=OBJECTIVE FUNCTION MINIMUM VALUE
C ZGFN=CURRENT ZGITER ITERATION OBJECTIVE FUNCTION VALUE
C STFN=OBJECTIVE FUNCTION VALUE AT INITIAL POINT
C JUMP=FIRST ZGITER ITERATION FLAG
C OBJ=OBJFN AT START OF ZGITER
C PTOF=MAXIMUM NEW AND OLD POINT COMPONENT DIFFERENCE
C
10 CONTINUE
IF (IP.GT.C) GO TO 20
IF (IP.GT.C) GO TO 30
STFN=FUNC(NEW,N)
FF=STFN
OBJFN=OBJFN+1
GO TO 40
30 STFN=FUNC(PT,N)
OBJFN=OBJFN+1
GO TO 40
40 IF (LIST(IC),IX) GO TO 80
PRINT 230, N, C, E
PRINT 480, LIST, IHMCRO, IX
IF (IP.EQ.3) GO TO 50
PRINT 430, (IC,PT(IC),IC=1,N)
GO TO 60
50 PRINT 430, (IC,NEW(IC),IC=1,N)
60 PRINT 530, STFN
IF (IP.GT.C) GO TO 70
GO TO 40
70 CONTINUE
I1F (I-0) GO TO 130
IF (I EQ. 0) GO TO 90
C
USER SUPPLIES INITIAL NONCOORDINATE DIRECTIONS
C
READ 340, (XI(I),IC=1,N),10=1,N
GO TO 130
C
C
COMPUTE INITIAL NONCOORDINATE DIRECTIONS
C
90
DO 120 I=1,N
DO 110 IC=1,N
IF (IC.EQ.10) GO TO 100
XI(IC,10)=0.0000
GO TO 110
100
XI(IC,10)=1.0000
110 CONTINUE
120 CONTINUE
C
MINIMIZE IN ONE DIMENSION USING INITIALIZATION DATA
C
130
DO 140 IC=1,N
140 D(I)=XI(I,N)
IF (IP.EQ.0) GO TO 160
IF (LIST.EQ.0) GO TO 145
PRINT 580, (IC,PT(IC),IC=1,N)
145 CALL POWELL (N,OLD,I),LAMBDA,FF)
DO 150 IC=1,N
150 OLD(IC)=OLD(IC)+LAMBDA*D(IC)
IF (LIST.EQ.0) GO TO 155
PRINT 590, (IC,OLD(IC),IC=1,N)
155 CONTINUE
GO TO 130
160 IF (LIST.EQ.0) GO TO 165
PRINT 560, (IC,NEW(IC),IC=1,N)
165 CALL POWELL (N,NEW,I),LAMBDA,FF)
C
C
COMPUTE FIRST POINT
C
170 OLD(IC)=NEW(IC)*LAMBDA*D(IC)
IF (LIST.EQ.0) GO TO 175
PRINT 600, (IC,OLD(IC),IC=1,N)
175 CONTINUE
C
INITIALIZE FOR FIRST ITERATION
C
K=1
T=1
JUMP=1
180 IF (I.EQ.0) GO TO 100
JUMP=JUMP+1
DO 20 C = 1, N
20 C = OLD (C)
DO 220 I = 1, N
DO 210 J = 1, N
210 X (I, J) = X (I, J)
220 CONTINUE
IF (LIST * IF, 0) GO TO 230
PRINT 540, K
C
230 CALL 7GTLUI (N, OLD, XI, I, ALPHA, PT, XINEW, JUMP, NOYES, FF)
C
240 ZGNF = FF
C
C COMPARISON OF NEW AND OLD POINTS FOR N-DIMENSIONAL MINIMUM
ACHIEVED
C
PTDF = ABS (PT (1) - OLD (1))
DO 250 I = 2, N
ON = ABS (PT (I) - OLD (I))
IF (IN * LE, PTDF) GO TO 250
PTDF = ON
250 CONTINUE
IF (PTDF * GF, 0) GO TO 260
IF (K * NE, 2) GO TO 260
OBJF = OBJ
K = K - 1
260 IF (LIST * IF, J) GO TO 270
IF (LIST * FO, 2) GO TO 270
PRINT 430, (I, C, OLD (I), I = 1, N)
PRINT 450, (I, (I, ID, XI (IC, ID), I = 1, N), ID = 1, N)
PRINT 550, K
PRINT 431, (I, OLD (IC), I = 1, N)
PRINT 450, (I, (I, ID, XI (IC, ID), I = 1, N), ID = 1, N)
PRINT 550, (I, XINC (IC), I = 1, N)
PRINT 520, K, ZGFN
PRINT 460, (I, PT (IC), I = 1, N)
PRINT 420, K, OBJF
IF (PTDF * GF, 0) GO TO 280
C
C TEST FOR N-DIMENSIONAL MINIMUM ACHIEVED
270 IF (COUNT * LT, N) GO TO 320
C
C N-DIMENSIONAL MINIMUM ACHIEVED
C
280 CONTINUE
IF (1MCKO * LE, 1) GO TO 290
PUNCH 610, 1, 1X
PUNCH 370, 4, 0, F
PUNCH 340, (PT (IC), ID = 1, N)
PUNCH 410, LIST, 1MCKD, KLIICK
IF (1X, E) GO TO 290
PUNCH 330, (XI (IC), ID = 1, N), ID = 1, N)
290 CONTINUE
MINF= FF
IF (LIST,LF,1) GO TO 31C
PRINT 350, MINF
PRINT 360, (CPT(IC),IC=1,N)
PRINT 440, (QJ)M,N
C. TEST FOR FINISH AND RESTART IF NOT
310 IF(GF,ISTOP) GO TO 33C
Q=E
E=E*E
K=K+1
JUMP=JUMP+1
IP=IP+1
IPOWEL=IPOWEL+IPOWEL
GO TO 1C
C. REITERATE UNTIL REQUIRED ACCURACY ACHIEVED
320 K=K+1
JUMP=JUMP+1
GO TO 19C
330 DO 335 J=1,N
335 XIP(I)=OLD(I)
PRINT 62C
RETURN
C.
340 FORMAT (4(F15.10,5X))
350 FORMAT (1HC,13HOBJECTIVE FUNCTION MINIMUM VALUE=,E19.11)
360 FORMAT (1HC,21HNO. MINIMUM POINT/(13,3X,E18.11))
370 FORMAT (12,3X,E15.1C,3X,E15.1C)
390 FORMAT (1HC,18HNC. CF DIM OR VAR=.12,5X,16'MONE DIM START
Q=E18.111,5X,9HACCUPACY=.E13.11)
400 FORMAT (1HC,21HNO. INITIAL POINT/(13,3X,E18.11))
410 FORMAT(21H)
420 FORMAT(1HC,21HITERATIONS 1 THROUGH ,19,1CH REQUIRED ,
19,31H OBJECTIVE FUNCTION EVALUATIONS)
430 FORMAT(28H ENTER ZGITER WITH THE POINT, /4H NO.7/(13,3X,E18.11))
431 FORMAT(26H LEAVE ZGITER AT THE POINT, /4H NO.7/(13,3X,E18.11))
440 FORMAT (1HC,32H ACHIEVING THIS MINIMUM REQUIRED ,19,36H
OBJECTIVE FUNCTION EVALUATIONS AND ,19,11H ITERATIONS)
450 FORMAT (1HC,31H NO. XI NO. DIRECTION/(2X,13,5X,
13,4X,E18.11))
460 FORMAT (1HC,17HNC. NEW POINT/(13,3X,E18.11))
460 FORMAT (1HC,11H PRINT CODE=,13,5X,11HPUNCH CODE=,13,5X,12HTIMING
CODE=13,5X,4X11 FLAG=,13)
510 FORMAT (1HC,74HNO. XI(N+1) /(13,3X,E18.11))
520 FORMAT (1HC,12HTERATION ,19,37H CHANGES OBJECTIVE FUNCTION VALUE
10. ,E18.11)
530 FORMAT (1HC,42H OBJECTIVE FUNCTION VALUE AT INITIAL POINT=,E18.11)
540 FORMAT (1HC,23H ENTER ZGITER ITERATION ,19)
550 FORMAT (1HC,23H LEAVE ZGITER ITERATION ,19)
560 FORMAT (1HC,32HNO. INITIALIZATION ENTER POWELL/(13,3X,E18.11))
590 FORMAT (1HC,17HNO. ENTER POWELL/(13,3X,E18.11))
590 FORMAT (1HC,17HNO. LEAVE POWELL/(13,3X,E18.11))
600 FORMAT (1HC,32HNO. INITIALIZATION LEAVE POWELL/(13,3X,E18.11))
610 FORMAT (21H)
620 FORMAT (1H1)
END
SUBROUTINE ZGITER (NOV, DPT, DIR, NC, DIST, G, EXTRA, JSW, ISW, FF)
IMPLICIT REAL*8 (A-H, O-Z)
DIMENSION DPT(20), DIR(20, 21), G(20), (20), M(20), NEW(20), EXTRA
L(20)
REAL*8 L, NEW,
INTEGER OBJFN, COUNT
COMMON/ZANG/EPSSILN, O, E, POWEL, OBJFN, LIST, COUNT
C
UNCONSTRAINED NOV-DIMENSIONAL MINIMIZATION WITHOUT USING DERIVATIVES
USING GIVEN POINT DPT AND GIVEN DIRECTION DIR
NOV=QUANTITY OF VARIABLES IN OBJECTIVE FUNCTION
NC=NUMBER OF CURRENT COORDINATE DIRECTION
C(DT(I))=IC-TH COMPONENT OF OLD POINT
NEW(I)=IC-TH COMPONENT OF NEW POINT
DIST=MINIMUM STEP LENGTH ALONG CURRENT DIRECTION NC
C(1C(I))=IC-TH COMPONENT OF CURRENT NORMALIZED COORDINATE DIRECTION
G(I)=IC-TH COMPONENT OF MINIMUM POINT IN NOV-MINIMIZATION
L=MINIMUM STEP LENGTH IN ONE DIRECTIONAL MINIMIZATION ALONG
NONCOORDINATE DIRECTION H
H(1C(I))=IC-TH COMPONENT OF CURRENT NONCOORDINATE DIRECTION IN ONE
DIMENSIONAL MINIMIZATION
COUNT=CURRENT TOTAL OF COORDINATE DIRECTIONS USED WITHOUT COMPUTING
'EXTRA' NONCOORDINATE DIRECTION
JSW=FIRST ZGITER ITERATION FLAG
C
PART ONE
COUNT = 0
IF (JSW.NE.1) GO TO 20
NC=NC+1
DO 10 JK=1, NOV
10 NEW(JK)=DPT(JK)
GO TO 110
C
COMPUTE CURRENT COORDINATE DIRECTION
C
20 DO 30 JC=1, NOV
IF (JC.EQ. NC) GO TO 30
C(JC)=0.0000
GO TO 30
30 CONTINUE
C
MINIMIZE IN ONE DIRECTION ALONG CURRENT COORDINATE DIRECTION
C
IF (LIST(NF).EQ. 4) GO TO 45
PRINT 250, (JC, DPT(JC), JC=1, NOV)
45 CALL POWELL (NOV, DPT, C, DIST, FF)
C
TEST FOR ALL COORDINATE DIRECTIONS USED
IF (NC.NE. NOV) GO TO 50
C
END
NC=1
GO TO 60
60 NC=NC+1
C TEST FOR MINIMUM STEP LENGTH ALONG C(NC)
40 IF (DAABS(DIST),GE,PSILN) GO TO 90
IF (COUNT.GE.NDV) GO TO 70
COUNT=COUNT+1
GO TO 20
C MINIMUM STEP LENGTH ALONG C(NC) FOUND
70 DB RC JK=1,NDV
40 G(JK)=OPT(JK)
F F U N N
C UPDATE CURRENT POINT
90 DO 100 JG=1,NDV
100 NEW(JG)=OPT(JG)+DIST*C(JG)
IF (LIST,NE,4) GO TO 110
PRINT 26C, (JC,NEW(JC),JC=1,NDV)
C PART TWO
C MINIMIZE IN ONE DIMENSION USING CURRENT POINT AND CURRENT
C NUNCOORDINATE DIRECTIONS
110 DO 120 JG=1,NDV
120 H(JG)=DIQ(JC,JG)
C TEST FOR ZERO DIRECTION
17=0
DO 130 JG=1,NDV
IF (DAABS(H(JG)) .LE.1.0D-15) GO TO 130
17=1
130 CONTINUE
IF (17,NE,0) GO TO 185
IF (LIST,NE,4) GO TO 155
PRINT 255, (IJ,NEW(IJ),IJ=1,NDV)
155 CALL POWELL (NDV,M=H,L,EP)
160 DO 170 KJ=1,NDV
170 NEW(KJ)=NEW(KJ)+H(KJ)
IF (LIST,NE,4) GO TO 180
PRINT 26C, (IJ,NEW(IJ),IJ=1,NDV)
180 CONTINUE
C COMPUTE EXTRA NUNCOORDINATE DIRECTION
C DENOM=0.0
DO 190 JG=1,NDV
190 DENOM=DENOM+ABS(NEW(JG)-OPT(JG))
DO 200 JG=1,NDV
DIQ(JG,NDV+1)=NEW(JG)-OPT(JG)/DENOM
200 CONTINUE
MINIMIZE IN ONE DIMENSION ALONG EXTRA UNIT COORDINATE DIRECTION
DO 210 JC=1,MDV
210 H(JC)=DIF(JC,NDV+1)
IF (LIST.NF,4) GO TO 215
PRINT 250, (JC,NF,H(JC),JC=1,MDV)
215 CALL POWELL (NDV,NEW,H,L,FF)
DO 220 JC=1,MDV
220 OPT(JC)=NEW(JC)+L(H(JC))
IF (LIST.NF,4) GO TO 225
PRINT 260, (JC,OPT(JC),JC=1,MDV)
C COMPUTE UNIT-COORDINATE DIRECTIONS FOR NEXT ITERATION
225 DO 240 JC=1,MDV
DO 230 JC=1,MDV
230 OPT(JC,J0)=0.0(JC,J0+1)
240 CONTINUE
140 DO 150 JK=1,MDV
150 G(JK)=NEW(JK)
RETURN
C 250 FORMAT (1HE,17HN), POWELL ENTER/(13,3X,E18.11)
260 FORMAT (1HE,17HN), POWELL LEAVE/(13,3X,E18.11)
END

SUBROUTINE POWELL (N,PO,S,STEP,FF)
IMPLICIT REAL*4 (A-H,O-Z)
DIMENSION S(20), PO(20), Z(20), F(4), V(4)
INTEGER OBJN, COUNT
COMMON/ZANG/FSILE,G,F,POWEL,OBJFN,LIST,COUNT
C UNCONSTRAINED ONE DIMENSIONAL MINIMIZATION WITHOUT USING DERIVATIVES
C FOR FINDING BELOW REASONABLE Q VALUE, SEE: CALAHAN,D.A., COMPUTER
C PO=INITIAL POINT VECTOR
C S=GIVEN DIRECTION VECTOR
C GIVEN PO,S: FIND STEP THAT MINIMIZES OBJECTIVE FUNCTION PO*STEP+S
C GIVEN PO,S: FIND STEP THAT MINIMIZES OBJECTIVE FUNCTION PO+STEP*
C N=QUANTITY OF VARIABLES IN OBJECTIVE FUNCTION
C Q=INITIAL STEP LENGTH ALONG S
C E=GEOMETRIC SERIES RATIO FOR FINDING REASONABLE STEP LENGTH IN
C COMPUTING SET OF INITIAL THREE POINTS
C CF=GEOMETRIC SERIES COEFFICIENT FOR FINDING ABOVE REASONABLE STEP
C E=REQUIRED ACCURACY OF MINIMUM POINT (EACH COMPONENT)
C V=ARRAY OF CURRENT POINT VALUES
C E=ARRAY OF CURRENT POINT OBJECTIVE FUNCTION VALUES
C OBJN=QUANTITY OF OBJECTIVE FUNCTION EVALUATIONS
C NUM=QUANTITY OF QUADRATIC INTERPOLATIONS
C N=M
C COMPUTE THREE STARTING VALUES AND THEIR OBJECTIVE FUNCTION VALUES
C
10 Z(1)=P(1)
FA=FF
DO 20 I=1,N
20 Z(1)=P(O(I)+0*ST(I)
FC=FUNC(Z,N)
OBJFN=OBJFN+1
IF (FF.LT.FA) GO TO 50
DO 30 I=1,N
30 Z(1)=P(O(I)-0*ST(I)
FNQ=FUNC(Z,N)
OBJFN=OBJFN+1
IF (FNQ.GE.FA) GO TO 40
B=-N
FF=FNQ
ISW=1
GO TO 60
40 V(1)=0
V(2)=0.0000
V(3)=0
F(1)=FNQ
F(2)=FA
F(3)=FNQ
GO TO 120
50 D=0
FR=FO
ISW=1
60 CF=1.0000
SUM=1.0000
70 CF=CF*FR
SUM=SUM+CF
IF (ISW.ED.1) GO TO 80
C=-0.0000
GO TO 90
80 C=0.0000
90 DO 100 I=1,N
100 Z(1)=P(O(I)+0*ST(I)
FC=FUNC(Z,N)
OBJFN=OBJFN+1
IF (FC.GT.FA) GO TO 110
A=R
FA=RR
B=C
FR=FC
GO TO 70
110 V(1)=A
V(2)=B
V(3)=C
F(1)=FA
F(2)=FR
F(3)=FC
C
C COMPUTE ZERO OF FIRST DERIVATIVE OF APPROXIMATING QUADRATIC THROUGH
C THREE CURRENT POINTS AND ITS OBJECTIVE FUNCTION VALUE
C
120 W1=V(2)-V(3)
W2=V(2)+V(3)
W3=V(3)-V(1)
W4=V(3)+V(1)
W5=V(1)-V(2)
W6=V(1)+V(2)
W7=2.*(W1*F(1)+W3*F(2)+W5*F(3))/W7
V(4)=(W1*W2*F(1)+W3*W4*F(2)+W5*W6*F(3))/W7
130 Z(I)=PO(I)+V(4)*S(I)
F(4)=FUNC(7,N)
OBJF=OBJF+1
IF (V(4).NE.V(2)) GO TO 140
GO TO 160
C TEST THREE CURRENT POINTS FOR CLOSENESS TO ZERO OF FIRST DERIVATIVE
140 IJ1=1
150 V1=VABS(V(1J1)-V(4))
IF (V1.EGE.0.0D0) GO TO 170
C ZERO OF FIRST DERIVATIVE IS MINIMUM DISTANCE ALONG S
160 STEP=F(4)
IF (F(4).LT.F(2)) GO TO 240
STEP=F(2)
GO TO 240
170 IF (IJI-2) 180,190,220
180 IJI=2
GO TO 150
190 IJI=3
GO TO 150
C SHRINK CLOSED INTERVAL CONTAINING MINIMUM BY DISCARDING BOTH POINT
C OUTSIDE OF INTERVAL AND ITS OBJECTIVE FUNCTION VALUE AND
C RELABELLING REMAINING POINTS AND THEIR OBJECTIVE FUNCTION VALUE
C
200 IF (V(4)-V(2).GE.0.0D0) GO TO 220
IF (F(4)-F(2).GE.0.0D0) GO TO 210
F(3)=F(2)
V(3)=V(2)
F(2)=F(4)
V(2)=V(4)
IF (NUM.GF.IP>WFL) GO TO 16C
NUM=NUM+1
GO TO 120
210 F(1)=F(4)
V(1)=V(4)
IF (NUM.GF.IP>WFL) GO TO 160
NUM=NUM+1
GO TO 120
220 IF (F(4)-F(2) > 0.1) GO TO 230
F(3)=F(4)
V(3)=V(4)
IF (NUM.GE.IN1WEL) GO TO 160
NUM=NUM+1
GO TO 120
230 F(11)=F(1)
V(11)=V(1)
F(12)=F(2)
V(12)=V(2)
IF (NUM.GE.IN1WEL) GO TO 160
NUM=NUM+1
GO TO 120
240 CONTINUE
RETURN
END
NELC FESDIR

CATALOG IDENTIFICATION:
E4 FESDIR

PROGRAMMER:
Gail Grotke, Decision and Control Technology Division

PURPOSE:
To minimize a function \( f(x_1, x_2, \ldots, x_n) \) of \( n \) variables whose values are constrained

RESTRICTIONS AND LIMITATIONS:
The function and constraints, and their first partial derivatives, must be continuous.

LANGUAGE:
FORTRAN IV

COMPUTER CONFIGURATIONS:
IBM 360/65

ENTRY POINTS:
FESDIR

SUBPROGRAMS AND WHERE REFERENCED:
Programmer-Supplied Programs
FUNC called by FESDIR, and POWEL, and BNDY
POWEL called by FESDIR
BNDY called by POWEL

Library Subprograms
SQRT called by FESDIR
ABS called by FESDIR, POWEL, and BNDY

DEFINITION OF VARIABLES

\begin{tabular}{ll}
XO & Initial feasible point \\
X & Working point and final minimum \\
S & Direction vector \\
GF & Gradient of the function \\
C & The constraints \\
GC & Gradients of the constraints \\
DELG & Normalized gradient of the violated constraint \\
DELF & Normalized gradient of the function \\
NUM & Number of iterations through Powel \\
N & Number of variables \\
IC & Number of constraints \\
STEP & Value returned from Powel that gives the minimum in the \( S \) direction \\
IFLAG & Number of violated constraints \\
\end{tabular}
FBOUND: Criterion by which it is determined whether function values are converging.
GBOUND: Criterion by which it is determined whether direction vector is converging.
CODE: Code that determines what will be printed.

INPUT FORMAT:
A driver and a function subprogram are needed. The driver calls FESDIR (F, XO, N, FBOUND, GBOUND).
The parameters N, IC, FBOUND, GBOUND, CODE and the feasible point XO for FESDIR and the parameters E, IPOWEL, and OBJFN for POWEL must be set in the driver. These three parameters for POWEL are in the common block labeled ZANG. IC and CODE are in common with FFSDIR.
The function FUNC has a parameter list: (K, X, C, GF, GC).
If k = 1 when FUNC is called, only the value of the objective function is returned.
If k = 2, the constraints are evaluated and returned in C.
If k = 3, the gradients of the constraint and the gradient of the function are returned in GC and GF, respectively.
If k = 4, only the gradient of the function is returned.

OUTPUT FORMAT:
Prints out according to the Print Code CODE.

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Value of CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function value, point, and constraint at each step in POWEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function value returned from POWEL, step size, and minimum point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Gradient and normalized gradients of violated constraints and function, and their sum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Initial and final points, minimum, number of iterations through POWEL, and number of function evaluations</td>
</tr>
</tbody>
</table>

ERROR MESSAGES:
None.

PROGRAM DESCRIPTION:
Main Program
The driver sets the values of the parameters and calls FESDIR.
Subroutines and Functions
FESDIR - finds the minimum of the function within the constraint set.
POWEL - is a one-dimensional quadratic search.
FUNC - evaluates the objective function, the constraints, and the gradients of each.
BNDY - checks to see if a point chosen by the quadratic search is within the constraint set and records the number of constraints that are violated.
MATHEMATICAL METHOD:

Given an objective function \( f(x_1, x_2, \ldots, x_n) \) and a set of constraints \( g_i(x_1, x_2, \ldots, x_n) \leq 0 \), FESDIR finds \( x \) such that \( f(x) \) is minimum and each constraint \( g_i(x) \) is satisfied; that is, less than zero. To accomplish this, FESDIR uses a method of feasible directions given in the article "Nonlinear Programming with the Aid of a Multiple-Gradient Summation Technique" by Klingman and Himmelblau. Klingman and Himmelblau suggest using a new direction given by \( \text{(NSD)} \), new successful direction, defined as

\[
\text{(NSD)} = \sum_{j=1}^{KC} \left\{ \frac{\text{Grad } C_j(x)}{|\text{Grad } C_j(x)|} + \frac{\text{Grad } F(x)}{|\text{Grad } F(x)|} \right\},
\]

where \( KC \) is the number of constraints violated and \( \text{Grad} \) is the gradient or first partial derivative. So \( \text{(NSD)} \) is the sum of the normalized gradients of the violated constraints and the normalized gradient of the function.

FESDIR uses this idea in choosing a new direction. Starting with a feasible point — that is, a point, \( X \), that satisfies all the constraints — and with a direction, \( S \), FESDIR uses a one-dimensional quadratic search, POWEL, to find a value \( \text{STEP} \) such that the minimum feasible point along the direction \( S \) is given by \( X_{\text{min}} = X + \text{STEP} \cdot S \). If any constraints were violated in finding the minimum, a new direction is determined by the negative of the sum of the normalized gradients of violated constraints and the normalized gradient of the function. Otherwise, the direction is the negative of the normalized gradient of function. The point and direction are then used to find a new minimum point and a new direction. This process is continued until it satisfies the convergence criterion; that is, until the function values converge or the direction vector converges.

1. The Two-Variable Problem

Minimize \( f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 \)

subject to

\( g_1 = x_1 + x_2 - 2 \leq 0 \)
\( g_2 = x_1^2 - x_2^2 \leq 0 \)

Minimum \( f(1,1) = 1 \)

2. The Circle Problem

Minimize \( f(x_1, x_2) = -\frac{1}{(x_1 - 1)^2 + x_2^2} \)

Subject to

\( g_1 = -x_1^2 - x_2^2 + 4 \leq 0 \)
\( g_2 = -16 + x_1^2 + x_2^2 \leq 0 \)

Minimum \( f(-2,0) = -1.0 \)

103
3. The Three-Variable Problem

Minimize 
\[ f(x_1, x_2, x_3) = x_1^2 - 6x_1^2 + 11x_1 + x_3 \]

subject to
\[ g_1 = x_1^2 + x_2^2 - x_3^2 \leq 0 \]
\[ g_2 = -x_1^2 - x_2^2 - x_3^2 + 4 \leq 0 \]
\[ g_3 = -x_1 \leq 0 \]
\[ g_4 = -x_2 \leq 0 \]
\[ g_5 = -x_3 \leq 0 \]
\[ g_6 = x_3 - 5 \leq 0 \]

Minimum 
\[ f(0, \sqrt{2}, \sqrt{2}) = -\sqrt{2} \]

4. The Colville Problem

1. The Two-Variable Problem

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>0.999999982</td>
<td>1.0000003</td>
<td>32</td>
</tr>
<tr>
<td>2.0</td>
<td>0.99999969</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. The Circle Problem

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>-0.199589 E01</td>
<td>-0.988216</td>
<td>191</td>
</tr>
<tr>
<td>2.8</td>
<td>0.141888 E00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. The Three-Variable Problem

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.378964 E00</td>
<td>0.17968266 E-08</td>
<td>1.5673426</td>
<td>30</td>
</tr>
<tr>
<td>0.168076 E01</td>
<td>0.15673425 E01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.234720 E01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. The Colville Problem

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.78619999 E02</td>
<td>0.78012675 E02</td>
<td>-0.30631831 E05</td>
<td>29</td>
</tr>
<tr>
<td>0.33439999 E02</td>
<td>0.33144035 E02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.31070000 E02</td>
<td>0.30144462 E02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.44180000 E02</td>
<td>0.44999996 E02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.35319999 E02</td>
<td>0.36546304 E02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SAMPLE PROBLEM AND PROGRAM LISTING

C FEASIBLE DIRECTIONS
DIMENSION X(5), XO(5)
INTEGER OBJFN, CODE
COMMON/ZANG/ F, IPOWEL, OBJFN
COMMON/CON/IC, KQ, CODE, ICON(16)
KQ=6
CODE=1
N=2
IC=2
XO(1)=-1.0
XO(2)=2.0
FBOUND = 1.0E-12
GBOUND = 1.0E-12
E=1.0E-6
IPOWEL=5
OBJFN=0
CALL FESDIR (F, XO, X, N, FBOUND, GBOUND)
STOP
END

FUNCTION FUNC (K, X, C, GF, GC)
DIMENSION C(16), GF(5), GC(16, 5), X(5)
FUNC = 1.0
TA=X(1)-2.0
TB=X(2)-1.0
GO TO (100, 200, 300, 400), K
100 CONTINUE
FUNC=TA*TA+TB*TB
RETURN
200 CONTINUE
C(1)=X(1)+X(2)-2.0
C(2)=X(1)*X(1)-X(2)
RETURN
300 CONTINUE
GC(1, 1)=1.0
GC(1, 2)=1.0
GC(2, 1)=2.0*X(1)
GC(2, 2)=-1.0
400 CONTINUE
GF(1)=2.0*(X(1)-2.0)
GF(2)=2.0*(X(2)-1.0)
RETURN
END
SUBROUTINE FESDIR (F, XO, X, N, FBOUND, GBOUND)

C
C REFERENCE: 'NONLINEAR PROGRAMMING WITH THE AID OF A MULTIPLE-
GRADIENT SUMMATION TECHNIQUE' BY W.R. KLINGMAN AND D.M. HIMMELBLAU
IN THE JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, VOL. II,

C DEFINITION OF THE VARIABLES.
XO INITIAL FEASIBLE POINT
X WORKING POINT AND FINAL MINIMUM
S DIRECTION VECTOR
GF GRADIENT OF THE FUNCTION
C THE CONSTRAINTS
GC GRADIENTS OF THE CONSTRAINTS
DELG NORMALIZED GRADIENT OF THE VIOLATED CONSTRAINT
DELF NORMALIZED GRADIENT OF THE FUNCTION
NUM NUMBER OF ITERATIONS THROUGH POWEL
N NUMBER OF VARIABLES
IC NUMBER OF CONSTRAINTS
STEP VALUE RETURNED FROM POWEL THAT GIVES THE MINIMUM IN THE
S DIRECTION
IFLAG NUMBER OF VIOLATED CONSTRAINTS
FBOUND CRITERION TO SEE IF FUNCTION VALUES ARE CONVERGING
GBOUND CRITERION TO SEE IF DIRECTION VECTOR IS CONVERGING
CODE CODE THAT DETERMINES WHAT WILL BE PRINTED

DIMENSION S(5),GF(5),C(16),GC(16,5),XO(5),X(5),GSUM(5),P(5)
DIMENSION DELG(5),DELF(5)
INTEGER OBJFNCODE
COMMON/ZANG/ FIPOWELOBJFN
COMMON/CON/ICKQCODE,ICON(16)
NUM=0

C START WITH FEASIBLE POINT
DO 10 J=1,N
10 X(J)=XO(J)

C SET DIRECTION TO THE NORMALIZED GRADIENT OF F
SUM=0.
DO 20 J=1,N
SQUAD=GF(J)*GF(J)
20 SUM=SUM+SQUAD
DO 21 J=1,N
S(J)=-GF(J)/SQRT(SUM)

C COMPUTE MINIMUM ALONG THE S DIRECTION
25 NUM=NUM+1
CALL POWEL (N, X, S, STEP, F, IFLAG)
DO 26 J=1,N
26 X(J)=X(J)+STEP*S(J)
IF (CODE.NE.0) WRITE(KQ,500) NUM,F,STEP,(J,X(J)),J=1,N)
27 IF (NUM.EQ.1) GO TO 28
28 FSAVE=F

C CHECK FOR CONVERGENCE
IF (ABS(F-FSAVE).LT.FBOUND) GO TO 110
VAL = FUNC (3,X,Y,C,GF,GC)

C COMPUTE SUM OF NORMALIZED GRADIENTS OF VIOLATED CONSTRAINTS
DO 30 J=1,N
30 GSUM(J)=0.0
DO 60 I=1,IC
IF (ICON(I).EQ.1) GO TO 60
SUM=0.0
DO 40 J=1,N
GSQD=GC(1,J)*GC(1,J)
        SUM=SUM+GSQD
DO 50 J=1,N
DELF(J) = GC(1,J)/ SQRT(SUM)
        IF (GC(1,J).LT.0.) AND (DELF(J).GT.0.) DELF(J)=-DELF(J)
50 GSUM(J)= GSUM(J)+DELF(J)
IF (CODE.LE.1) GO TO 60
WRITE (1,540)
WRITE (KQ,550) (GF(J),DELF(J),S(J),J=1,N)
60 CONTINUE

C COMPUTE THE NORMALIZED GRADIENT OF F
SUM=0.0
DO 70 J=1,N
FSQD=GF(J)*GF(J)
70 SUM=SUM+FSQD
ABDELF = SQRT(SUM)
DO 80 J=1,N
DELF(J) = GF(J)/ABDELF

C S IS THE NEW DIRECTION
S(J)= -DELF(J) -GSUM(J)
80 CONTINUE
IF (CODE.LE.1) GO TO 85
WRITE (KQ,560)
WRITE (KQ,570) (GF(J),DELF(J),S(J),J=1,N)
85 SUM=0.0
DO 90 J=1,N
SSQD=S(J)*S(J)
90 SUM=SUM+SSQD
DELS=SQRT(SUM)

C CHECK FOR CONVERGENCE
IF (DELS.LT.GBOUND ) GO TO 110
DO 100 J=1,N

C S NORMALIZED
100 S(J)=S(J)/DELS
SUBROUTINE POWFL (N, PO, S, STEP, FFYIFLAG)
DIMENSION S(20), PO(20), Z(20), F(4), V(4), CO(30)
INTEGER OBJFN
COMMON/ZANG/ F, IPOWFL, OBJFN
STEP=0.0
NUM=0
R=1.5
A=0.0
Q=0.1
GO TO 5

4
R=1.
DO 8 I=1,N

8 PO(I)=PO(I)+STEP*S(I)

5 CONTINUE
FA=FF
DO 20 I=1,N

20 Z(I)=PO(I)+Q*S(I)
CALL BNDY (CO, IFLAG, Z, N)
IF (IFLAG.GE.1) GO TO 25
FQ = FUNC (1, Z, OF, GF, GC)
OBJFN=OBJFN+1
IF (FQ.LT.FA) GO TO 50
CONTINUE
IFLAG=1-IFLAG
DO 30 I=1,N

30 Z(I)=PO(I)-Q*S(I)
CALL BNDY (CO, IFLAG, Z, N)
IF (IFLAG.EQ.0) GO TO 32
IF (STEP.EQ.0) GO TO 4
RETURN

32 FNC = FUNC (1, Z, OF, GF, GC)
OBJFN=OBJFN+1
IF(FNQ.GE.FA.AND.IFLAG1.EQ.0) GO TO 40
IF (FNQ.LT.FA) GO TO 35
IF (STEP.EQ.0.1) GO TO 4
DO 33 I=1,N
  33 Z(I) = PO(I) + Q*S(I)
  CALL BNDY(CO,IFLAG,Z,N)
RETURN
  35 B=-Q
  FB=FNQ
  ISW=0
  GO TO 6C
  40 STEP=-Q
  FF=FNQ
  RETURN
  50 B=0
  FR=FO
  ISW=1
  60 CF=1.0
  SUM=1.0
  70 CF=CF*R
  SUM=SUM+CF
  IF (ISW.EQ.1) GO TO 80
  C=-Q*SUM
  GO TO 90
  80 C=Q*SUM
  90 DO 100 I=1,N
    Z(I)=PO(I)+C*S(I)
    CALL BNDY(CO,IFLAG,Z,N)
  IF (IFLAG.EQ.01 GO TO 102
  101 STEP=B
    FF=FB
    RETURN
  102 CONTINUE
    FC = FUNC(1,Z,0,GF,GC)
    ORJFN=OBJFN+1
    IF (FC.GT.FB) GO TO 11C
    A=R
    FA=FR
    B=C
    FR=FC
    GO TO 70
  110 STEP=A
    FF=FA
    IF (STEP.EQ.0.0) GO TO 4
    RETURN
END

SUBROUTINE BNDY(C,IFLAG,X,N)
  DIMENSION C(16),X(5)
  INTEGER CODE
  COMMON/CUN/IC,KQ,CODE,ICUN(16)
  IFLAG=0
VAL = FUNC (2, X, C, GF, GC)
F = FUNC (1, X, C, GF, GC)
IF (CODE, NE, 3) GO TO 10
WRITE (KQ, 59) F
WRITE (KQ, 60) (K, X(K), K=1, N)
WRITE (KQ, 70) (L, C(L), L=1, IC)
10 DO 40 J = 1, IC
IF (ABS(C(J)), LT, 1, CE-15) C(J) = 0.
IF (C(J), LT, 0, 0 ) GO TO 30
ICON(J) = 1
IFLAG = IFLAG + 1
IF (CODE, NE, 3) GO TO 40
WRITE (KQ, 80) J
GO TO 40
30 ICON(J) = 1
40 CONTINUE
60 FORMAT (4H F =, F15.8)
60 FORMAT (4H X(1, 12H) =, E15.8)
70 FORMAT (2H C, 12H, E15.8)
80 FORMAT (11H CONSTRAINT, 12H IS VIOLATED)
RETURN
END
RICOCHET GRADIENT (28 SUBROUTINES AND DRIVER)

PROGRAMMER:

J. Greenstadt and R. T. Mertz, IBM/Adapted for Use at NELC by D. C. McCall, Decision and Control Technology Division

PURPOSE:

To find the point \((x_1, x_2, \ldots, x_n)\) at which the objective function \(f(x_1, x_2, \ldots, x_n)\) takes on its maximum value, subject to the constraints \(g_k(x_1, x_2, \ldots, x_n) \geq 0\)

RESTRICTIONS AND LIMITATIONS:

The program handles up to 50 constraints and 50 variables. The first partial derivatives of the function and all constraints must be obtainable.

LANGUAGE:

FORTRAN IV

COMPUTER CONFIGURATIONS:

IBM 360/65

ENTRY POINTS:

Main

SUBPROGRAMS AND WHERE REFERENCED:

Programmer-supplied Programs

PROB called by OBFUNC, OBGRAD, CONSTR, CSTRNM

Library Subprograms

ABS
SQRT

DEFINITION OF VARIABLES:

The manual gives a complete definition of the variables and gives a summary for each subroutine.

INPUT FORMAT:

A subroutine PROB and a set of data cards are needed. The subroutine defines the objective function, constraints, and gradients. The form used is

SUBROUTINE PROB (NUMX, X, KK, INDX, VAR, GRAD, N, NC, C)
The necessary dimensioning is
DIMENSION X(50), GRAD(50), NC(200), C(200)

DEFINITION OF THE VARIABLES:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMX</td>
<td>Serial number of X (unused by PROB)</td>
</tr>
<tr>
<td>X</td>
<td>The coordinates of the point</td>
</tr>
<tr>
<td>KK = 0</td>
<td>Gives a value of the objective function</td>
</tr>
<tr>
<td>1</td>
<td>Gives a value of the KKth constraint</td>
</tr>
<tr>
<td>INDX = 0</td>
<td>Gives a function value</td>
</tr>
<tr>
<td>1</td>
<td>Gives a gradient value</td>
</tr>
<tr>
<td>VAR</td>
<td>Returns the objective or constraint function value</td>
</tr>
<tr>
<td>GRAD</td>
<td>Returns the N-gradients</td>
</tr>
<tr>
<td>N</td>
<td>Number of variables</td>
</tr>
<tr>
<td>NC</td>
<td>Can be used to read in integer data (up to 200)</td>
</tr>
<tr>
<td>C</td>
<td>Can be used to read in real data (up to 200)</td>
</tr>
</tbody>
</table>

See sample subroutine.
**SAMPLE SUBROUTINE**

```plaintext
C SUBROUTINE, Resize (DIMX/X/K, DIMY/Y/K, VAL/X/K, VAL/Y/K)

C NO. PROBLEM 3

C KL16, MN, AND 111111. PLAN TEST EXAMPLE

C

C DIMENSION X(50), GRAD(50), NC(200), C(200)

C

C ROUTING

100 IF (I(IOA)) 210, 110, 120

C EVALUATE

110 VALUE

111 IF (KK) 200, 1000, 112

112 IF (KK-1) 113, 1002, 200

113 J001=KK

114 GO TO (1001, 1002), J001

C

C GRADIENT

120 GO 121 J001

121 GRAD(J, J2)

122 IF (KK) 210, 2000, 124

124 IF (KK-2) 125, 2002, 210

125 J002=KK

126 GO TO (2010, 2020), J002

C

C RETURN

210 VALUE

210 RETURN

C

C CALCULATION - VALUES

C

1000 VALUES 1.0*(X(1)1)1.1*(X(1)1.1)+Y(2)1+Y(2)2) Objective function

1001 VALUES X(1)1+Y(1)2+X(2)2 Constraints

1002 VALUES 16.0-X(1)1+Y(1)2+X(2)2 Constraints

GO TO 200

C

C CALCULATION - GRADIENTS

C

2000 GRAD(X(1)1)1.0*(X(1)1.1)+Y(2)1+Y(2)2) Gradient of objective function

2010 GRAD(X(1)2)2.0*(X(1)1.1)+Y(2)1+Y(2)2)

2010 GRAD(Y(2)2)2.0*(X(1)1.1)+Y(2)1+Y(2)2)

GO TO 210

C

2020 GRAD(X(1)1)2.0*(X(1)1.1)

2020 GRAD(Y(1)1)2.0*(X(1)1.1)

GO TO 210

C

C

C

* indicates cards that are changed from program to program.

113
```
The user-supplied data follow the required data. The user's data are divided into seven classes; preceding each class is a header card with the class number in the first column. All other data cards must be blank in the first column and are read only through column 72.

The seven classes are:

0 Description of the problem
1 Setting for the output control switches (See OUTPUT FORMAT)
2 Integer parameters used in the algorithm subroutines
3 Real parameters used in the algorithm subroutines, including the feasible starting point
   Classes 2 and 3 usually use the same parameters from problem to problem except for the starting point.
   The starting point begins in the 47th entry under Class 3, and is read in with an E14.8, 3E15.8 format.
4 Size of the problem. The number of variables in columns 2-5; the number of constraints in columns 6-10.
5 Integer Parameters for the PROB subroutine – I4, I5I5 format
6 Real Parameters for the PROB subroutine – E14.8, 3E15.8 format

The reading in of data is terminated by two cards with a 9 in column 1. If the program is to be run more than once with different data, one card with a 9 in column 1 is placed between the data for the different problems and two 9-cards are used to terminate the program.
## SAMPLE DATA DECK

### ODD CONSTRUCTED TEST EXAMPLE FROM KLINGMAN AND HINZPLOKA PAPERS

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>81</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>91</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Condition codes

- 1: for printout control

### Integer parameters

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>20</th>
<th>20</th>
<th>20</th>
<th>20</th>
<th>50</th>
<th>6</th>
<th>20</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100</td>
<td>1.E-5</td>
<td>1.E-4</td>
<td>1.E-3</td>
<td>1.E-2</td>
<td>1.E-1</td>
<td>1.E0</td>
<td>1.E1</td>
<td>1.E2</td>
</tr>
</tbody>
</table>

### Real parameters


### Initial point

2.6 1 2.6 2.6

### Program size

2 2

---

See page 116 for further information.
OUTPUT FORMAT:

The output for the program is controlled by control switches set under the first class of data. Following the header card are 10 cards numbered consecutively from 1-10 in columns 4 and 5. The switch controls are in columns 6 through 72. A "1" punch indicates the value will be printed; a blank indicates the value will be omitted. In the description of each subroutine in the manual the code for printing the values is given. For example, to print the final function value from the subroutine MONITR, we use the code 1, 44, F (on page 38). A "1" in column 44 of the first card will cause F to be printed. This code appears with the printout to help with the identification of the values.

A good choice for the minimal amount of printout (just the final result) is 1's in columns 11, 16, 17, 35, 38, 44, 45, 48, 51, 54 on the first card; the other cards are blank in columns 6-72.

The maximum printout occurs when all 10 cards are blank in columns 6 through 72. For debugging purposes the values desired may be chosen from the manual.

MATHEMATICAL METHOD AND REFERENCE:

See the manual Contributed Program Library #360D-15.3001 'Non-Linear Optimization-Ricochet Gradient Method.'

TEST EXAMPLES:

The following problems were tested with this program:

Problem 1.
Minimize \( f = (x_1 - 2)^2 + (x_2 - 1)^2 \)
\( g_1 = x_1 + x_2 - 2 \leq 0 \)
\( g_2 = x_1^2 - x_2 \leq 0 \)
\( f_{\text{min.}} = 1.0 \)

Problem 2. (Circle)
Maximize \( f = \frac{1}{(x_1 + 1)^2 + x_2^2} \)
subject to \( g_1 = x_1^2 + x_2^2 - 4 \geq 0 \)
\( g_2 = 10 - x_1^2 - x_2^2 \geq 0 \)
\( f_{\text{max.}} = 1.0 \)

Ref. Klingman and Himmelblau.
Problem 3.
Maximize \( f = y + \sin x \)
subject to \( 0 \leq x \leq 1 \)
\( 0 \leq y \leq 1 \)
\( x^2 + y^2 \leq 1 \)
\( f_{\text{max.}} = 1.366 \)

Ref. problems 3 through 6 were from Krolak and Cooper as found in the Kringman and Himmelblau paper.\(^6\)

Problem 4.
Maximize \( f = -(y - x)^4 + (1 - x) \)
subject to \( 0.2 \leq x \leq 2.0 \)
\( 0.2 \leq y \leq 2.0 \)
\( x^2 + y^2 \leq 1 \)
\( f_{\text{max.}} = 0.8 \)

Problem 5.
Maximize \( f = -x^2 + x - y^2 + y + 4 \)
subject to \( 0.2 \leq x \leq 2.0 \)
\( 0.2 \leq y \leq 2.0 \)
\( x^2 + y^2 \leq 4 \)
\( f_{\text{max.}} = 4.5 \)

Problem 6.
Maximize \( f = \exp\left(\left((x - 1)^2 - \frac{(y^2 - 0.5)^2}{0.132}\right)\right) \)
subject to \( 0.2 \leq x \leq 2.0 \)
\( 0.2 \leq y \leq 2.0 \)
\( x^2 + y^2 \leq 4 \)
\( f_{\text{max.}} = 1.0 \)

Problem 7. (Lootsma)
Minimize \( f = x_1^3 - 6x_1^2 + 11x_1 + x_3 \)
subject to 
\[-x_1^2 - x_2^2 + x_3^2 \geq 0 \quad x_1 \geq 0\]
\[x_1^2 - x_2^2 + x_3^2 - 4 \geq 0 \quad x_2 \geq 0\]
\[-x_3 + 5 \geq 0 \quad x_3 \geq 0\]

\[f_{\text{min.}} = \sqrt{2}\]

Ref. Lootsma, F. A. {46}

Problem 8.

Minimize
\[\sum_{j=1}^{5} e_j y_j + \sum_{j=1}^{5} \sum_{i=1}^{5} c_{ij} y_i y_j + \sum_{j=1}^{5} d_j y_j^3\]

subject to \[\sum_{j=1}^{5} a_{ij} \geq b_i, \text{ where } i = 1, \ldots, 10\]

\[f_{\text{min.}} = -32.349\]

<table>
<thead>
<tr>
<th>(i)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{ij})</td>
<td>1</td>
<td>-16</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-3.5</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>-9</td>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| \(b_i\) | -40 | -2  | -2.5| -4  | -4  |
|         | -49 | -1  | -60 | 5   | 1   |

| \(c_{ij}\) | 1  | 30 | -20 | -10 | 32  | -10|
|            | 2  | -20| 39  | -6  | -31 | 32 |
|            | 3  | -10| -6  | 10  | -6  | -10|
|            | 4  | 32 | -31 | -6  | 39  | -20|
|            | 5  | -10| 32  | -10 | -20 | 30 |

| \(d_j\)   | 4  | 8  | 10  | 6  | 2  |
|           | -15| -27| -36 | -18| -12|

Ref. Colville, A. R. {65}
RESULTS OF TESTED EXAMPLES:

1. Problem 1

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$ -1.0</td>
<td>0.99999994</td>
<td>-1.00</td>
<td>47</td>
</tr>
<tr>
<td>$x_2$ 2.0</td>
<td>0.99999988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ -1.0</td>
<td>0.99999994</td>
<td>-1.00</td>
<td>36</td>
</tr>
<tr>
<td>$x_2$ 0.0</td>
<td>0.99999988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ 2.0</td>
<td>0.99999994</td>
<td>-1.00</td>
<td>7</td>
</tr>
<tr>
<td>$x_2$ 2.0</td>
<td>0.99999988</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Circle

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>-2.0</td>
<td>-2.00</td>
<td>1.000</td>
<td>265</td>
</tr>
<tr>
<td>-2.8</td>
<td>0.00076647</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>2.0</td>
<td>0.111111</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td>-1.9999857</td>
<td>0.999971</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td>-0.00756613</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>-1.99999914</td>
<td>0.999982</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>0.0060059</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>-2.0</td>
<td>1.000</td>
<td>316</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>-0.000674</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Krolak and Cooper #1

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$ 0.0</td>
<td>0.629556</td>
<td>1.3657408</td>
<td>33</td>
</tr>
<tr>
<td>$x_2$ 0.9</td>
<td>0.776955</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ -0.3</td>
<td>0.63038069</td>
<td>1.3657379</td>
<td>29</td>
</tr>
<tr>
<td>$x_2$ -0.3</td>
<td>0.77628624</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ 0.0</td>
<td>0.63037407</td>
<td>1.3657379</td>
<td>30</td>
</tr>
<tr>
<td>$x_2$ -2.0</td>
<td>0.77629149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ 1.0</td>
<td>0.62933254</td>
<td>1.3657408</td>
<td>53</td>
</tr>
<tr>
<td>$x_2$ -2.0</td>
<td>0.77713621</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1$ 1.0</td>
<td>0.63037902</td>
<td>1.3657379</td>
<td>30</td>
</tr>
<tr>
<td>$x_2$ 1.0</td>
<td>0.77628750</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Krolak and Cooper #2

| $x_1$ -.5     | 0.1999999999     | 0.80000001         | 7                    |
| $x_2$ -.5     | 0.1999999999     |                    |                      |
| $x$ 2.0       | 0.1999999999     | 0.80000001         | 25                   |
| $x_2$ 0.0     | 0.1999999999     |                    |                      |
| $x_1$ 1.0     | 0.1999999999     | 0.80000001         | 29                   |
| $x_2$ 1.0     | 0.1999999999     |                    |                      |
### 5. Krolak and Cooper #3

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.0</td>
<td>0.500000012</td>
<td>4.5000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.9</td>
<td>0.500000018</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>-0.3</td>
<td>0.500000030</td>
<td>4.5000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>-0.3</td>
<td>0.500000030</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>0.0</td>
<td>0.500000006</td>
<td>4.5000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>-2.0</td>
<td>0.500000006</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>1.0</td>
<td>0.500000000</td>
<td>4.5000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.0</td>
<td>0.49999994</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>1.0</td>
<td>0.500000006</td>
<td>4.5000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1.0</td>
<td>0.500000006</td>
<td></td>
</tr>
</tbody>
</table>

### 6. Krolak and Cooper #4

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>-0.5</td>
<td>0.99974561</td>
<td>0.99999994</td>
</tr>
<tr>
<td>$x_2$</td>
<td>-0.5</td>
<td>0.70710695</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>2.0</td>
<td>1.0002337</td>
<td>0.99999994</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.0</td>
<td>0.70710659</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>1.0</td>
<td>1.0000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1.0</td>
<td>0.70711035</td>
<td></td>
</tr>
</tbody>
</table>

### 7. Lootsma

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.37896395</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>1.6807594</td>
<td>1.4142141</td>
<td>1.4142141</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2.3471994</td>
<td>1.4142141</td>
<td></td>
</tr>
</tbody>
</table>
8. Colville 1

<table>
<thead>
<tr>
<th>Initial Point</th>
<th>Computed Optimum</th>
<th>Objective Function</th>
<th>Number of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.0</td>
<td>0.30000037</td>
<td>-32.343491</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.0</td>
<td>0.33695012</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.0</td>
<td>0.40000027</td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td>0.0</td>
<td>0.43677974</td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td>1.0</td>
<td>0.21579641</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX 3: REFERENCES


5. Joint Conference on Mathematical and Computer Aids to Design, Anaheim, California, 1969, Digest Record, Institute of Electrical and Electronics Engineers, 1969 (IEEE Catalogue No. 69C63-C)


7. Naval Electronics Laboratory Center Technical Document 82, Direct-Search Methods For the Solution of the Two-Point Boundary Value Problem [TPBVP], by W. J. Dejka, 30 January 1970

8. Naval Electronics Laboratory Center Technical Note 1417,* Computer Programs to Find the Minimum of a Nonlinear Function of Several Variables, by P. S. Winterbauer, 1 August 1968


*NELC technical notes are informal documents intended chiefly for use within the Center.


29. IBM. Program Information Department Program 3600-15.2.005, Branch and Bound Mixed Integer Programming BBMIP, by R. Shareshian, April 1967


33. IBM Application Program H20-0147-0, 1620 Electronic Circuit Analysis Program [ECAP1 (1620-EF-02X); Application Description, n.d.


42. AEC Research Development Report ANL-5990, Revision 2, Variable Metric Method For Minimization, by W. C. Dav'don, February 1966


