NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPMENT OF A MICROCOMPUTER-BASED
ENGINEERING DESIGN OPTIMIZATION
SOFTWARE PACKAGE

by

Richard L. Booth

December 1983

Thesis Advisor: G. N. Vanderplaats

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Development of a Microcomputer-Based Engineering Design Optimization Software Package		5. TYPE OF REPORT & PERIOD COVERED Engineer's Thesis; December 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard L. Booth		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE December 1983
		13. NUMBER OF PAGES 100
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

Optimization, Microcomputer, Desktop Computer, Engineering Design

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

A general purpose software package was developed to perform nonlinear constrained optimization of user-defined engineering design problems of significant complexity using desktop computers. The package, designated Microcomputer-based Design Optimization Tool (MDOT), will accept nonlinear functions of up to ten variables, which may be bounded, with

DD $_{1\ JAN\ 73}^{FORM}$ 1473   EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

1

as many as fifty constraints. It was implemented on a Hewlett-Packard Model 85 microcomputer with 32 Kbytes of random access memory.

MDOT employs the method of feasible directions for constrained optimization, and a variable metric method for unconstrained functions. It is interactive, provides for monitoring the optimization progress, and can be interrupted to restart from a new point in the design space. Typical applications of MDOT are in the design of machine components composite laminates, and piping systems.

Development of a Microcomputer-Based
Engineering Design Optimization Software Package

by

Richard L. Booth
Lieutenant, United States Navy
B.S.M.E., University of New Mexico, 1977

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER of SCIENCE in MECHANICAL ENGINEERING

and

MECHANICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL

December, 1983

ABSTRACT

A general purpose software package was developed to
perform nonlinear constrained optimization of user-defined
engineering design problems of significant complexity using
desktop computers.   The package, designated Microcomputer-
based Design Optimization Tool (MDOT), will accept nonlinear
functions of up to ten variables, which may be bounded, with
as many as fifty constraints.   It was implemented on a
Hewlett-Packard Model 85 microcomputer with 32 Kbytes of
random access memory.

MDOT employs the method of feasible directions for con-
strained optimization, and a variable metric method for
unconstrained functions. It is interactive, provides for
monitoring the optimization progress, and can be interrupted
to restart from a new point in the design space. Typical
applications of MDOT are in the design of machine com-
ponents, composite laminates, and piping systems.

TABLE OF CONTENTS

5

LIST OF FIGURES

NOMENCLATURE

The nomenclature defined here is that used in the text. Definitions of parameters associated with the program code accompany the code listings in Appendix B. Boldface characters denote vectors or matrices.

a   One-dimensional search step length

D   Inverse Hessian approximation update matrix

F   Objective function

G   Vector of inequality constraints

H   Hessian matrix, or an approximation to its inverse

l   Vector of design variable lower bounds

m   Number of inequality constraints

n   Number of design variables

p   Vector used in constructing **D**

q   Iteration number

S   Search direction vector

s   Scaler used in constructing **D**

t   Scaler used in constructing **D**

u   Vector of design variable upper bounds

w   Scaler used in constructing **D**

X   Vector of design variables

y   Vector used in constructing **D**

∇**F**   objective function gradient vector

ACKNOWLEDGEMENT

I. INTRODUCTION

A. OBJECTIVE

This thesis presents, and describes the development of, a computer software package: "Microcomputer-based Design Optimization Tool" (MDOT). The motivation for this work stemmed from the lack of available general purpose programs capable of performing nonlinear constrained optimization of engineering design problems of significant complexity using desktop computers.

In a more general sense, MDOT is intended to help focus attention on the versatility and computational power of microcomputers. These machines are a potentially very valuable resource which is just beginning to be tapped by the engineering design community.

The remainder of Chapter I is devoted to an overview of where microcomputers stand in engineering design , where MDOT stands amid the optimization software currently available, and the implementation of MDOT. In Chapter II, a general description of optimization concepts and methods, and their application to engineering design, is presented. In Chapter III, the program development of MDOT is described, and flowcharts of the algorithms coded are provided. In Chapter IV, the test problems which were used to validate MDOT are described, along with the solutions

9

obtained. Chapter V is a brief summary. Appendix A is the
MDOT user manual. Appendix B contains an annotated listing
of the MDOT program code.

B. OVERVIEW

There are desktop computers available today with memory
size and computational speed in excess of those of the
mainframes of just a few years ago. The fact that their
capabilities are not yet being fully exploited in the day-
to-day process of engineering design can be attributed in
part to the lack of available software. As Falk [Ref. 1:
p.50] observes, "...engineers...have little time or pa-
tience to do computer programming." Even among those engin-
eers who have the time and patience, there persists a reluc-
tance to program on microcomputers because of a perceived
lack of general purpose utility or under-estimation of the
capability of these machines.

Design optimization is a concept which, similar to the
desktop computer, has received "mixed reviews" from the
engineering disciplines. While there are few who would ques-
tion the virtue of seeking the "best" solution to a problem,
there are many who are reluctant to relinquish to a computer
what they see as the engineers' proprietary decisions in the
design process . Our whirlwind courtship of computer aided
design (CAD) is being tempered somewhat by a counter trend
back toward "human aided design".

It was within the framework of these two ideas that the development of MDOT was undertaken; not only to make available a useful interactive design optimization program, but to demonstrate that a powerful general purpose problem solver can be implemented in a microcomputer. Little knowledge of programming is required of the MDOT user. Problem entry and program execution are convenient. The interactive features of the code permit the design engineer to keep in close touch with the progress of the problem solution and to interrupt program execution to make parameter adjustments based on engineering judgement.

The chief advantage that microcomputers enjoy over mainframes is their low cost. Small computers are typically purchased outright, so that their use incurs no additional expense for connection or run time. Their major disadvantage is comparatively slow computational speed, but it is doubtful that engineering design ever progresses so rapidly as to make it imperative that a solution be obtained in seconds rather than minutes. In any case, this disparity is rapidly disappearing with the development of 16 and 32-bit microprocessor-based desktop computers [Ref. 2: p.2].

C. OPTIMIZATION SOFTWARE CURRENTLY AVAILABLE

There are several powerful general purpose optimization programs available, such as COPES/CONMIN [Ref. 3], which can deal with a wide range of design problems. These programs

must reside in a mainframe computer, and their use can be cumbersome, especially for the occasional user. At the other extreme are those codes developed for use in computers with limited memory. Typically these are special purpose programs employing zero order or simple first order methods, such as random search or steepest descent, capable of handling only relatively small problems. They are convenient, but of limited usefulness.

The gap between these two categories requires that optimization of the great number of general design and analysis problems which are on a scale that could easily be handled by small computers be done on a mainframe or not at all. This often leads to overmodeling, wherein a relatively simple problem is unnecessarily made more complicated in order to more fully utilize the machine capability or to justify the expense of computer services. MDOT was developed specifically to bridge this gap.

MDOT provides the design engineer with a convenient tool for optimization of nonlinear problems in up to ten bounded independent variables subject to as many as fifty inequality constraints. All that is required is access to a desktop computer, and today there are certainly few engineers who lack this.

D.   IMPLEMENTATION OF MDOT

The program development for MDOT was done on a Hewlett-Packard model 85A microcomputer, which is built around an 8-bit microprocessor. This particular machine is so often used as a data aquisition system controller that its stand-alone computational capability may frequently be overlooked. A versatile, engineering oriented computer with high machine precision, it is nonetheless on the low end of the memory size scale with just 16 Kbytes. The computer's capabilities were enhanced by the addition of a 16 Kbyte memory extension module and three read-only-memory modules: the matrix, advanced programming, and printer/plotter ROMs. As configured, there were just over 30 Kbytes of memory available for programming.

MDOT was written in HPBASIC, which differs in some respects from standard BASIC. As such, in its present form MDOT is limited to use in the HP series 80 computers. Without much difficulty, though, the code could be translated and run on almost any available hardware. Additional comments concerning transferrability of the code are presented in chapter III.

II.   OPTIMIZATION

A.   APPLICATION TO ENGINEERING DESIGN

Design has been described as the creative process through which the engineering profession develops devices, processes and systems to fill the needs of man [Ref. 4: p.170]. A "need" must first be defined in terms of specific requirements which the design must meet. Then the engineer, drawing on available resources, synthesizes proposed solutions to meet these requirements. Many such designs may, and usually do, exist. Thus there arises the subproblem of finding the best of these designs, and the inherently iterative nature of design.

Traditionally, the solution to this subproblem was sought through comparitive analysis of a reasonable number of alternative designs; a tedious and expensive procedure for problems of even moderate complexity. The recent development of a broad range of very useful CAD software has made it possible to remove a good deal of the tedium, and, perhaps to a lesser degree, the expense of engineering design. For the most part, though, these tools have made no fundamental change in the approach taken to solve the design problem. What they have done is redefined the phrase "a reasonable number of alternative designs". By programming

the analysis and comparison tasks into a computer, the engineer is able to consider many more possible solutions in the same amount of time.

Optimization methods are a significant extension of the CAD concept in that they enable the engineer to exploit the capabilities of the computer over the entire scope of the design process. In optimization, the computer is tasked not only with analysis and comparison of previously selected designs, but with selection of the designs to be considered in subsequent iterations as well. Since this intermediate design selection can be quite complex, closing the design loop in the computer can lead to a considerable savings of time and effort in the search for the optimum.

B.   THE NATURE OF THE PROBLEM

Fundamental to the economical solution of the design problem is that it be quantified and formulated mathematically to permit conceptual, rather than physical, manipulation of resources. The design, then, is specified by assigning values to a set of independent variables which represent its physical characteristics. The measure of goodness of the design used for comparison with others is expressed as some functional relationship between these variables. The requirements placed on the design, as well as the physical limitations of the design itself, define a region in the multi-dimensional mathematical design space. The design must

reside inside this region to be acceptable. In the termin-
ology of optimization, the measure of goodness is the
objective function, the requirements and limitations are
constraints, and designs which fall within the region
bounded by the constraints are considered feasible. As
examples, the set of independent design variables might be
the cross-sectional dimensions of a structural element, the
objective function its weight, and the constraints its
maximum allowable stress and size limitations.

In general, the formulation of a design problem leads to
an objective and a number of constraints, all of which may
be linear or nonlinear functions, explicit or implicit in
many design variables which themselves are subject to limi-
tations, called bounds or side constraints. Stated mathe-
matically [Ref. 5: p.9], the design optimization problem is
to

Minimize: $F(X)$ objective function

Subject to: $G_j(X) \leq 0, \quad j=1,m$ inequality constraints

$l_i \leq X_i \leq u_i \quad i=1,n$ side constraints

Where: $X^T = \{X_1, X_2, \ldots X_n\}$ design variables

$l^T = \{l_1, l_2, \ldots l_n\}$ lower bounds on X

$u^T = \{u_1, u_2, \ldots u_n\}$ upper bounds on X

If in the problem formulation it is more convenient to
define the objective function as a quantity which is to be

16

maximized, such as efficiency or torque, then the above statement may simply be modified to read "Minimize: -F(X)".

C. THE NATURE OF THE SOLUTION

Optimization is an application of mathematical theory concerning identification of the extrema of functions. In multivariable calculus, for example, the method of Lagrange multipliers is developed, which provides a closed form solution for the extremum of a constrained function. While useful for demonstrating concepts and developing methods, such analytical techniques are not practical for solving any but the simplest of problems. Design optimization methods involve numerical approximation techniques and iterative search schemes. They are ideally suited to, and in fact made practical only through, the use of digital computers.

Many optimization algorithms have been developed around widely varying strategies. Common to most are the three basic tasks that make up one iteration of the solution loop:

1. Selection of a direction in the design space along which to search.

2. A search for the most improved design in this direction.

3. Convergence testing to determine when the optimum design has been found.

For unconstrained problems, these tasks are relatively straightforward. Addition of a constraint set may, depending on the sophistication of the method employed, complicate the first two steps considerably.

Except in the case of zero order methods, selection of a search direction involves calculation of partial derivatives, for which general purpose optimizers use numerical techniques, such as finite forward differences. At any point in the design space, the negative of the gradient of the objective function indicates the direction in which the objective function is most rapidly decreasing. This may not be the best direction in which to search, however, if the objective function is highly nonlinear or if the design is near one or more constraints. Efficient algorithms variously employ constraint gradients, Hessian matrix approximations, and previous iteration history information in addition to objective function gradients to select the search direction.

Finding the best improved design along a line in the specified direction is termed a "one-dimensional search", because the objective and constraints are treated as functions only of the "distance" along this line from the current design point. Techniques employed in the one-dimensional search include the golden section and Fibonacci methods, polynomial approximations, and combinations of these [Refs. 5,6]. If constraints are present, the best improved design may not be the point on this line at which

the objective function is minimized. If a constraint is violated, the design is infeasible, so the search algorithm must seek the point at which the objective function is minimized while remaining inside the feasible region.

Part of the optimization problem formulation is specification of an initial design point from which to start the solution process. For constrained problems, the possibility exists that this initial design will be infeasible. To provide for this, the search direction routine must find the direction which will yield the shortest path to the feasible region, and the one-dimensional search algorithm must allow for the possible necessity of increasing the objective function in order to attain feasibility.

Convergence to a global optimum generally cannot be guaranteed. Theory provides the Kuhn-Tucker conditions necessary for the existence of an optimum, but these are neither convenient to evaluate nor sufficient to define optimality [Ref. 5: pp. 17-20]. In practice, convergence is typically considered to be indicated by one or more of the following:

1. Failure to find a search direction which will lead to an improved design.

2. Given a direction, failure to find any significant search step length to improve the design.

3. Finding no appreciable design improvement over a specified number of iterations.

If the possibility of local minima exists, the optimization should be repeated from several different initial designs. For constrained problems, the optimization process may fail to find any feasible solution, in which case the problem must be reformulated.

An iteration in the optimization solution, then, may be summarized. Beginning from the current design point X^q, a search direction, S^q, is determined. Then the one-dimensional search is conducted to find the "distance", a, along S^q, which yields the best improved design. The design is then updated as

$$X^{q+1} = X^q + aS^q$$

at which point the objective function is reevaluated and the design checked for convergence.

Optimization algorithmic efficiency and convergence behavior are affected by the mathematical characteristics of the problem. As numerical methods, they are susceptible to ill-conditioning. Truncation and round-off errors, which are an unavoidable consequence of the use of digital computers, aggravate this. Given an optimizer suitable to the problem type, careful problem formulation, as discussed in Appendix A, is the best insurance against poor optimizer performance.

III. PROGRAM DEVELOPMENT

A. BASIC CRITERIA

At the outset of the program development, four basic criteria were established to be met by MDOT: utility, minimization of required memory, user convenience, and reduction of problem run time. At points where conflict existed between them, these criteria were prioritized in the order listed. Few such compromises were necessary, as the requirements were found to be generally complimentary.

The utility criterion meant that MDOT should be a general purpose optimizer which could be applied to a wide range of design problems. Minimization of memory was dictated by the limitaions of microcomputers, and affected not only algorithm selection and problem size, but many aspects of the actual coding as well. User convenience considerations drove the development of those portions of the code which are interactive, and those involved with problem entry and output options. Reduction of problem run time was a factor throughout the development, most notably in the incorporation of optimization progress display and the interrupt/restart option.

B. MDOT ALGORITHMS

1. Algorithm Selection

Of the many optimization algorithms available, the zero order methods, as well as the simpler of the first order methods, were ruled out on the basis of their lack of general purpose utility. Others, including linear and quadratic programming types, were eliminated because of their excessive memory requirements [Refs. 7,8]. Finally, the need to reduce problem run time while retaining utility lead to the selection of two algorithms, each capable of nonlinear multidimensional optimization. The first is a variable metric method which is used in MDOT for unconstrained optimization, the second is a method of feasible directions, for minimizing constrained functions.

The selection of the one-dimensional search strategy to be employed was driven by the need to reduce problem run time. There is a trade-off to be made between the precision to which the search step length determination is made and the time required for each optimization iteration. Both the golden section and Fibonacci search methods can attain very precise solutions, but to do so they become computationally expensive. In MDOT the one-dimensional search routines were designed to seek a less precise step length solution in order to complete each iteration more quickly. The method employed in both the optimizers estimates an initial step length based on a reasonable change in objective function

magnitude. The golden section ratio is then used to establish bounds on the solution, which is finally refined by polynomial approximation.

2. The Unconstrained Optimizer

From the derivation of the Kuhn-Tucker conditions, it is known that if at some point X^* the objective function $F(X^*)$ has a local minimum, then the gradient of the objective at this point, $\nabla F(X^*)$, must vanish and the Hessian matrix H must be positive definite. Combined with a second order Taylor series expansion of $F(X)$ about some point, say X^O, near the minimum, these conditions lead to an expression for the direction from X^O to X^* as

$$X^* - X^O = -H(X^O)^{-1} \nabla F(X^O)$$

In practice, determination of the Hessian matrix by finite difference approximation, as well as inversion of the matrix, would be computationally so expensive as to outweigh the theoretical gain in algorithmic efficiency.

In variable metric methods, information gathered as the optimization progresses is used to develop an approximation to the inverse of the Hessian matrix, which is then used in determining the search direction. As such, these first order methods have some convergence characteristics comparable to those of second order methods. The algorithm for the variable metric method is shown in Fig. 1.
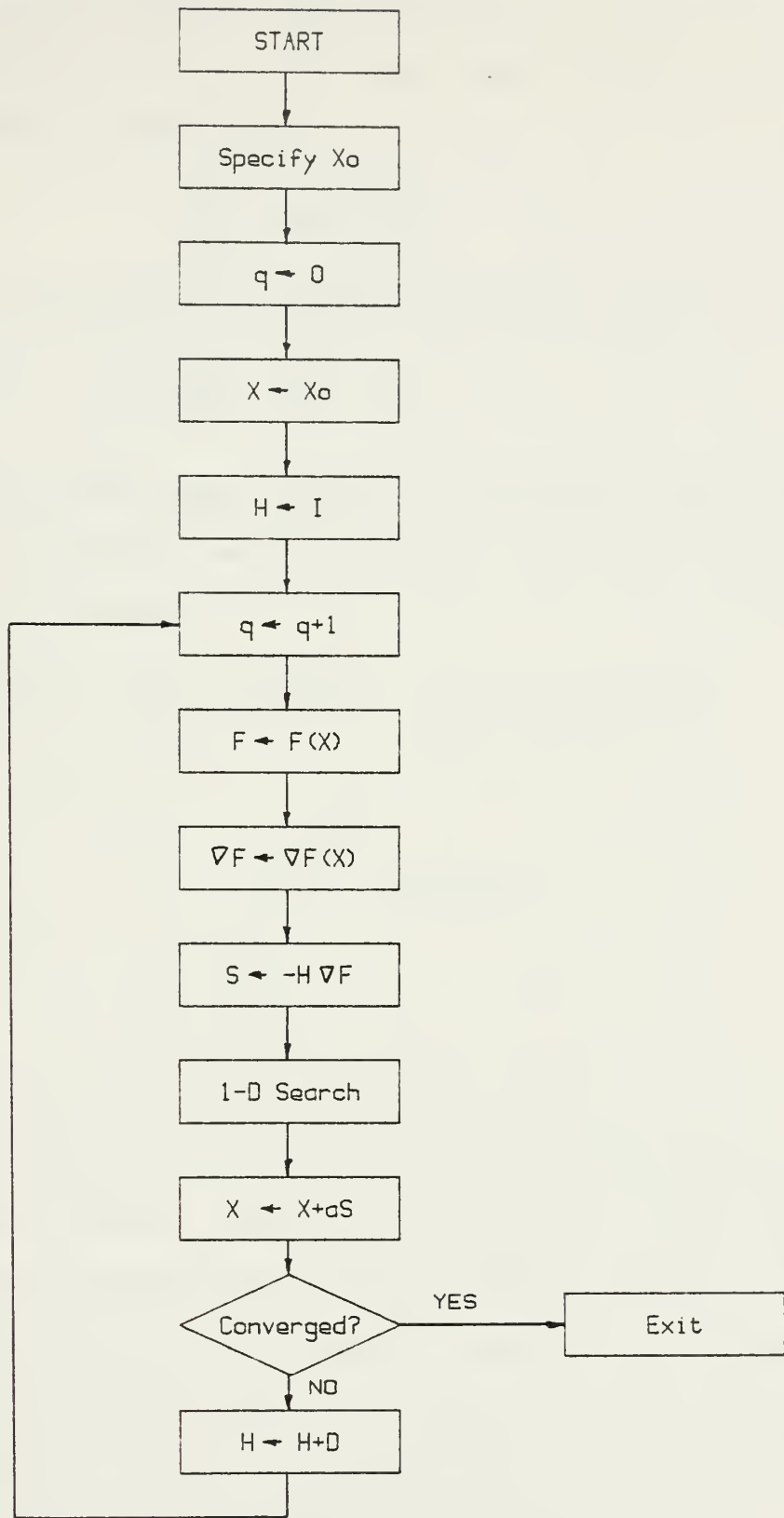
23

Fig. 1 Algorithm for the Variable Metric Method

24

The inverse Hessian approximation is initialized as an n x n identity matrix. To begin each iteration, the search direction is defined as

$$S^q = -H \nabla F(X^q)$$

After the one-dimensional search and design update, H is modified as

$$H^{q+1} = H^q + D^q$$

where the form of the update matrix D determines which one of a family of variable metric methods is being used. In general, D is defined as

$$D^q = \frac{s+wt}{s^2} pp^T + \frac{w-1}{t} H^q y(H^q y)^T - \frac{w}{s}[H^q yp^T + p(H^q y)^T]$$

where: $\quad p = X^q - X^{q-1}$

$\quad y = \nabla F(X^q) - \nabla F(X^{q-1})$

$\quad s = p \cdot y$

$\quad t = y^T H^q y$

Two forms of D, and thus two variable metric methods, are available in MDOT. The first is the Davidon-Fletcher-Powell method, where w is set equal to zero. The second is the Broydon-Fletcher-Goldfarb-Shanno method, with w equal to one [Ref. 5: pp. 92,93]. As the convergence behavior of a given algorithm can be somewhat problem dependent, this feature allows the MDOT user to compare the results of two variations of unconstrained optimization.

The one-dimensional search routine employed by MDOT in the variable metric optimizer first finds bounds on the unconstrained minimum of the objective function, then refines the minimum by a three-point cubic polynomial approximation.

3. The Constrained Optimizer

The addition of a set of constraints to the optimization problem requires that more sophisticated techniques be applied to its solution, particularly in the determination of a search direction and the subsequent one-dimensional search. As is the case for unconstrained optimizers, it is generally the method used to find a search direction which distinguishes the different constrained optimization algorithms. In the method of feasible directions, a search direction in which a finite step will reduce the objective function is termed useable, while one which will avoid constraint violation is called feasible. The direction finding problem is then formulated as a sub-optimization task to determine the best of the possible useable-feasible directions. MDOT employs the algorithm presented by Vanderplaats [Ref. 5: pp. 163-170] for the solution of this sub-problem.

As shown in the flowchart of Fig. 2, the feasible directions optimizer begins as a simple steepest descent algorithm, provided the initial design is feasible. Optimization thus proceeds quickly to a point where one or more constraints are encountered.
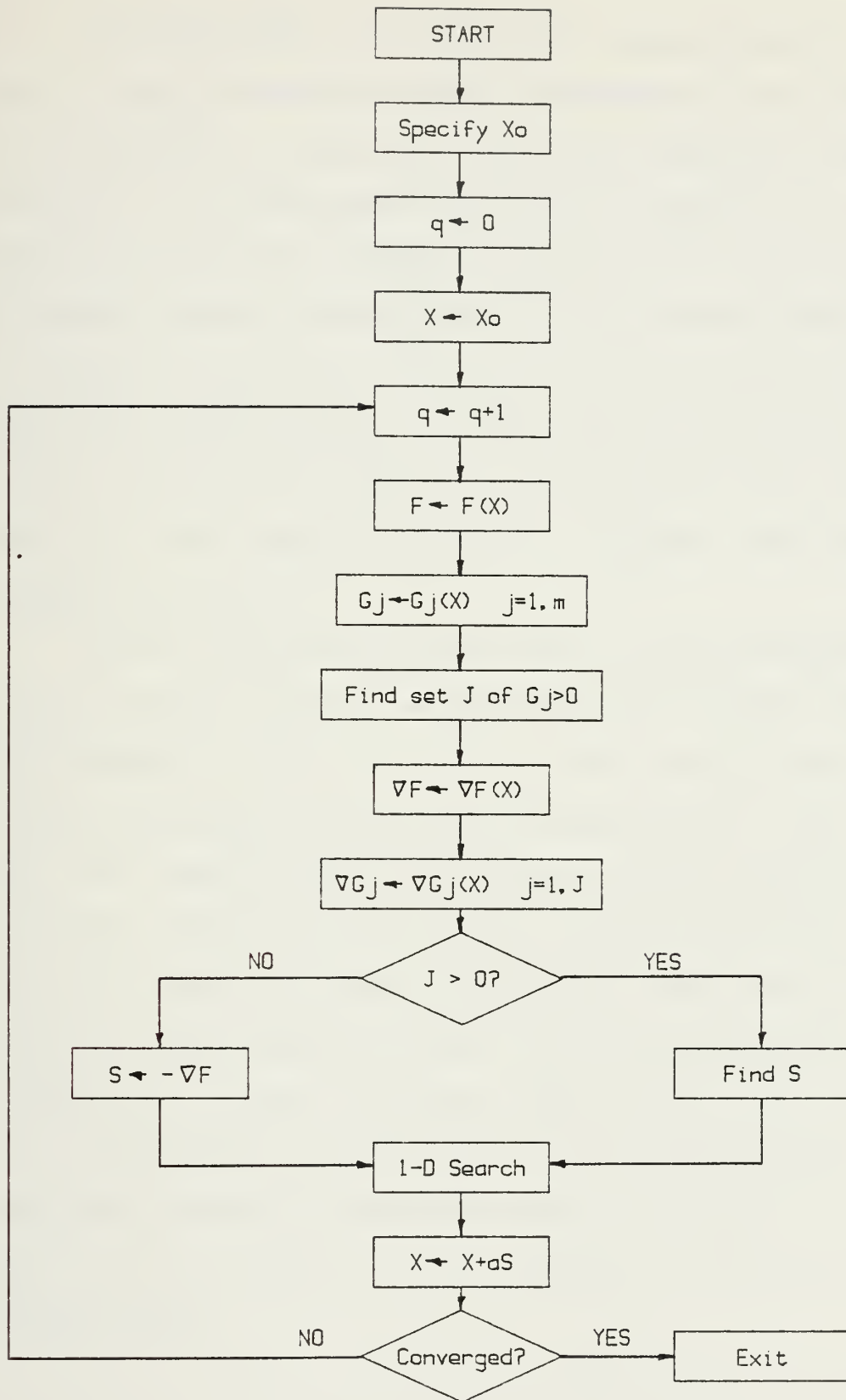
Fig. 2 Algorithm for the Method of Feasible Directions

27

On subsequent iterations the sub-optimization routine is used to determine search directions so as to satisfy the fundamental requirement that the optimum design be feasible. The constrained optimizer in MDOT will accept initially infeasible designs. In this case, the direction and search routines are modified so as to attain feasibility as quickly as possible. Thus a feasible direction and step length are sought which will overcome the constraint violations, even at the expense of increasing the objective function. Once inside the feasible region, optimization proceeds as before.

As in the unconstrained case, in the one-dimensional search routine employed in MDOT for constrained optimization, bounds on the solution are first established, followed by refinement by polynomial approximation. Here, however, the search must be conducted for the zeros of the constraint functions as well as for the minimum of the objective. The step length selected is then the one which yields the best feasible design. Provision must also be made to ensure the design variables remain within their bounds (side constraints). In MDOT, if at any time during the one-dimensional search a design variable is found to exceed an upper or lower bound, it is set equal to the value of the violated bound.

4. Program Logic

The relationships between the modules of MDOT are depicted in Figs. 3 and 4 for unconstrained and constrained optimization, respectively. The main program is named "Autost" because this signals the HP-85 operating system to load and run this program automatically when the computer is powered up with the mass storage cartridge inserted in the tape drive. All the other modules of MDOT are subprograms which are called into memory and executed as needed by Autost or another subprogram. Once entered into main memory, a subprogram resides there for the duration of the optimization unless a "SCRATCHSUB" instruction is executed.

Following is a brief description of the function of each of MDOT's program segments:

|	
Autost	MDOT main calling program
LOGO	Displays introductory (welcome) graphic
DEFALT	Sets program parameters to default values
PROB	Problem entry, evaluation of F and G
CCONT	Control of constrained optimization
UCONT	Control of unconstrained optimization
ACON	Identification of active/violated constraints
GRAD	Calculation of gradients of F and G
DIRECT	Direction finding subproblem solver
FDSRCH	Constrained one-dimensional search
VMSRCH	Unconstrained one-dimensional search
NEWH	Update approximation to H^{-1}
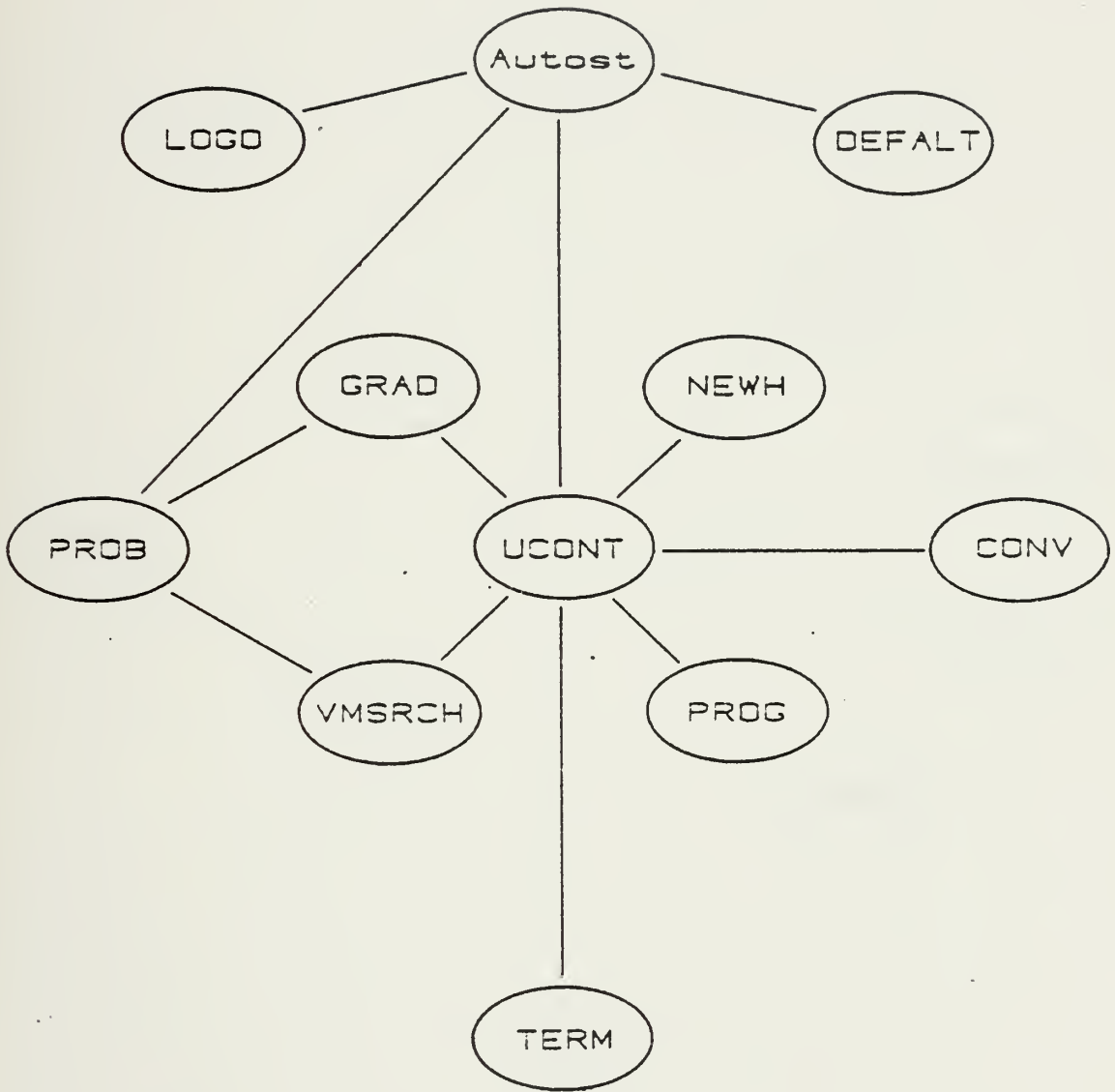
Fig. 3   MDOT Organization - Unconstrained
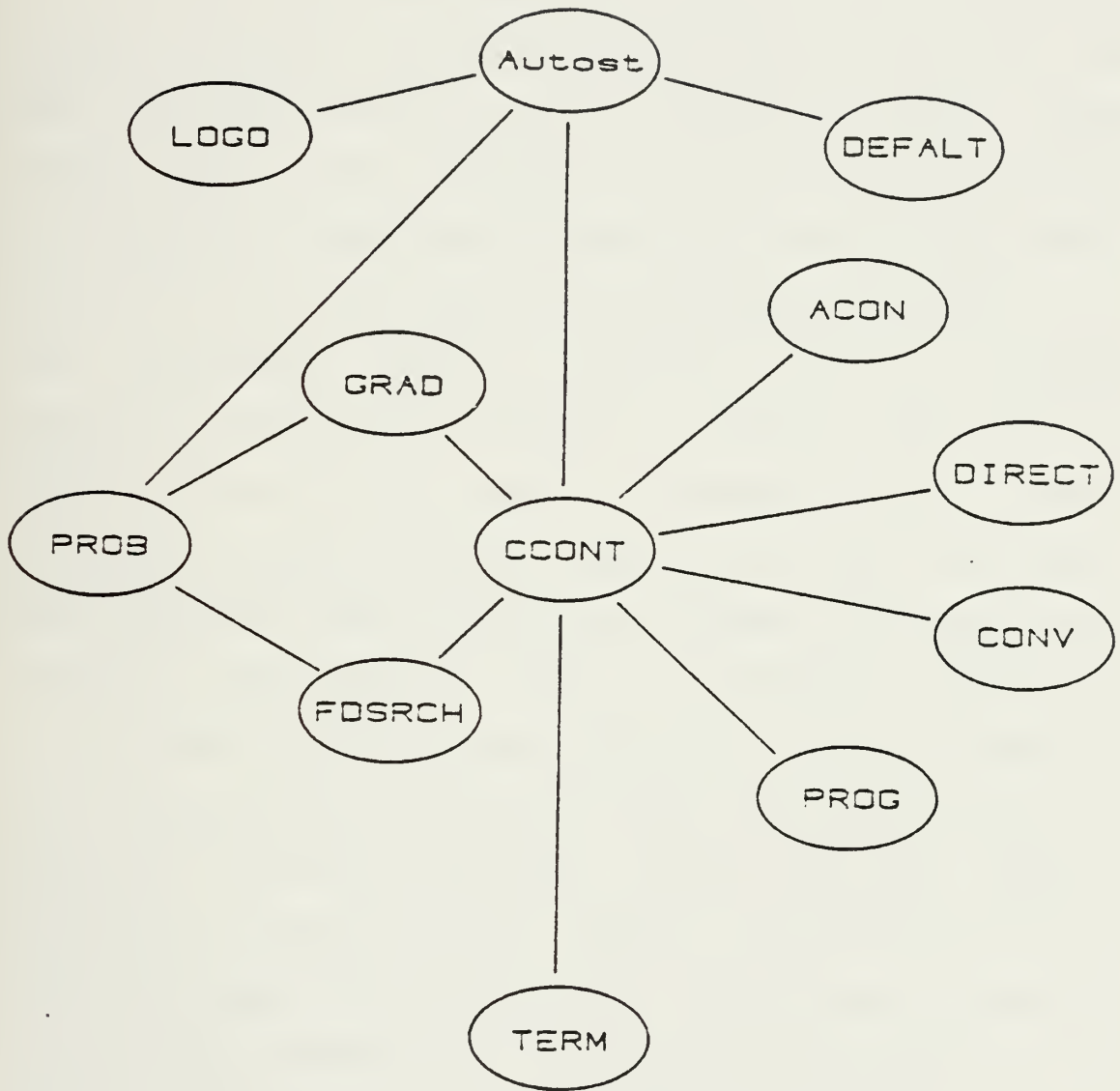
Fig. 4   MDOT Organization - Constrained

CONV	Convergence testing
PROG	Optimization progress information
TERM	Output of results of the optimization

The PROB subprogram is created by the user by editing a skeleton problem entry code which is stored on the tape. The edited version is then renamed and stored. Autost queries the user for the problem name, which is then common to all subprograms. Autost makes the first call to PROB in order to select the appropriate optimization control routine. Thereafter, PROB is called any time an objective function or constraint evaluation is required. Both LOGO and DEFALT are called by Autost. LOGO generates a simple "welcome" graphics display, and is scratched from memory upon execution. DEFALT initializes a number of program parameters to their default values, as defined in Appendix B.

PROG is called upon completion of each iteration. This subprogram generates the user selected optimization progress indicators. Options include data and graphics displays and printed output. Based on the progress information provided, the user may elect to continue the optimization, restart MDOT from a different initial design, or stop and reformulate the problem. CONV is also called upon completion of each iteration, to determine, based on the convergence criteria set by DEFALT, whether the optimum design has been found.

TERM is called to end the optimization and generates the output of results. Termination may be invoked by a number of conditions in other than meeting convergence criteria. If the optimizer exceeds a specified number of iterations, if the components of the search direction vector are all essentially zero, if no search step length can be found to improve the design, or if there are an excessive number of violated constraints, MDOT will terminate. In any case, TERM will generate an output message to indicate the condition upon which the decision to terminate was based, and offer the user the option of editing and restarting the program.

C. ADAPTATION OF MDOT TO OTHER SYSTEMS

Since MDOT is coded in HPBASIC, it is not immediately transferrable to hardware other than the Hewlett-Packard series 80 desktop computers. Translation of the package, either into another version of BASIC or into FORTRAN, is certainly a "do-able" project which would significantly expand the applicability of MDOT.

This section highlights those features of HPBASIC used in MDOT which would have the greatest impact on this project. They are: variable name assignment, SUBPROGRAMS, matrix manipulations, and graphics.

The limitation of HPBASIC which most decreases the readability of the code is that variable names are restricted to

either one letter or a letter followed by a digit. One result of this is that arrays of subscripted variables are sometimes used where individual characteristic names might otherwise be assigned. A feature of the language which helps considerably, not only to overcome this limitation, but in programming complicated algorithms, is the SUBPROGRAM.

Similar to a SUBROUTINE in FORTRAN, the SUBPROGRAM is called when needed and variables may be passed either by name or by value. In HPBASIC this allows for the use of the same variable name to denote different parameters in separate program segments. MDOT makes extensive use of SUBPROGRAMs. This feature is not available in all versions of BASIC. Without it, the translation of MDOT would be more difficult, but still possible through the use of functions and subroutines, particularly if multi-character variable names are permitted.

Matrix manipulation is convenient in HPBASIC. Operations such as matrix multiplications, transpositions, dot products, and identifying extreme array elements are all accomplished through simple "MAT" statements. This feature is available in some of the other versions of BASIC, but not in FORTRAN. Without it, additional subprogramming would be required to perform these operations.

Graphics capabilities vary widely from one hardware manufacturer to another, as do the coding instructions used to execute the displays. It is likely that the graphics pro-

grammed into MDOT would require major modification to make them transferrable.

Two more details of the HPBASIC code are worthy of note: program flags and the @ symbol. Program flags are built-in indicators which can be set to 1 or cleared to 0, and are used in MDOT for conditional branching decisions. They could easily be replaced by integer variables. The @ symbol is used to condense the code and thus conserve memory. It simply separates multiple executable statements on one program line. Without this feature, each statement must have its own line number.

D. POTENTIAL FOR FUTURE GROWTH

Besides its obvious potential for expanded problem size if implemented in a computer with a larger memory, there are many refinements and additions which could be incorporated into MDOT, either to enhance its general purpose utility or to tailor it to a particular type of problem. Some modifications for improved utility might involve coding additional algorithms, automatic design variable scaling, and the handling of equality constraints.

MDOT could be customized by modification of the problem input subprogram, graphics display, output format, or the algorithms themselves. Coupling of MDOT to an external CAD or analysis code also presents many interesting and potentially useful possibilities. One such configuration might be

to use MDOT as a subprogram called to perform optimization on localized aspects of a large scale design problem in conjunction with a desktop computer CAD system [Ref. 9].

IV. TEST CASES

Validation of an optimization program typically consists of testing it on a battery of representative problems to which the solutions are known [Ref. 10]. Based on the results of such tests, a number of yardsticks exist by which the optimizer is judged relative to others. These can be grouped into three categories: stability, robustness, and efficiency [Ref. 8: p.75].

An optimizer is stable if, once a feasible design is attained, the objective function remains non-increasing until the optimum has been found. A robust optimizer is one which yields a valid solution given a poor initial approximation. Efficiency refers either to the number of function and derivative evaluations required in the solution or to the problem run time. These two measures of efficiency are closely related if comparing different optimizers run on the same machine, since function and derivative evaluations are typically costly operations.

To these measures of an optimizer's performance, a fourth category should be added; that of utility. Given a stable and robust optimizer, there are characteristics in addition to its efficiency which should be considered in determining its utility. Problem size and type solvable by

the program are of fundamental importance, as is memory
storage required to run it. There are trade-offs involved
here in program development between the sophistication of
the algorithms used, speed of convergence, hardware
capabilities, cost of run time, and user convenience. The
utility of the optimizer is an indication of how these
trade-offs were made, and involves much more than just
efficiency.

A. UNCONSTRAINED TEST PROBLEM

Among the unconstrained test problems run on MDOT was
the so-called "banana" function:

$$F(\mathbf{X}) = 10X_1^4 - 20X_1^2X_2 + 10X_2^2 + X_1^2 - 2X_1 + 5$$

which has an optimum of $F(1,1)=4.0$. This function derives
its name from the shape of the contours of constant objec-
tive function (Fig. 5). Although only two-dimensional, the
banana function is a good test of an unconstrained optimizer
because the objective function surface becomes a steep,
narrow, curved "valley" as the optimum is approached. An
inefficient optimizer will tend to "zig-zag" in such a
design space, resulting in slow convergence near the solu-
tion, while a non-robust optimizer will tend to terminate
prematurely.
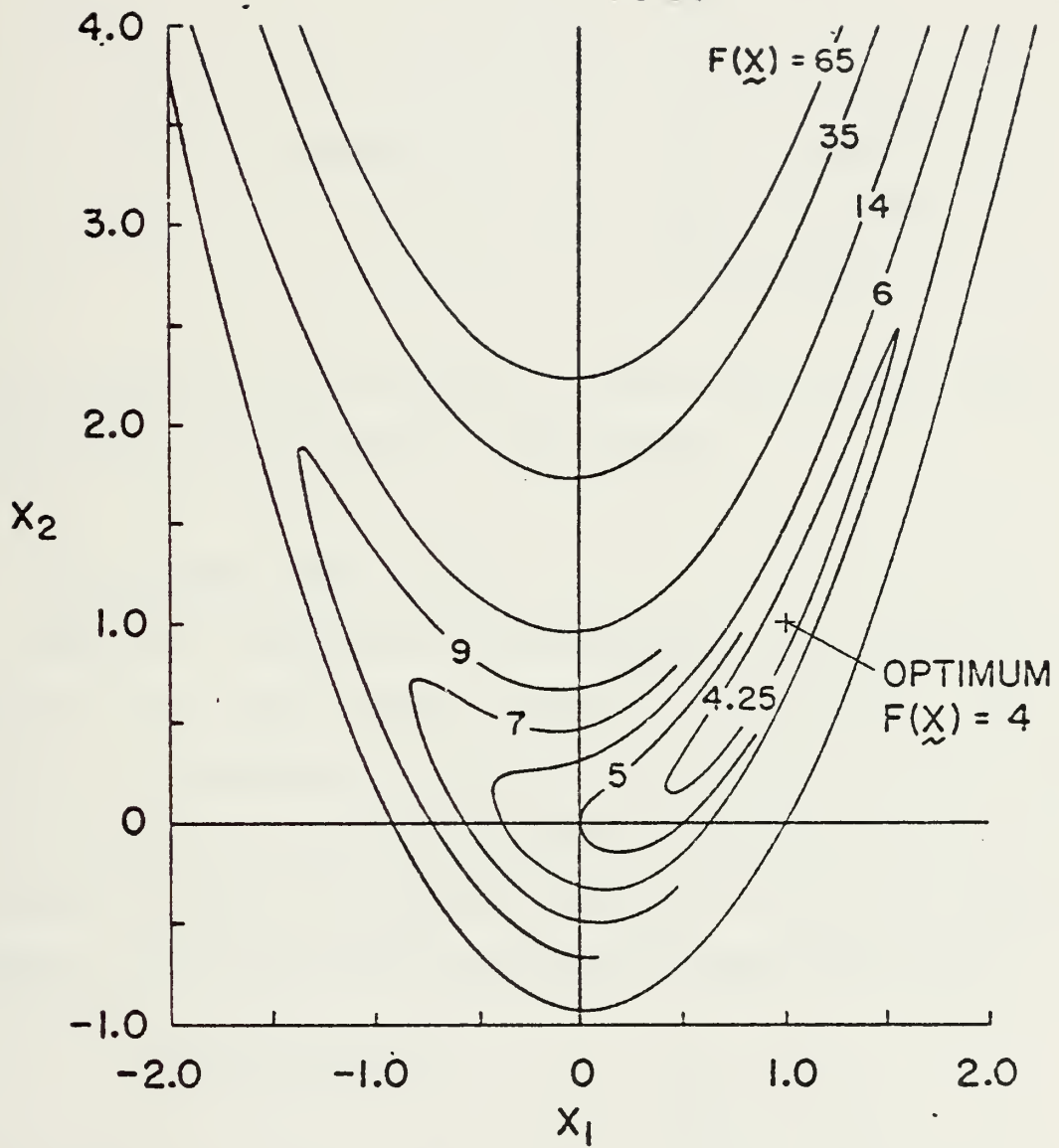
Fig. 5 Unconstrained Test Problem Design Space

Results of the performance of MDOT on the banana function are summarized below.

Initial design: $X_1{}^o = -1.0$

$X_2{}^o = 1.5$

$F^o = 10.5$

Optimum: $X_1{}^* = 0.95979$

$X_2{}^* = 0.91442$

$F^* = 4.002$

MDOT arrived at this solution in about one minute, after 12 iterations and 66 function evaluations.

B. CONSTRAINED TEST PROBLEM

Among the constrained test problems run on MDOT was the cantilevered beam problem posed by Vanderplaats [Ref. 3: p.8], as illustrated in Fig. 6. The objective function in this case is the volume of the beam, for which a theoretical optimum of 6603.9 is known. Results of the performance of MDOT on this beam design problem are summarized below.

Initial design: $X_1{}^o = 3.5$

$X_2{}^o = 16.0$

$F^o = 11200$

Cantilevered Beam:
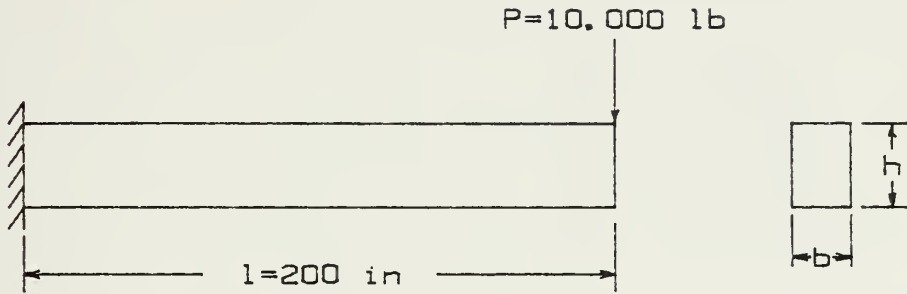
Design Variables:	b, h
Objective Function:	Volume
Constraints:	Bending Stress < 20,000 psi Deflection < 1.0 in Ratio of h to b < 10.0 0.5 < b < 5.0 1.0 < h < 20.0
Where:	Bending Stress = Mc/I Deflection = $Pl^3/3EI$

Fig. 6   Constrained Test Problem

$$\text{Optimum:} \qquad X_1{}^* = 1.8261$$

$$X_2{}^* = 18.174$$

$$F^* = 6637.5$$

$$G_1{}^* = -0.024$$

$$G_2{}^* = -0.00656$$

$$G_3{}^* = -0.00522$$

MDOT arrived at this solution in about two minutes, after 11 iterations and 56 function evaluations.

With refinement of the algorithms, improvement could likely be realized in the program performance. All variables in MDOT are declared "SHORT", which in HPBASIC means they are carried to 5 digits. In a machine with just 64 Kbytes of memory, this could be changed to "REAL", in which case 9 digits would be carried, with an attendant improvement in the precision of the solution. The number of function evaluations, and thus the problem run time, could be decreased by modification of the algorithms such that gradients are not calculated in every iteration. Also, the efficiency of the one-dimensional search routines could be improved by distinguishing between linear and nonlinear constraint functions.

V. <u>SUMMARY</u>

Optimization is a useful tool in engineering design. The desktop computer is the vehicle through which this tool can be made widely available, convenient, and inexpensive. The development of MDOT affirms the feasibility of implementing a powerful general purpose optimization algorithm in a computer with limited memory.

The applicability of MDOT could be expanded through conversion to standard BASIC or translation to FORTRAN. It has potential for growth in terms of versatility and problem size, and lends itself to tailoring to suit a particular class of problem. MDOT could be coupled with a microcomputer CAD package to close the design loop in the computer.

MDOT has been validated by tests on a number of problems, both constrained and unconstrained. Its performance is good, and could be made better through refinement of the algorithms. Specific modifications might involve the one-dimensional search routines and the frequency of gradient calculations.

As microcomputers continue to become more commonplace and their capabilities continue to improve, emphasis will shift away from the mainframes for the solution of problems which are on a scale easily handled by smaller machines. Software such as MDOT will both accompany and encourage this shift.

APPENDIX A
<u>MDOT</u> <u>USER</u> <u>MANUAL</u>

1.   INTRODUCTION

To avoid repetition, references to material presented in the preceding chapters are made in this appendix. A useful follow-on project would be to assemble a user manual for MDOT independent of the background and developmental material in the body of the thesis. Details of computer operation have not been included here, as it is assumed that the user is either familiar with the machine or has access to the operating manual.

MDOT is currently available only on magnetic tape cartridge for use in Hewlett-Packard series 80 computers. If it is to be implemented in an HP-85A, the machine must be configured with four enhancements: a 16 Kbyte memory extension module, a matrix ROM, advanced programming ROM, and a printer / plotter ROM. No peripheral devices are required, nor is extensive programming.

2.   PROBLEM FORMULATION

Formulation of a well-posed problem, as discussed in chapter II, is fundamental to the satisfactory performance of an optimization program. First, the design variables must be identified. These are the parameters of the problem which

the optimizer will be permitted to change in its search for the best design. The objective must then be a function of these variables, and the minimum of this function is what the optimizer will seek. Constraints may be imposed on the design in two ways:

1. Upper and/or lower bounds (side constraints) may be specified for any of the design variables.
2. General inequality constraints may be expressed as functions of the design variables.

Side constraints are explicitly assigned when the initial design estimate is entered into the problem subprogram. Inequality constraints must be formulated as quantities which are to be less than or equal to zero. Care should be exercised to avoid redundant or otherwise unnecessary constraints. In MDOT, an unconstrained problem has neither side constraints nor inequality constraints. MDOT has no provision for equality constrained problems.

The objective function can be any characteristic of the design expressible mathematically in terms of the design variables. It is important to keep in mind that it is the minimum of this function which is sought. If the problem is formulated around an objective which is to be maximized, then it must be entered in such a way that MDOT will seek to minimize the negative of this objective.

An important consideration during problem formulation is the range of orders of magnitude of the design variables. Optimizer performance is best in a design space in which the contours of constant objective function are concentric hyperspheres, such that a given change in one variable has the same effect on the objective as an equal change in any other variable [Ref. 11, p.17]. In practice, this is approximated by scaling the design variables such that they are all of the same order of magnitude, or nearly so. Some optimizers do this automatically, MDOT does not.

Selection of the initial design point from which to start MDOT will affect its performance and problem run time. Any available information which will improve the initial approximation should be used. If a constrained problem is being entered, a check should be made to ensure the initial design falls within the side constraints. Although MDOT is equipped to handle initially infeasible designs, convergence will likely be more rapid if the initial design is free of violated inequality constraints. In some cases, initial feasibility may be a difficult thing to build into the problem formulation. MDOT will display the results of the first design evaluation and indicate whether or not it is feasible. At that time, the user may elect to proceed with the optimization or edit the initial design and restart the program.

3.   PROBLEM ENTRY

MDOT problem entry is accomplished by editing the PROB subprogram. With this module loaded into memory and listed on the CRT, modifications are made on the program lines noted below.

Line 10:

> The file name of the subprogram is changed to any name up to 6 characters in length, except any of those already assigned to MDOT files.

Line 100:

> Just after the word DATA, two integers are added, separated by a comma. The first is the number of design variables, (N1), the second is the number of inequality constraints, (N2). For unconstrained problems N2 is always zero.

Lines 201-210:

> Initial values of the design variables are entered, beginning with X1 on line 201 and continuing, one variable per line. If the problem is constrained, each initial value is followed by the lower and upper bounds assigned to the corresponding variable. If no bound is to be specified, the field is filled by an "N".

Lines 231-398:

These lines are available for defining expressions
to be used in the design evaluation. These state-
ments will be executed prior to each objective or
constraint computation. Whenever the design vari-
ables are used in this, and the remaining sections
of the subprogram, they are expressed as X1,
X2,...X0.

Lines 400-459:

These lines are available for defining the object-
ive function, which must be assigned the variable
name F.

Lines 500-9000

These lines are available for defining the in-
equality constraint functions, which must be sub-
scripted variables named $G(i)$, $i=1,N2$.

Depending on the complexity of the problem, the user may
elect to use any BASIC programming structures in this sub-
program. As examples, FOR-NEXT loops, FUNCTIONS, SUB-
ROUTINES, and even other SUBPROGRAMS could be used in the
problem formulation. With about 7.3 Kbytes of memory avail-
able for constrained problem entry, and twice that for
unconstrained problems, there is space available for con-
siderable programming. Variable names used may be any except
those included in the PROB nomenclature, as defined in

48

Appendix B. If subscripted variables are to be used, a DIM (dimension) statement must be inserted near the beginning of the subprogram. When problem entry is complete, the subprogram is stored under the new file name.

Listings of the subprograms created for the test cases of Chapter IV are included at the end of Appendix B, under the names "BANANA" and "BEAM". Comparison of these listings to that of PROB, which is the unedited version, will help to illustrate the problem entry procedure for both constrained and unconstrained cases.

4. PROGRAM EXECUTION

MDOT is started either by execution of LOAD "Autost" and RUN commands, or by powering up the computer after inserting the tape cartridge, as explained in section III.B.4. After the "welcome" graphic, the user is queried as to the problem file name, progress display options and output format desired. Optimization then proceeds.

The user may choose to monitor the optimization closely, or perhaps not at all. For example, if a constrained problem is being run for the first time, the user may want to check for rapidly changing design variables so that the option of editing and restarting might be exercised. On the other hand, if a problem known to be well-behaved is being re-run with relatively minor changes, the user may elect to "ignore" MDOT until the solution is obtained.

Any of the values carried by the DEFALT subprogram, as defined in the listing in Appendix B, may be changed. They may be permanently modified by editing and re-storing DEFALT. Alternatively, a default value may be changed during program execution, whenever the edit option is invoked, by reassigning its value from the keyboard. If the program is restarted, though, DEFALT is recalled.

APPENDIX B
ANNOTATED PROGRAM LISTINGS

In an effort to conserve memory, MDOT was coded without
remark statements. The program listings are therefore pre-
sented in an annotated format to aid in following the logic
flowpaths. Nomenclature common to two or more modules is
defined first, then the additional nomenclature unique to
each module immediately precedes the applicable segment of
the code listing. Numbers in parentheses in the function
descriptions refer to line numbers in the associated program
segment. Parentheses following a variable name in the nomen-
clature lists indicate vectors, while parentheses enclosing
a comma indicate two-dimensional arrays.

COMMON NOMENCLATURE

A1()   Addresses in G() of violated and active constraints

A2()   Values of constraints identified in A1()

C1()   Vector of integer default values

C2()   Vector of non-integer default values

D()   Current objective function gradient

D1()   Previous iteration objective function gradient

F   Current value of objective function

F0   Initial value of objective function

51

F1	Previous iteration value of objective function
G()	Vector of current constraint values
G1(,)	Gradients of violated and active constraints
H(,)	Approximation to the inverse of the Hessian
L()	Vector of design variable lower bounds
N1	Number of design variables
N2	Number of inequality constraints
N3	Number of currently violated constraints
N4	Number of currently active constraints
P$	Problem subprogram file name character string
PÓ-P3	Polynomial approximation coefficients
Q1	Iteration counter
Q2	Function evaluation counter
Q3	Convergence counter
Q4	Convergence counter
R1	Golden section ratio
R2	Golden section ratio
S()	Search direction vector
X()	Current design
X0()	Initial design
X1()	Previous iteration design
X9()	Working vector of perturbed design variables

PROGRAM FLAGS

Unless otherwise noted, the flags listed are set to a value of 1 when the associated condition exists.

FLAG	CONDITION
1	Maximum number of iterations has been exceeded
2	Number of violated constraints greater than $N1+2$
3	Search vector components are all essentially zero
4	No appreciable move parameter can be found
5	Convergence to optimum has occurred
6	Variable metric algorithm is to be restarted
7-8	Unassigned
9	Subprogram is to be exited without execution
10	Termination has occurred
11	Progress option select: 1 = data display / 2 = graphic display

Module: Autost

Calls: Autost calls all other modules of MDOT into
             memory, then scratches those not needed for the
             type of problem (constrained or unconstrained)
             being run. For information transfer, Autost
             calls: PROB (edited), UCONT, CCONT, LOGO, and
             DEFALT.

Function: MDOT main calling program (10-370).
             Interactive problem initiation (1000-1200).
             Program loading (2000-2070).
             Initial design display generation (3000-3130).
             Design/default editing (4000-4260).

Nomenclature:

  A$ Interactive query response (string)

  S$&A$ Concatenated string to call appropriate optimizer

  D$ Display string ("FEASIBLE" or "INFEASIBLE")

  B$ Display string ("C1(" or "C2(")

  N0   Working variable for editing default values

  N   Interactive query response (numeric)

  K0   If = 0, indicates first problem of run

54

```
10 ! Autost....
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 DIM S$[4],A$[1],D$[10],B$[3]
50 SHORT X0(10),F0,G(50),L(10),U(10),N0
60 INTEGER N,I,K0
70 S$="CONT"
80 GOSUB 2000
90 CALL "LOGO" @ SCRATCHSUB "LOGO"
100 CALL "DEFALT"
110 Q1,Q2,Q3,Q4,N3,N4,K0=0
120 FOR I=1 TO 11
130 CFLAG I
140 NEXT I
150 WAIT 1000 @ GOSUB 1000
160 CLEAR @ CALL P$ ( 0 )
170 REDIM X0(N1)
180 IF N2=0 THEN 210
190 REDIM G(N2),L(N1),U(N1)
200 GOTO 220
210 MAT G=ZER(1)@ MAT L=G@ MAT U=L
220 CALL P$ ( 1,X0(),F0,G(),L(),U() )
230 GOSUB 3000
240 IF N2=0 THEN A$="U" ELSE A$="C"
250 CALL A$&S$ ( X0(),F0,G(),L(),U() )
260 CFLAG 10 @ CFLAG 9
270 CLEAR @ DISP @ DISP "Select option..." @ DISP @ DISP "
 1)  EDIT/RESTART"
280 DISP @ DISP "   2)  NEW PROBLEM" @ DISP @ DISP "   3)  E
XIT"
290 INPUT N
300 ON N GOTO 320,310,360
310 SCRATCHSUB P$ @ GOTO 100
320 GOSUB 4000
330 CALL P$ ( 1,X0(),F0,G(),L(),U() )
340 GOSUB 3000
350 GOTO 250
360 CLEAR
370 END
```

```
1000 IF K0>0 THEN 1070
1010 K0=1
1020 CLEAR @ DISP @ DISP "Select problem type for this    ru
n of MDOT:" @ DISP
1030 DISP "    1)  Constrained" @ DISP @ DISP "    2)  Uncon
strained" @ INPUT N
1040 ON N GOTO 1050,1060
1050 SCRATCHSUB "UCONT" @ SCRATCHSUB "VMSRCH" @ SCRATCHSUB "
NEWH" @ GOTO 1070
1060 SCRATCHSUB "CCONT" @ SCRATCHSUB "FDSRCH" @ SCRATCHSUB "
ACON" @ SCRATCHSUB "DIRECT"
1070 CLEAR @ DISP @ DISP "Have you created & stored your  pr
oblem?"
1080 DISP @ DISP "        (Y or N; ENTER)" @ INPUT A$
1090 IF A$="N" THEN 1120
1100 IF A$="Y" THEN 1160
1110 GOTO 1000
1120 CLEAR @ DISP @ DISP "        Please refer to the" @ DISP

1130 DISP "        MDOT USER MANUAL" @ DISP
1140 DISP "        for instructions..."
1150 GOTO 370
1160 DISP @ DISP "Enter problem subprogram       file name.
.." @ INPUT P$
1170 CLEAR @ DISP @ DISP "Select progress option..." @ DISP
@ DISP "    1)  GRAPHIC DISPLAY"
1180 DISP @ DISP "    2)  DATA DISPLAY" @ INPUT N@ CFLAG 11
1190 IF N=2 THEN SFLAG 11
1200 RETURN
2000 SFLAG 9 @ DISP "Loading MDOT..."
2010 CALL "DEFALT" @ CALL "CCONT" @ CALL "UCONT" @ CALL "ACO
N"
2020 DISP @ DISP "Still loading MDOT..."
2030 CALL "GRAD" @ CALL "DIRECT" @ CALL "FDSRCH" @ CALL "VMS
RCH"
2040 DISP @ DISP "Almost finished..."
2050 CALL "NEWH" @ CALL "CONV" @ CALL "PROG" @ CALL "TERM"
2060 CFLAG 9 @ CLEAR
2070 RETURN
```

```
3000 CLEAR @ DISP USING 3010 ; P$
3010 IMAGE 3/,13X,6A
3020 DISP USING 3030 ; N1,N2
3030 IMAGE 1/,4X,2D," Design Variables",/,4X,2D," Inequality
  Constraints"
3040 DISP USING 3050 ; F0
3050 IMAGE 2/,1X,"Initial Design:  F = ",K
3060 IF N2=0 THEN 3090
3070 IF AMAX(G)<C2(3) THEN D$="FEASIBLE" ELSE D$="INFEASIBLE
"
3080 DISP USING "11X,10A" ; D$
3090 ON KEY# 1,"EDIT" GOSUB 4000
3100 ON KEY# 4,"CONTINUE" GOTO 3130
3110 KEY LABEL
3120 GOTO 3120
3130 RETURN
4000 CLEAR @ DISP @ DISP "Select editing option..." @ DISP @
  DISP "    1)  DESIGN VARIABLES"
4010 DISP @ DISP "    2)  DEFAULT VALUES" @ INPUT N
4020 ON N GOTO 4030,4140
4030 CLEAR @ DISP "Enter the address in X of the   variable
to be changed" @ INPUT I
4040 DISP USING 4050 ; I,X0(I)
4050 IMAGE 2/,1X,"current value: ","X(",K,") = ",K
4060 DISP @ DISP "Enter the new value..." @ INPUT X0(I)
4070 CLEAR @ DISP @ DISP "Editing complete ?" @ DISP "
  (Y or N; ENTER)" @ INPUT A$
4080 IF A$="N" THEN 4000
4090 IF A$="Y" THEN 4110
4100 GOTO 4070
4110 Q1,Q2=0
4120 CALL P$ ( 2,X0(),F0,G(),L(),U() )
4130 GOTO 230
4140 CLEAR @ DISP @ DISP "Select default array..." @ DISP @
DISP "    1)  C1() integers"
4150 DISP @ DISP "    2)  C2() reals" @ INPUT N
4160 CLEAR @ DISP "Enter the address in C of the   variable
to be changed" @ INPUT I
4170 IF N=1 THEN N0=C1(I) ELSE N0=C2(I)
4180 IF N=1 THEN B$="C1(" ELSE B$="C2("
4190 DISP USING 4200 ; B$,I,N0
4200 IMAGE 2/,1X,"current value: ",3A,K,") = ",K
4210 DISP @ DISP "Enter the new value..." @ INPUT N0
4220 IF N=1 THEN C1(I)=N0 ELSE C2(I)=N0
4230 CLEAR @ DISP @ DISP "Editing complete ?" @ DISP "     (
Y or N; ENTER)" @ INPUT A$
4240 IF A$="N" THEN 4000
4250 IF A$="Y" THEN 4260
4260 RETURN
```

Module: LOGO

Called by: Autost

Function: Generate MDOT "welcome" graphic display on CRT.
           Scratched from memory upon execution.

```
1 SUB "LOGO"
10 PEN 1 @ GCLEAR
20 SCALE 0,10,0,10
30 PEN -1 @ GCLEAR 7
40 PEN 1 @ GCLEAR 3
50 CSIZE 30 @ LORG 5 @ PEN -1
60 MOVE 5,5 @ LABEL "m"
70 CSIZE 6 @ MOVE 5,6.5 @ LABEL "o"
80 WAIT 1000 @ PEN 1 @ GCLEAR
90 CSIZE 12 @ LORG 2
100 MOVE 1,8 @ LABEL "m"
110 MOVE 1,6 @ LABEL "d"
120 MOVE 1,4 @ LABEL "o"
130 MOVE 1,2 @ LABEL "t"
140 CSIZE 3
150 MOVE 2,8 @ LABEL "icrocomputer-based"
160 MOVE 2,6 @ LABEL "esign"
170 MOVE 2,4 @ LABEL "ptimization"
180 MOVE 2,2 @ LABEL "ool"
190 SUBEND
```

Module: DEFALT

Called by: Autost

Function: Set MDOT working parameters to their default val-
           ues. DEFALT is called each time a new problem is
           executed.

Nomenclature:

  C1(1)   Maximum number of iterations
  C1(2)   Consecutive iterations for convergence criteria
  C1(3)   Variable metric method select: 0 = DFP, 1 = BFGS
  C1(4)-C2(10) Unassigned
  C2(1)   Finite difference perturbation factor
  C2(2)   Minimum absolute finite difference step
  C2(3)   Violated constraint criterion (tolerance)
  C2(4)   Active constraint criterion (thickness)
  C2(5)   Push-off factor multiplier (theta zero)
  C2(6)   Maximum value of push-off factor
  C2(7)   Factor used in DIRECT when infeasible
  C2(8)   Factor used in step length estimate based on F
  C2(9)   Factor used in step length estimate based on X
  C2(10)   F convergence criterion (relative)
  C2(11)   F convergence criterion (absolute)
  C2(12)   Defined zero
  C2(13)   Epsilon, used to prevent division by zero
  C2(14)-C2(20) Unassigned

```
10 SUB "DEFALT"
20 OPTION BASE 1
30 COM F$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 MAT C1=ZER(10)
42 MAT C2=ZER(20)
44 C1(1)=20 ! max # iterations
46 C1(2)=2 ! consec. conv.
48 C1(3)=1 ! w in H update
50 C2(1)=.01 ! fin. diff. mult.
52 C2(2)=.001 ! min. fin. step
54 C2(3)=.004 ! const. viol.
56 C2(4)=-.1 ! active const.
58 C2(5)=1 ! push-off mult.
60 C2(6)=50 ! max. push-off
62 C2(7)=100000 ! Phi (DIRECT)
64 C2(8)=.1 ! obj. mult.
66 C2(9)=.1 ! des. var. mult.
68 C2(10)=.001 ! min rel F
70 C2(11)=.001 ! min abs F
72 C2(12)=.001 ! zero
74 C2(13)=.0001 ! epsilon
100 SUBEND
```

Module: PROB

Called by: User

Function: Provide the user with a skeleton subprogram into
           which the optimization problem is entered by
           editing.

Module: PROB (edited)

Called by: Autost, GRAD, FDSRCH, VMSRCH

Function: Input the number of design variables and the
           number of inequality constraints (90-100).
           Input the intial design and side constraints
           (130-210).
           Evaluate the objective function and inequality
           constraints (230-9000).

Nomenclature:

  K1   Flag to indicate first call or subsequent call
  L$ String used in lower bound input
  U$ String used in upper bound input
  X1-X0   Design variable names used in problem input

62

```
10 SUB "PROB" (K1,X(),F,G(),L(),U())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
35 SHORT X1,X2,X3,X4,X5,X6,X7,X8,X9,X0
40 DIM L$[6],U$[6]
50 IF K1>1 THEN 220
90 READ N1,N2
100 DATA
105 IF K1=0 THEN SUBEXIT
130 FOR I=1 TO N1
140 READ X(I)
142 IF N2=0 THEN 170
144 READ L$,U$
150 IF L$="N" THEN L(I)=-1.E99 ELSE L(I)=VAL(L$)
160 IF U$="N" THEN U(I)=1.E99 ELSE U(I)=VAL(U$)
170 NEXT I
201 DATA
202 DATA
203 DATA
204 DATA
205 DATA
206 DATA
207 DATA
208 DATA
209 DATA
210 DATA
220 GOSUB 9010
230 ! User-defined expressions
399 ! Objective function
499 Q2=Q2+1
500 ! CONSTRAINTS
9000 SUBEND
9010 X1=X(1) @ IF N1=1 THEN RETURN
9020 X2=X(2) @ IF N1=2 THEN RETURN
9030 X3=X(3) @ IF N1=3 THEN RETURN
9040 X4=X(4) @ IF N1=4 THEN RETURN
9050 X5=X(5) @ IF N1=5 THEN RETURN
9060 X6=X(6) @ IF N1=6 THEN RETURN
9070 X7=X(7) @ IF N1=7 THEN RETURN
9080 X8=X(8) @ IF N1=8 THEN RETURN
9090 X9=X(9) @ IF N1=9 THEN RETURN
9100 X0=X(10) @ RETURN
```

Module: CCONT

Called by: Autost

Calls: TERM, PROG, ACON, GRAD, DIRECT, FDSRCH, CONV

Function: Control constrained optimization by the method of
           feasible directions (Fig. 2).

```
10 SUB "CCONT" (X0(),F0,G(),L(),U())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT X1(10),X(10),D(10),G1(12,10),A2(12),S(10)
50 SHORT F,B0,A0,F1
60 INTEGER A1(12)
70 IF FLAG(9) THEN SUBEXIT
80 REDIM X1(N1),D(N1),G1(N1+2,N1),S(N1)
90 A0,F1,B0=0 @ F=F0
100 MAT S=ZER(N1)@ MAT X=X0
110 MAT A1=ZER(N1+2)@ MAT A2=A1
120 Q1=Q1+1
130 IF Q1<C1(1) THEN 160
140 SFLAG 1 @ CALL "TERM" ( X(),F,G() ) @ SUBEXIT
150 SUBEXIT
160 CALL "PROG" ( X0(),X(),F/F0,F ) @ IF FLAG(9) THEN SUBEXI
T
170 CALL "ACON" ( G(),A1(),A2() ) @ IF FLAG(9) THEN SUBEXIT
180 IF FLAG(2) THEN CALL "TERM" ( X(),F,G() )
190 IF FLAG(10) THEN SUBEXIT
200 IF N3+N4#0 THEN REDIM G1(N3+N4,N1)
210 CALL "GRAD" ( X(),F,D(),G(),A1(),A2(),G1(,) ) @ IF FLAG(
9) THEN SUBEXIT
220 IF N3+N4#0 THEN 240
230 MAT S=-D@ GOTO 360
240 CALL "DIRECT" ( D(),G1(,),A1(),A2(),S(),B0 ) @ IF FLAG(9
) THEN SUBEXIT
250 CFLAG 3
260 IF MAXAB(S)>C2(12) AND ABS(B0)>C2(12) THEN 360
270 SFLAG 3
280 IF N3#0 THEN CALL "TERM" ( X(),F,G() )
290 IF FLAG(10) THEN SUBEXIT
300 IF MAXAB(A2)<=C2(12) THEN CALL "TERM" ( X(),F,G() )
310 IF FLAG(10) THEN SUBEXIT
320 IF C2(4)>=-C2(12) THEN CALL "TERM" ( X(),F,G() )
330 IF FLAG(10) OR FLAG(9) THEN SUBEXIT
340 C2(4)=C2(4)/3
350 GOTO 170
360 MAT S=(1/MAXAB(S))*S@ CFLAG 4
370 MAT X1=X@ F1=F
380 CALL "FDSRCH" ( X(),L(),U(),F,D(),G(),A1(),A2(),G1(,),S(
),A0 )
390 IF FLAG(9) THEN SUBEXIT
400 IF FLAG(4) THEN 280
410 CFLAG 5
420 MAT X=(1)*X1+(A0)*S
430 CALL P$ ( 2,X(),F,G() )
440 CALL "CONV" ( X(),X1(),F,F1 ) @ IF FLAG(9) THEN SUBEXIT
450 IF FLAG(5) THEN CALL "TERM" ( X(),F,G() )
460 IF FLAG(10) OR FLAG(9) THEN SUBEXIT ELSE GOTO 120
470 SUBEND
```

Module: UCONT

Called by: Autost

Calls: GRAD, PROG, TERM, VMSRCH, CONV, NEWH

Function: Control unconstrained optimization by the
           variable metric method (Fig. 1).

Nomenclature:

K   Indicates an iteration in which no move parameter
           to improve the design was found
A   One-dimensional search move parameter

```
10 SUB "UCONT" (X0(),F0,G(),L(),U())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT X(10),X1(10),D(10),D1(10),S(10),H(10,10)
50 SHORT F,F1,A
60 INTEGER K
70 IF FLAG(9) THEN SUBEXIT
80 REDIM X(N1),X1(N1),D(N1),D1(N1)
90 MAT X=X0@ F=F0 @ K=0
100 CALL "GRAD" ( X(),F,D(),G() )
110 MAT H=IDN(N1,N1)
120 Q1=Q1+1
130 CALL "PROG" ( X0(),X(),F/F0,F ) @ IF FLAG(9) THEN SUBEXI
T
140 IF Q1<C1(1) THEN 170
150 SFLAG 1 @ CALL "TERM" ( X(),F,G() )
160 IF FLAG(10) OR FLAG(9) THEN SUBEXIT
170 MAT S=H*D
180 MAT S=-S
190 MAT S=(1/MAXAB(S))*S
200 MAT X1=X@ MAT D1=D@ F1=F
210 CALL "VMSRCH" ( X(),F,D(),S(),A ) @ IF FLAG(9) THEN SUBE
XIT
220 IF A>C2(12) THEN 270
230 K=K+1
240 IF K<2 THEN 110
250 SFLAG 4 @ CALL "TERM" ( X(),F,G() )
260 IF FLAG(10) OR FLAG(9) THEN SUBEXIT
270 MAT X=(1)*X+(A)*S
280 CALL "CONV" ( X(),X1(),F,F1 ) @ IF FLAG(9) THEN SUBEXIT
290 IF FLAG(5) THEN CALL "TERM" ( X(),F,G() )
300 IF FLAG(10) OR FLAG(9) THEN SUBEXIT
310 IF FLAG(6) THEN 110
320 CALL "GRAD" ( X(),F,D(),G() ) @ IF FLAG(9) THEN SUBEXIT
330 CALL "NEWH" ( X(),X1(),D(),D1(),H(,) ) @ IF FLAG(9) THEN
 SUBEXIT
340 GOTO 120
350 SUBEND
```

67

Module: ACON

Called by: CCONT

Function: Determine the number of currently violated (N3),
           and active (N4), inequality constraints.
           Construct the A1() vector of the addresses in G()
           of the active/violated constraint set.
           Construct the A2() vector of the current values
           of this set.
           Violated constraint information is stored in the
           first N3 rows of A1() and A2(), active constraint
           information in the last N4 rows.

```
10 SUB "ACON" (G(),A1(),A2())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 ON KEY# 1 GOTO 300
50 IF FLAG(9) THEN SUBEXIT
60 CFLAG 2 @ N3,N4=0
70 FOR I=1 TO N2
80 IF G(I)>=C2(3) THEN 110
90 IF G(I)>=C2(4) THEN N4=N4+1
100 GOTO 120
110 N3=N3+1
120 NEXT I
130 IF N3<=N1+2 THEN 150
140 SFLAG 2 @ SUBEXIT
150 IF N3+N4=0 THEN SUBEXIT
160 MAT A1=ZER(N3+N4)@ MAT A2=A1
170 J3,J4=1
180 FOR I=1 TO N2
190 IF G(I)>=C2(3) THEN 250
200 IF G(I)<C2(4) THEN 280
210 A1(N3+J4)=I
220 A2(N3+J4)=G(I)
230 J4=J4+1
240 GOTO 280
250 A1(J3)=I
260 A2(J3)=G(I)
270 J3=J3+1
280 NEXT I
290 GOTO 310
300 SFLAG 9 @ SUBEXIT
310 SUBEND
```

Module: GRAD

Called by: CCONT, UCONT

Calls: PROB (edited)

Function: Calculate the gradient of the objective function
           by first forward finite difference approximation.
           In constrained optimization, calculate gradients
           of the active/violated inequality constraint set.

Nomenclature:
  F2   Intermediate function evaluation
  N   Design variable perturbation

```
10 SUB "GRAD" (X(),F,D(),G(),A1(),A2(),G1(,))
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT F2,X9(10),N
50 ON KEY# 1 GOTO 210
60 IF FLAG(9) THEN SUBEXIT
70 FOR I=1 TO N1
80 MAT X9=X
90 N=C2(1)*ABS(X9(I))
100 IF N<C2(2) THEN N=C2(2)
110 X9(I)=X9(I)+N
120 CALL P$ ( 2,X9(),F2,G() )
130 IF N2=0 THEN 180
140 IF N3+N4=0 THEN 180
150 FOR J=1 TO N3+N4
160 G1(J,I)=(G(A1(J))-A2(J))/N
170 NEXT J
180 D(I)=(F2-F)/N
190 NEXT I
200 GOTO 220
210 SFLAG 9 @ SUBEXIT
220 SUBEND
```

71

Module: DIRECT

Called by: CCONT

Function: Solve the direction-finding subproblem in the
           method of feasible directions.
           Calculate constraint push-off factors (170-200).
           Initialize working arrays for currently feasible
           (2000-2080), or infeasible (1000-1090) designs.
           Determine the direction vector S(), and the para-
           meter B0 (300-700).

Nomenclature:

  A(,)   Working array constructed from G1(,) and T()

  B(,)   Working array initialized as $-A^T A$

  B0   Kuhn-Tucker parameter

  B2   Intermediate element value used in pivoting

  B3   Intermediate element value used in pivoting

  B9   Intermediate variable used in pivoting

  C()   Working vector

  D0()   Working vector initialized as D()

  G0(,)   Working array initialized as G1(,)

  G9()   Working vector used in constructing G0(,)

  I9()   Working vector of element indices

  J9   Working integer scaler

  K9   Working integer scaler

  N9   Dimension of active/violated constraint set

P()   Working vector

T()   Vector of constraint push-off factors

U()   Working vector

Y()   Solution vector, contains S() and B0

```
10 SUB "DIRECT" (D(),G1(,),A1(),A2(),S(),B0)
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT B2,B3,B9,A(13,11),T(12),B(13,13),P(11),U(13),C(13),
Y(11),D0(10),G0(12,10),G9(10)
50 INTEGER J9,K9,I9(13),N9
60 ON KEY# 1 GOTO 650
70 IF FLAG(9) THEN SUBEXIT
80 N9=N3+N4
90 REDIM T(N9),P(N1+1),Y(N1+1)
100 FOR I=1 TO N9
110 T(I)=C2(5)*(1-A2(I)/C2(4))^2
120 IF T(I)>C2(6) THEN T(I)=C2(6)
130 NEXT I
140 MAT P=ZER(N1+1)@ MAT D0=D
150 MAT D0=(1/MAXAB(D0))*D0
160 MAT G0=G1
170 FOR I=1 TO N9
180 MAT G9=G0(I,)
190 MAT G9=(1/MAXAB(G9))*G9
200 MAT G0(I,)=G9
210 NEXT I
220 IF N3>0 THEN GOSUB 1000 ELSE GOSUB 2000
230 MAT B=A*TRN(A)
240 MAT B=-B
250 N=UBND(B,1)
260 MAT I9=ZER(N)
270 A9,K9=0
280 FOR I=1 TO N
290 IF C(I)>=0 THEN 330
300 B9=C(I)/B(I,I)
310 IF B9<=A9 THEN 330
320 K9=I @ A9=B9
330 NEXT I
340 IF K9=0 THEN 540
350 J9=I9(K9)
360 I9(K9)=0
370 IF J9=0 THEN I9(K9)=K9
380 B2=B(K9,K9)
390 FOR I=1 TO N
400 B(K9,I)=B(K9,I)/B2
410 NEXT I
```

74

```
420 C(K9)=A9
430 B(K9,K9)=1/B2
440 FOR I=1 TO N
450 IF I=K9 THEN 520
460 B3=B(I,K9)
470 B(I,K9)=0
480 FOR J=1 TO N
490 B(I,J)=B(I,J)-B3*B(K9,J)
500 NEXT J
510 C(I)=C(I)-B3*A9
520 NEXT I
530 GOTO 270
540 FOR I=1 TO N
550 U(I)=0
560 J9=I9(I)
570 IF J9<=0 THEN 590
580 U(I)=C(J9)
590 NEXT I
600 MAT Y=TRN(A)*U
610 MAT Y=P-Y
620 MAT S=Y(1:N1)
630 B0=Y(N1+1)
640 GOTO 660
650 SFLAG 9 @ SUBEXIT
660 SUBEND
1000 MAT C=ZER(N9)@ MAT U=ZER(N9)
1010 MAT A=CON(N9,N1+1)
1020 MAT A(,1:N1)=G0
1030 MAT A(,N1+1)=T
1040 MAT P(1:N1)=D0
1050 MAT P=-P
1060 P(N1+1)=C2(7)
1070 MAT C=A*P
1080 MAT C=-C
1090 RETURN
2000 MAT C=ZER(N9+1)@ MAT U=ZER(N9+1)
2010 MAT A=CON(N9+1,N1+1)
2020 MAT A(1:N9,1:N1)=G0
2030 MAT A(1:N9,N1+1)=T
2040 MAT A(N9+1,1:N1)=D0
2050 P(N1+1)=1
2060 MAT C=A(,N1+1)
2070 MAT C=-C
2080 RETURN
```

Module: FDSRCH

Called by: CCONT

Calls: PROB (edited)

Function: Perform one-dimensional search for constrained optimization in the method of feasible directions.

Estimate initial search move parameter (1000-1180).

Check for side constraint violations (2000-2050).

Establish bounds on solution, feasible (3000-3340) or infeasible (4000-4360).

- Refine solution by polynomial approximation, feasible (5000-5240) or infeasible (6000-6580).

Nomenclature:

A	Move parameter
A0()	Working vector of move parameters
A()	Working vector of move parameters
A1	Initial A based on change in objective function
A2	Initial A based on attaining feasibility
B(,)	Array of constraint values during search
D0	Dot product of D() and S() or G1(i,) and S()
G0()	Working vector of constraint gradients
M()	Working vector of maximum constraint values
Y()	Working vector of objective function values

```
10 SUB "FDSRCH" (X(),L(),U(),F,D(),G(),A1(),A2(),G1(,),S(),A
)
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT X9(10),R1,R2,A1,A2,A(4),Y(4),M(4),B(4,50),D0,G0(10)
,P0,P1,P2,B0,A0(51)
50 INTEGER I,K
60 ON KEY# 1 GOTO 210
70 IF FLAG(9) THEN SUBEXIT
80 REDIM X9(N1),B(4,N2),G0(N1),A0(N2+1)
90 R1=(3-SQR(5))/2 @ R2=2-R1
100 A,A1,A2,A3,A(1)=0 @ CFLAG 4
110 MAT M=ZER(4)@ MAT X9=X@ Y(1)=F @ MAT B(1,)=G@ M(1)=AMAX(
G)
120 GOSUB 1000
130 A(2)=A @ GOSUB 2000
140 IF N3=0 THEN GOSUB 3000 ELSE GOSUB 4000
150 IF N3=0 THEN GOSUB 5000 ELSE GOSUB 6000
160 A=A(K)
170 IF A>C2(12) THEN 190
180 SFLAG 4 @ SUBEXIT
190 F=Y(K) @ MAT G=B(K,)
200 GOTO 220
210 SFLAG 9 @ SUBEXIT
220 SUBEND
1000 GOSUB 1050
1010 IF N3#0 THEN 1030
1020 A=A1 @ RETURN
1030 IF A2>2*A1 THEN A=2*A1 ELSE A=MAX(A1,A2)
1040 RETURN
1050 D0=DOT(D,S)
1060 A1=C2(8)*ABS(F)/ABS(D0)
1070 FOR I=1 TO N1
1080 A=C2(9)*ABS(X(I))/ABS(S(I))
1090 IF A<A1 THEN A1=A
1100 NEXT I
1110 IF N3=0 THEN RETURN
1120 FOR I=1 TO N3
1130 MAT G0=G1(I,)
1140 D0=DOT(G0,S)
1150 A=-(A2(I)/D0)
1160 IF A>A2 THEN A2=A
1170 NEXT I
1180 RETURN
2000 MAT X=(1)*X9+(A)*S
2010 FOR I=1 TO N1
2020 IF X(I)<L(I) THEN X(I)=L(I)
2030 IF X(I)>U(I) THEN X(I)=U(I)
2040 NEXT I
2050 RETURN
```

```
3000 M(1)=AMAX(G)
3010 CALL P$ ( 2,X(),F,G() )
3020 M(2)=AMAX(G)
3030 Y(2)=F
3040 MAT B(2,)=G
3050 IF Y(2)>Y(1) THEN 3070
3060 IF M(2)<=C2(12) THEN 3150
3070 A(3)=A(1)+R1*(A(2)-A(1))
3080 A=A(3)
3090 GOSUB 2000
3100 CALL P$ ( 2,X(),F,G() )
3110 M(3)=AMAX(G)
3120 Y(3)=F
3130 MAT B(3,)=G
3140 RETURN
3150 A(3)=A(2)
3160 Y(3)=Y(2)
3170 M(3)=M(2)
3180 MAT G=B(2,)
3190 MAT B(3,)=G
3200 A(2)=(1+R2)*A(3)-R2*A(1)
3210 A=A(2)
3220 GOSUB 2000
3230 CALL P$ ( 2,X(),F,G() )
3240 Y(2)=F
3250 M(2)=AMAX(G)
3260 MAT B(2,)=G
3270 IF Y(2)>Y(3) THEN RETURN
3280 IF M(2)>C2(12) THEN RETURN
3290 A(1)=A(3)
3300 Y(1)=Y(3)
3310 M(1)=M(3)
3320 MAT G=B(3,)
3330 MAT B(1,)=G
3340 GOTO 3150
```

```
4000 M(1)=AMAX(G)
4010 CALL P$ ( 2,X(),F,G() )
4020 Y(2)=F
4030 M(2)=AMAX(G)
4040 MAT B(2,)=G
4050 IF M(2)>M(1) THEN 4080
4060 IF M(2)>0 THEN 4160
4070 IF Y(2)<Y(1) THEN 4160
4080 A(3)=A(1)+R1*(A(2)-A(1))
4090 A=A(3)
4100 GOSUB 2000
4110 CALL P$ ( 2,X(),F,G() )
4120 Y(3)=F
4130 M(3)=AMAX(G)
4140 MAT B(3,)=G
4150 RETURN
4160 A(3)=A(2)
4170 Y(3)=Y(2)
4180 MAT G=B(2,)
4190 MAT B(3,)=G
4200 M(3)=M(2)
4210 A(2)=(1+R2)*A(3)-R2*A(1)
4220 A=A(2)
4230 GOSUB 2000
4240 CALL P$ ( 2,X(),F,G() )
4250 Y(2)=F
4260 M(2)=AMAX(G)
4270 MAT B(2,)=G
4280 IF M(2)>M(3) THEN RETURN
4290 IF M(2)>0 THEN 4310
4300 IF Y(2)>Y(3) THEN RETURN
4310 A(1)=A(3)
4320 Y(1)=Y(3)
4330 M(1)=M(3)
4340 MAT G=B(3,)
4350 MAT B(1,)=G
4360 GOTO 4160
```

```
5000 P2=((Y(3)-Y(1))/(A(3)-A(1))-(Y(2)-Y(1))/(A(2)-A(1)))/(A
(3)-A(2))
5010 P1=(Y(2)-Y(1))/(A(2)-A(1))-P2*(A(1)+A(2))
5020 A0(1)=-(P1/(2*P2))
5030 FOR I=1 TO N2
5040 P2=((B(3,I)-B(1,I))/(A(3)-A(1))-(B(2,I)-B(1,I))/(A(2)-A
(1)))/(A(3)-A(2))
5050 P1=(B(2,I)-B(1,I))/(A(2)-A(1))-P2*(A(1)+A(2))
5060 P0=B(1,I)-P1*A(1)-P2*A(1)^2
5070 B0=P1^2-4*P0*P2
5080 IF B0>0 THEN 5110
5090 A0(I+1)=-(P0/P1)
5100 GOTO 5130
5110 A0(I+1)=MAX(-P1+SQR(B0),-P1-SQR(B0))
5120 A0(I+1)=A0(I+1)/(2*P2)
5130 NEXT I
5140 A(4)=AMIN(A0)
5150 A=A(4)
5160 GOSUB 2000
5170 CALL P$ ( 2,X(),F,G() )
5180 Y(4)=F @ MAT B(4,)=G
5190 M(4)=AMAX(G)
5200 K=1
5210 FOR I=2 TO 4
5220 IF Y(I)<Y(K) AND M(I)<=C2(12) THEN K=I
5230 NEXT I
5240 RETURN
```

```
6000 MAT A0=ZER(N3+1)
6005 IF AMIN(M)>0 THEN 6360
6010 P2=((Y(3)-Y(1))/(A(3)-A(1))-(Y(2)-Y(1))/(A(2)-A(1)))/(A
(3)-A(2))
6020 P1=(Y(2)-Y(1))/(A(2)-A(1))-P2*(A(1)+A(2))
6030 A0(1)=-(P1/(2*P2))
6040 FOR I=1 TO N3
6050 GOSUB 6490
6060 IF B0>0 THEN 6090
6070 A0(I+1)=-(P0/P1)
6080 GOTO 6110
6090 A0(I+1)=MIN(-P1+SQR(B0),-P1-SQR(B0))
6100 A0(I+1)=A0(I+1)/(2*P2)
6110 NEXT I
6120 A(4)=AMAX(A0)
6130 MAT A0=ZER(N2-N3)@ J1=1
6140 FOR I=1 TO N2
6150 FOR J=1 TO N3
6160 IF I=A1(J) THEN 6240
6170 NEXT J
6180 GOSUB 6540
6190 IF B0>0 THEN 6220
6200 A0(J1)=-(P0/P1)
6210 J1=J1+1 @ GOTO 6240
6220 A0(J1)=MIN(-P1+SQR(B0),-P1-SQR(B0))
6230 A0(J1)=A0(J1)/(2*P2)
6235 J1=J1+1
6240 NEXT I
```

```
6250 A(4)=MIN(A(4),AMIN(A0))
6260 A=A(4)
6270 GOSUB 2000
6280 CALL F$ ( 2,X(),F,G() )
6290 Y(4)=F @ MAT B(4,)=G
6300 M(4)=AMAX(G)
6310 K=1
6320 FOR I=2 TO 4
6330 IF Y(I)<Y(K) AND M(I)<=C2(4) THEN K=I
6340 NEXT I
6350 RETURN
6360 P2=((M(3)-M(1))/(A(3)-A(1))-(M(2)-M(1))/(A(2)-A(1)))/(A
(3)-A(2))
6370 P1=(M(2)-M(1))/(A(2)-A(1))-P2*(A(1)+A(2))
6380 A(4)=-(P1/(2*P2))
6390 IF A(3)<A(4) AND A(4)<A(2) THEN 6420
6400 M(4)=2*M(2)
6410 GOTO 6460
6420 A=A(4)
6430 GOSUB 2000
6440 CALL F$ ( 2,X(),F,G() )
6450 M(4)=AMAX(G) @ Y(4)=F @ MAT B(4,)=G
6460 M=AMIN(M)
6470 K=AMINROW
6480 RETURN
6490 P2=((B(3,A1(I))-B(1,A1(I)))/(A(3)-A(1))-(B(2,A1(I))-B(1
,A1(I)))/(A(2)-A(1)))/(A(3)-A(2))
6500 P1=(B(2,A1(I))-B(1,A1(I)))/(A(2)-A(1))-P2*(A(1)+A(2))
6510 P0=B(1,A1(I))-P1*A(1)-P2*A(1)^2
6520 B0=P1^2-4*P0*P2
6530 RETURN
6540 P2=((B(3,I)-B(1,I))/(A(3)-A(1))-(B(2,I)-B(1,I))/(A(2)-A
(1)))/(A(3)-A(2))
6550 P1=(B(2,I)-B(1,I))/(A(2)-A(1))-P2*(A(1)+A(2))
6560 P0=B(1,I)-P1*A(1)-P2*A(1)^2
6570 B0=P1^2-4*P0*P2
6580 RETURN
```

Module: VMSRCH

Called by: UCONT

Calls: PROB (edited)

Function: Perform unconstrained one-dimensional search in
            variable metric method.
            Establish bounds on the solution (100-350).
            Refine the solution by polynomial approximation
            (360-520).

Nomenclature:

   A0   Move parameter
   A()   Working vector of move parameters
   A1()   Working vector of move parameters
   D0   Dot product of D() and S()
   F9   Working value of objective function
   Y()   Working vector of objective function values
   Y1()   Working vector of objective function values

```
10  SUB "VMSRCH" (X(),F,D(),S(),A0)
20  OPTION BASE 1
30  COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40  SHORT D0,A1(11),X9(10),Y(4),A(4),R1,R2,P1,P2,P3,Y1(4),F9,
B
50  ON KEY# 1 GOTO 540
60  IF FLAG(9) THEN SUBEXIT
70  R1=(3-SQR(5))/2 @ R2=(1+SQR(5))/2
80  MAT A=ZER(4)@ MAT Y=A
90  Y(1)=F @ F9=0
100 D0=DOT(D,S)
110 MAT A1=ZER(N1+1)
120 MAT X9=X
130 A1(1)=C2(8)*ABS(F)/ABS(D0)
140 FOR I=1 TO N1
150 A1(I+1)=C2(9)*ABS(X9(I))/ABS(S(I))
160 NEXT I
170 A(2)=AMIN(A1)
180 MAT X9=(1)*X9+(A(2))*S
190 CALL P$ ( 2,X9(),F9 )
200 Y(2)=F9
210 IF Y(2)>Y(1) THEN 310
220 A(3)=A(2) @ Y(3)=Y(2)
230 A(2)=(1+R2)*A(3)-R2*A(1)
240 MAT X9=X
250 MAT X9=(1)*X9+(A(2))*S
260 CALL P$ ( 2,X9(),F9 )
270 Y(2)=F9
280 IF Y(2)>Y(3) THEN 360
290 A(1)=A(3) @ Y(1)=Y(3)
300 GOTO 220
310 A(3)=R1*A(2)
320 MAT X9=X
330 MAT X9=(1)*X9+(A(3))*S
```

```
340 CALL P$ ( 2,X9(),F9 )
350 Y(3)=F9
360 P3=(Y(3)-Y(1))*(A(2)-A(1))/(A(3)-A(1)-(Y(2)-Y(1))*(A(3)-
A(1)))
370 P3=P3/(A(2)-A(1)+(A(3)-A(2))*D0)/((A(2)-A(1))*(A(3)-A(1)
)*(A(3)-A(2)))
380 P2=((Y(2)-Y(1))/(A(2)-A(1))-D0)/(A(2)-A(1))-P3*(2*A(1)+A
(2))
390 P1=D0-2*P2*A(1)-3*P3*A(1)^2
400 B=P2^2-3*P1*P3
410 IF B>=0 THEN 430
420 J=3 @ GOTO 490
430 A(4)=(-P2+SQR(B))/(3*P3)
440 MAT X9=X
450 MAT X9=(1)*X9+(A(4))*S
460 CALL P$ ( 2,X9(),F9 )
470 Y(4)=F9
480 J=4
490 MAT Y1=Y(1:J)
500 F=AMIN(Y1)
510 K=AMINROW
520 A0=A(K)
530 GOTO 550
540 SFLAG 7 @ SUBEXIT
550 SUBEND
```

```
Module:    NEWH

Called by: UCONT

Function:  Update the approximation to the inverse of the
           Hessian matrix used in determining the search
           direction in the variable metric method.

Nomenclature:
   D0(,)     Update matrix
   P()       Working vector initialized as X()-X1()
   S         Dot product of P() and Y()
   T()       Working vector
   T1()      Working vector
   T2(,)     Working array
   T3(,)     Working array
   Y()       Working vector initialized as D()-D1()
```

```
10 SUB "NEWH" (X(),X1(),D(),D1(),H(,))
20 OPTION BASE 1
30 COM P$[4] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT D0(10,10),Y(10),P(10),T1(10),T2(10,10),T(1),S,T3(10
,10)
50 ON KEY# 1 GOTO 230
60 IF FLAG(9) THEN SUBEXIT
70 REDIM D0(N1,N1),Y(N1),P(N1),T1(N1),T2(N1,N1),T3(N1,N1)
80 MAT P=X-X1
90 MAT Y=D-D1
100 S=DOT(P,Y)
110 MAT T1=H*Y
120 MAT T2=T1*TRN(T1)
130 MAT T=TRN(Y)*T1
140 MAT D0=P*TRN(P)
150 MAT D0=((S+T(1)*C1(3))/S^2)*D0
160 MAT D0=(1)*D0+((C1(3)-1)/T(1))*T2
170 MAT T2=T1*TRN(P)
180 MAT T3=P*TRN(T1)
190 MAT T2=T2+T3
200 MAT D0=(1)*D0+(-(C1(3)/S))*T2
210 MAT H=H+D0
220 GOTO 240
230 SFLAG 9 @ SUBEXIT
240 SUBEND
```

Module: CONV

Called by: CCONT, UCONT

Function: Determine whether the design has converged to the
            optimum in the last iteration.
            Update convergence criteria based on iteration
            history.

```
10 SUB "CONV" (X(),X1(),F,F1)
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 SHORT X9(10)
50 IF FLAG(9) THEN SUBEXIT
60 CFLAG 6
70 C2(8)=(C2(8)+ABS((F1-F)/F1))/2
80 MAT X9=X1-X@ MAT X9=X9/X1
90 C2(9)=(C2(9)+MAXAB(X9))/2
100 IF ABS(F1-F)<MIN(C2(13),C2(11)*ABS(F1)) THEN Q3=Q3+1 ELS
E Q3=0
110 IF ABS(F1-F)/MAX(ABS(F1),.00001)<C2(10) THEN Q4=Q4+1 ELS
E Q4=0
120 IF MAX(Q3,Q4)>=C1(2) THEN SFLAG 5
130 IF MAX(Q3,Q4)>0 THEN SFLAG 6
140 SUBEND
```

Module: PROG

Called by: CCONT, UCONT

Function: Generate optimization progress information
           output.

```
10 SUB "PROG" (X0(),X(),Y,F)
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 ON KEY# 1,"INTRPT" GOTO 300
50 IF FLAG(9) THEN SUBEXIT
60 IF FLAG(11) THEN 230
70 IF Q1>1 THEN 180
80 GCLEAR @ PEN 1 @ LORG 5
90 SCALE -2,21,-3.2,3.2
100 XAXIS 0,1,0,20
110 YAXIS 0,1,-3,3
120 FOR I=-3 TO 3
130 MOVE -.5,I @ LABEL I
140 NEXT I
150 MOVE 10,3 @ LABEL "Iteration History"
160 MOVE 10,2.7 @ LABEL P$
170 MOVE -3.1,5 @ LABEL "K1 to interrupt"
180 FOR I=1 TO N1
190 MOVE Q1,X(I)/X0(I) @ LABEL I
200 NEXT I
210 MOVE Q1,Y @ LABEL "f"
220 SUBEXIT
230 IF Q1>1 THEN 270
240 CLEAR @ DISP USING "13X,6A" ; P$
250 DISP "Iteration     Objective Function"
260 KEY LABEL
270 DISP USING 280 ; Q1,F
280 IMAGE 3X,2D,13X,K
290 GOTO 310
300 SFLAG 9 @ SUBEXIT
310 SUBEND
```

Module: TERM

Called by: CCONT, UCONT

Function: Generate output of optimization results.

.

```
10 SUB "TERM" (X(),F,G())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
40 IF FLAG(9) THEN SUBEXIT
50 GOSUB 1000
60 IF FLAG(1) THEN GOSUB 2000
70 IF FLAG(2) THEN GOSUB 3000
80 IF FLAG(3) THEN GOSUB 4000
90 IF FLAG(4) THEN GOSUB 5000
100 IF FLAG(5) THEN GOSUB 6000
110 IF FLAG(11) THEN 150
120 PRINT @ PRINT @ PRINT
130 GRAPH @ COPY @ SUBEXIT
140 PRINT @ PRINT @ COPY
150 SUBEND
1000 PRINT @ PRINT
1010 PRINT USING "13X,6A" ; P$
1020 PRINT USING 1030 ; F
1030 IMAGE 1/,8X,"OPTIMUM = ",K
1040 FOR I=1 TO N1
1050 PRINT USING 1060 ; I,X(I)
1060 IMAGE 1/,10X,"X(",K,") = ",K
1070 NEXT I
1080 IF N2=0 THEN 1130
1090 FOR I=1 TO N2
1100 PRINT USING 1110 ; I,G(I)
1110 IMAGE 1/,10X,"G(",K,") = ",K
1120 NEXT I
1130 PRINT @ PRINT @ PRINT "Termination based on:" @ PRINT
1140 SFLAG 10
1150 RETURN
2000 PRINT "Exceeded max. no. of iterations." @ RETURN
3000 PRINT "Excessive number of violated    constraints." @
RETURN
4000 PRINT "Failure to find a direction to  improve the desi
gn." @ RETURN
5000 PRINT "Failure to find a move parameter to improve the
design." @ RETURN
6000 PRINT "Convergence" @ RETURN
```

93

Listings of the edited versions of PROB which were used for the test cases presented in Chapter IV are included as examples of MDOT problem input. The subprogram used to enter the unconstrained test problem was renamed "BANANA", while that edited for the constrained problem was renamed "BEAM".

```
10 SUB "BANANA" (K1,X(),F,G(),L(),U())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
35 SHORT X1,X2,X3,X4,X5,X6,X7,X8,X9,X0
40 DIM L$[6],U$[6]
50 IF K1>1 THEN 220
90 READ N1,N2
100 DATA 2,0
105 IF K1=0 THEN SUBEXIT
130 FOR I=1 TO N1
140 READ X(I)
142 IF N2=0 THEN 170
144 READ L$,U$
150 IF L$="N" THEN L(I)=-1.E99 ELSE L(I)=VAL(L$)
160 IF U$="N" THEN U(I)=1.E99 ELSE U(I)=VAL(U$)
170 NEXT I
201 DATA -1
202 DATA 1.5
203 DATA
204 DATA
205 DATA
206 DATA
207 DATA
208 DATA
209 DATA
210 DATA
220 GOSUB 9010
230 ! User-defined expressions
399 ! Objective function
400 F=10*X1^4-20*X1^2*X2+10*X2^2+X1^2-2*X1+5
499 Q2=Q2+1
500 ! CONSTRAINTS
9000 SUBEND
9010 X1=X(1) @ IF N1=1 THEN RETURN
9020 X2=X(2) @ IF N1=2 THEN RETURN
9030 X3=X(3) @ IF N1=3 THEN RETURN
9040 X4=X(4) @ IF N1=4 THEN RETURN
9050 X5=X(5) @ IF N1=5 THEN RETURN
9060 X6=X(6) @ IF N1=6 THEN RETURN
9070 X7=X(7) @ IF N1=7 THEN RETURN
9080 X8=X(8) @ IF N1=8 THEN RETURN
9090 X9=X(9) @ IF N1=9 THEN RETURN
9100 X0=X(10) @ RETURN
```

```
10 SUB "BEAM" (K1,X(),F,G(),L(),U())
20 OPTION BASE 1
30 COM P$[6] ,INTEGER N1,N2,N3,N4,Q1,Q2,Q3,Q4,C1(10) ,SHORT
C2(20)
35 SHORT X1,X2,X3,X4,X5,X6,X7,X8,X9,X0
40 DIM L$[6],U$[6]
50 IF K1>1 THEN 220
90 READ N1,N2
100 DATA 2,3
105 IF K1=0 THEN SUBEXIT
110 REDIM L(N1),U(N1),X(N1),G(N2)
130 FOR I=1 TO N1
140 READ X(I)
142 IF N2=0 THEN 170
144 READ L$,U$
150 IF L$="N" THEN L(I)=-1.E99 ELSE L(I)=VAL(L$)
160 IF U$="N" THEN U(I)=1.E99 ELSE U(I)=VAL(U$)
170 NEXT I
201 DATA 3.5,.5,5
202 DATA 16,1,20
203 DATA
204 DATA
205 DATA
206 DATA
207 DATA
208 DATA
209 DATA
210 DATA
220 GOSUB 9010
230 ! User-defined expressions
301 B=X1
302 H=X2
379 ! Objective function
400 F=200*B*H
499 Q2=Q2+1
500 ! CONSTRAINTS
501 G(1)=600/(B*H^2)-1
502 G(2)=10666.7/(B*H^3)-1
503 G(3)=H/10-B
9000 SUBEND
9010 X1=X(1) @ IF N1=1 THEN RETURN
9020 X2=X(2) @ IF N1=2 THEN RETURN
9030 X3=X(3) @ IF N1=3 THEN RETURN
9040 X4=X(4) @ IF N1=4 THEN RETURN
9050 X5=X(5) @ IF N1=5 THEN RETURN
9060 X6=X(6) @ IF N1=6 THEN RETURN
9070 X7=X(7) @ IF N1=7 THEN RETURN
9080 X8=X(8) @ IF N1=8 THEN RETURN
9090 X9=X(9) @ IF N1=9 THEN RETURN
9100 X0=X(10) @ RETURN
```

LIST OF REFERENCES

1. Falk, H., "Software Tools for Mechanical Engineers", <u>Mechanical Engineering</u>, v. 105, no. 8, August 1983

2. Seireg, A., "From the Room-Size Computer to the Portable in Thirty Five Years", <u>Computers in Mechanical Engineering</u>, v. 2, no. 1, July 1983

3. Naval Postgraduate School Report NPS69-81-003, <u>COPES - A Fortran Control Program for Engineering Synthesis</u>, by L. Madsen and G. Vanderplaats, March 1982

4. Beakley, G., and Chilton, E., <u>Introduction to Engineering Design and Graphics</u>, Macmillan, 1973

5. Vanderplaats, G., <u>Numerical Optimization Techniques for Engineering Design: With Applications</u>, McGraw-Hill, 1984

6. Converse, A., <u>Optimization</u>, Holt, Rinehart and Winston, 1970

7. Kuester, J., and Mize, J., <u>Optimization Techniques with FORTRAN</u>, McGraw-Hill, 1973

8. Boot, J., <u>Quadratic Programming</u>, North-Holland, 1964

9. Zecher, J., et al, "Developing a Desktop Computer-Based Three Dimensional Modeling System", <u>Mechanical Engineering</u>, v. 105, no. 11, November, 1983

10. Hock, W., and Schittkowski, K., <u>Test Examples for Nonlinear Programming Codes</u>, Springer-Verlag, 1981

11. Adby, P., and Dempster, M., <u>Introduction to Optimization Methods</u>, John Wiley and Sons, 1974

BIBLIOGRAPHY

Albrecht, R., and Finkel, L., and Brown, J., BASIC,
John Wiley and Sons, 1978

Bazaraa, M., and Shetty, C., Nonliear Programming,
John Wiley and Sons, 1979

Conley, W., Computer Optimization Techniques,
Petrocelli, 1980

Fletcher, R., and Harwell, A., Optimization, Academic
Press, 1969

Fox, R., Optimization Methods for Engineering Design,
Addison-Wesley, 1971

Geoffrion, A., Perspectives on Optimization, Addison-
Wesley, 1972

Gerald, C., Applied Numerical Analysis, 2nd ed.,
Addison-Wesley, 1980

Hestenes, M., Conjugate Direction Methods in Optim-
ization, Springer-Verlag, 1980

Hornbeck, R., Numerical Methods, Quantum, 1975

Johnson, R., Optimum Design of Mechanical Elements,
John Wiley and Sons, 1961

Ketter, R., and Prawel, S., Modern Methods of Engine-
ering Computation, McGraw-Hill, 1969

Kunzi, H., et al, Numerical Methods of Mathematical
Optimization, Academic Press, 1971

Talbot, A., Approximation Theory, Academic Press, 1970

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, Virginia 22314

2. Library, Code 0142   2
   Naval Postgraduate School
   Monterey, California 93943

3. Department of Mechanical Engineering, Code 69   1
   Naval Postgraduate School
   Monterey, California 93943

4. G. N. Vanderplaats   5
   Department of Mechanical Engineering, Code 69
   Naval Postgraduate School
   Monterey, California 93943

5. Y. S. Shin   .   1
   Department of Mechanical Engineering, Code 69
   Naval Postgraduate School
   Monterey, California 93943

6. Prof. Lucien A. Schmit Jr.   1
   Department of Mechanics and Structures
   6731 Boelter Hall
   UCLA
   Los Angeles, California 90024

7. Dr. Steven Tsai   1
   3033 Locust Camp Road
   Dayton, Ohio 45419

8. Dr. Hirokazu Miura   1
   M.S. 237-11
   NASA Ames Research Center
   Moffett Field, California 94035

9. Dr. Jarek Sobieski   1
   M.S. 243
   NASA Langley Research Center
   Hampton, Virginia 23665

10. LT Richard L. Booth, USN   3
    105 Colleen Way
    Soquel, California 95073

11. Dr. Alan O. Lebeck   1
    Department of Mechanical Engineering
    University of New Mexico
    Albuquerque, New Mexico 87131

.