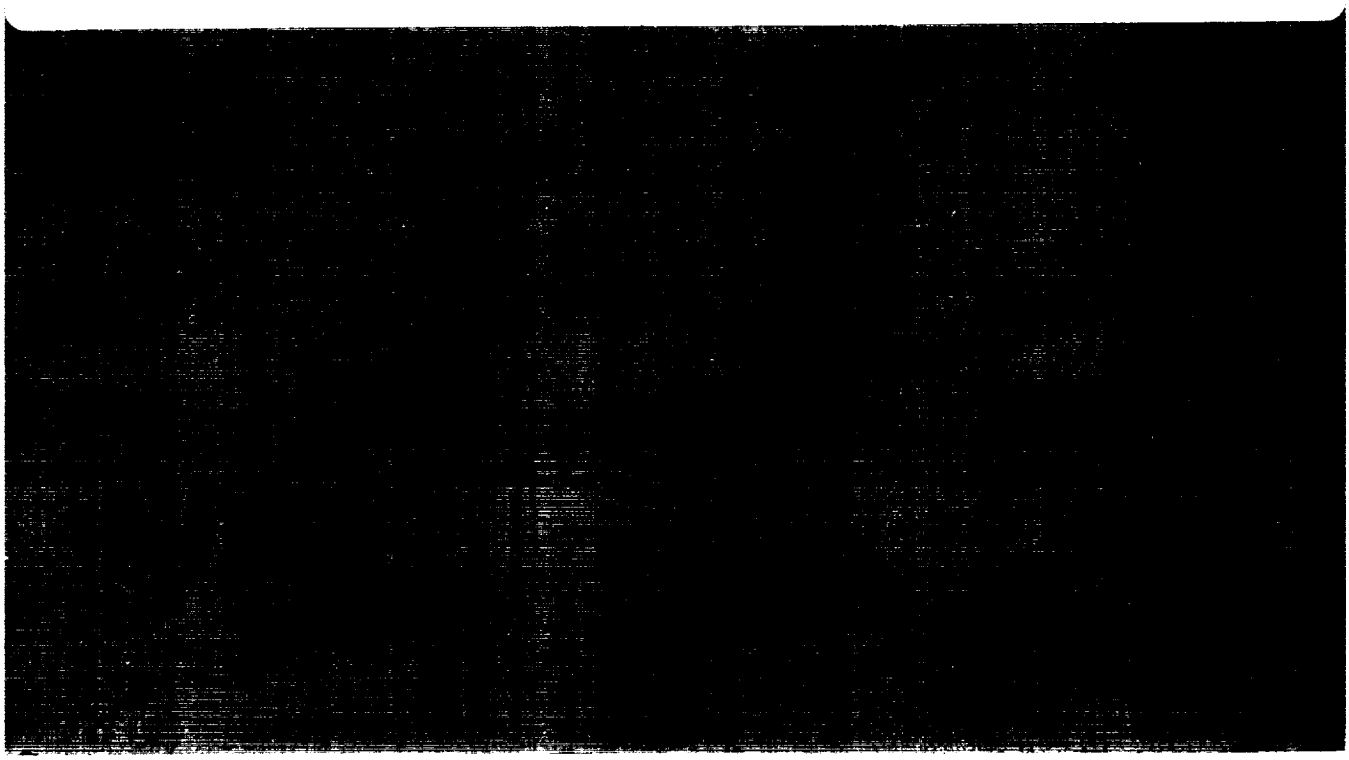
NASA Contractor Report 4220

An Indirect Method for Numerical Optimization Using the Kreisselmeier-Steinhauser Function

Gregory A. Wrenn

CONTRACT NAS1-18000
MARCH 1989



NASA Contractor Report 4220

An Indirect Method for Numerical Optimization Using the Kreisselmeier-Steinhauser Function

Gregory A. Wrenn
Planning Research Corporation
Aerospace Technologies Division
Hampton, Virginia

Prepared for
Langley Research Center
under Contract NAS1-18000

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1989

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

A	quadratic coefficient used to form quadratic minimum approximation
B	linear coefficient used to form quadratic minimum approximation
C	constant coefficient used to form quadratic minimum approximation
F_m	m^{th} objective function
F_m^*	m^{th} modified objective function
f	function used to form the KS function
g_j	j^{th} constraint function
J	number of constraints
K	number of functions used to form the KS function
KS	Kreisselmeier-Steinhauser function
M	number of objective functions
N	number of design variables
S	search direction vector
X	design variable vector
x_n	n^{th} design variable

Greek

α	distance traveled in the one-dimensional search direction
ρ	scalar multiplying factor used in the KS function

Subscripts

max	the maximum value of a set of functions evaluated at a point

Superscripts

0	evaluated at the beginning of an iteration
q	iteration number
*	the minimum point

SUMMARY

This paper describes a new technique for converting a constrained optimization problem into an unconstrained problem. The technique transforms one or more objective functions into reduced objective functions, which are analogous to goal constraints used in the goal programming method. These reduced objective functions are appended to the set of constraints and an envelope of the entire function set is computed using the Kreisselmeier-Steinhauser function. This envelope function is then searched for an unconstrained minimum. The technique may be categorized as a SUMT algorithm. Advantages of this approach are the use of unconstrained optimization methods to find a constrained minimum without the "draw down" factor typical of penalty function methods, and that the technique may be started from the feasible or infeasible design space. In multiobjective applications, the approach has the advantage of locating a compromise minimum design without the need to optimize for each individual objective function separately.

A description of the constrained to unconstrained conversion technique for multiobjective problems is presented. This is followed by a description of the computer program KSOPT, which implements the method as a general purpose optimization program written in the FORTRAN language. A description of each subroutine in KSOPT is included. Finally, example problems are solved demonstrating the use of KSOPT and the robustness of the method.

INTRODUCTION

The purpose of this paper is threefold, first to describe a new technique for converting a constrained optimization problem to an unconstrained one, second to discuss how the method is naturally extended to multiobjective optimization problems, and finally to present the computer program KSOPT as a general purpose FORTRAN subroutine which implements this new technique. A description of each subroutine in KSOPT is included in appendix A.

The conversion technique used in this new method employing the Kreisselmeier-Steinhauser function[1], which will be referred to as the KS method, may be categorized as a member of the Sequential Unconstrained Minimization Technique (SUMT) class of optimization methods[2]. SUMT methods in general convert a constrained optimization problem into an unconstrained one using a penalty term[3,4], composed from the constraint equations, which is summed with an objective function. Problems associated with most of these penalty methods are that they do not possess continuous first or second derivatives, they must be started from a particular region of the design space, and the user is required to specify a multiplying factor whose value is critical to the behavior of the optimization method. Many improvements have been made since interior and exterior penalty methods were first used. The quadratic extended interior penalty function method[5], in fact, has eliminated all of the above mentioned problems except the need for a user supplied multiplying or "draw-down" factor. The KS method described in this paper possesses the good qualities of the extended penalty function, namely the continuity of derivatives and that the optimization may be started from either the feasible or infeasible design regions. The draw-down factor of the penalty function method is not required in the KS method, but there is a user supplied factor required. The difference in these factors will be discussed in detail later in the paper.

In the design of complex engineering systems, it is often difficult to select a single objective function which will satisfy all of the desired design requirements. Several techniques have previously been developed and described for multiobjective optimization[6]. Most of the previous efforts to include multiple objective functions in a numerical optimization method can be separated

into two techniques. The first is to form a single composite objective function as some combination of the original objective functions such as the summation of each objective function multiplied by a judgmental weighting factor. An example of this approach is the utility function method. The second approach is to solve the optimization problem once for each single objective function and to use the resulting optimum objective function or design variable vector as a target, solving an additional optimization problem to attain a suitable compromise. Examples of this approach are the global criterion formulation, game theory approach, goal programming method, and goal attainment method.

The KS method is inherently capable of optimizing for multiple objective functions. Any number of objective functions and constraints are combined using the KS function to form a single composite function. This composite function is then used to solve the optimization problem. The primary advantage of the KS method for multiobjective optimization is the elimination of potentially expensive separate optimizations for each objective function. In addition, each objective function may retain its original scaling and units, such as a cost or return on investment objective in dollars and a weight objective in pounds.

CONSTRAINED TO UNCONSTRAINED CONVERSION

Before the new KS method conversion technique is described, a brief review of the classical constrained optimization problem is appropriate. Only inequality constraints will be considered in this discussion, because the KS method does not handle equality constraints directly. The typical definition of a constrained optimization problem is

$$\begin{aligned} &\text{Min } F_m(X) \quad m=1,2,3,...M \\ &g_j(X) \le 0 \quad j=1,2,3,...J \\ &X = x_1,x_2,x_3,...,x_N \end{aligned} \tag{1}$$

where the inequality constraints are defined to be violated if they have positive values and satisfied if they have negative values. A typical formulation for these constraints would be

$$\begin{aligned} g_j(X) &= \frac{\text{DEMAND}(X)}{\text{CAPACITY}} - 1 \\ g_j(X) &= 1 - \frac{\text{DEMAND}(X)}{\text{CAPACITY}} \end{aligned} \tag{2}$$

The first form imposes an upper limit and the second form imposes a lower limit on the value of the DEMAND quantity. This formulation scales the constraint value to be in a small region, usually between -1 and 1. This constraint scaling is not absolutely necessary for the KS method, but improves the computational characteristics of the method by reducing problems associated with large numbers.

The KS method combines one or more objective functions with all of the inequality constraints to form a single composite KS function. An unconstrained optimization algorithm is then used to find the minimum of this composite function, equivalent to other SUMT methods. The KS function was first used by Kreisselmeier and Steinhauser[1] and is defined as

$$KS(X) = \frac{1}{\rho} \log_e \sum_{k=1}^{K} e^{\rho f_k(X)} \tag{3}$$

An alternate definition which reduces numerical difficulties caused by computing the exponential of large numbers is

$$KS(X) = f_{max} + \frac{1}{\rho} \log_e \sum_{k=1}^{K} e^{(\rho f_k(X) - f_{max})} \tag{4}$$

where $f_k(X)$ is a set of K functions, which in the context of this discussion are the objective functions and constraints, and f_{max} is the maximum value of the set of functions evaluated at X and taken to be a constant. Both forms of the KS function produce the same results, subject to round-off errors. In addition, the first derivatives with respect to the design variable set X for both KS formulations have identical values, again subject to round-off errors. The first derivatives for both KS formulations are

$$\frac{\partial KS(X)}{\partial x_n} = \frac{\sum_{k=1}^{K} e^{\rho f_k(X)} \frac{\partial f_k(X)}{\partial x_n}}{\sum_{k=1}^{K} e^{\rho f_k(X)}}$$

$$\frac{\partial KS(X)}{\partial x_n} = \frac{\sum_{k=1}^{K} e^{(\rho f_k(X) - f_{max})} \frac{\partial f_k(X)}{\partial x_n}}{\sum_{k=1}^{K} e^{(\rho f_k(X) - f_{max})}} \tag{5}$$

which require the current value of each function $f_k(X)$ and its first partial derivatives with respect to the design variable set X. The second form of the KS function, equation (4), will be used in the rest of this paper due to its superior numerical behavior.

The KS function defines an envelope surface in N-dimensional design space replacing the J constraint surfaces and the M objective function surfaces. This envelope surface is continuously differentiable and represents the maximum of the set of functions. The KS function has a property such that

$$f_{max} < KS < f_{max} + \frac{\log_e(K)}{\rho} \tag{6}$$

The term on the right hand side of equation (6) represents the worst case value of the KS function, which occurs at a point where all of the functions intersect. In practice, the KS function surface is influenced by only a few functions at any one point and the KS surface will be only slightly above the maximum of the individual surfaces, approaching the maximum surface as the scalar multiplier

ρ increases. Typical values of the scalar ρ are between 5 and 200. Figure 1a shows a set of curves of one variable and the corresponding KS functions indicated for several values of the scalar ρ. Figure 1b shows in greater detail the region circled in figure 1a, where curve 1 and curve 2 intersect. These figures graphically show the ability of the KS function to form a single continuous curve following the maximum of a set of curves. Note that the KS function provides a smooth transition from one curve to another. Larger values of the scalar ρ "pull" the KS function curve closer to the actual curve intersections, with correspondingly sharper gradient transition in these regions. This is functionally similar to the draw-down factor of penalty function methods. The draw-down factor controls the relationship between constraints and the objective function of penalty function methods whereas in the KS method the multiplier ρ controls the distance from the KS surface to the surface of maximum function value.

Converting a constrained optimization problem to an unconstrained problem requires that the objective functions be combined with the constraints in some manner such as is done with penalty function methods. The resulting composite function is then minimized using an unconstrained minimization technique. Several iterations of calculating the composite function, determining a search direction, and performing a one-dimensional minimization are needed to converge to the optimum solution. The technique used in the KS method for forming the composite function is first to scale and offset each objective function such that its value at the start of an iteration is the same as the maximum constraint value but of opposite sign. As mentioned earlier, the constraints are assumed to be scaled already. The modified objective function stated numerically is

$$F_m^*(X) = \frac{F_m(X)}{F_m^0} - 1 - g_{max} \tag{7}$$

where F_m^0 is the value of the m^{th} objective function at the beginning of an iteration and g_{max} is the maximum value of the set of inequality constraints evaluated at X and taken to be a constant for this iteration. The value of $F_m^*(X)$, the modified objective function, at the beginning of an iteration is equal to the negative of g_{max} as shown in figure 2a. For the case of multiple objective functions,

the modified objective functions at the start of an iteration are shown in figure 2b. Note that all of the objective functions intersect at the starting point of the iteration. Each scaled and offset objective function is appended to the set of inequality constraints $g_j(X)$ to form a set of K functions written as $f_k(X)$ where K is the sum of the number of objective functions M and the number of constraints J.

$$\begin{aligned} f_1(X) &= F_1^*(X) \\ f_2(X) &= F_2^*(X) \\ &\vdots \\ f_M(X) &= F_M^*(X) \\ f_{M+1}(X) &= g_1(X) \\ f_{M+2}(X) &= g_2(X) \\ &\vdots \\ f_{M+J}(X) &= g_J(X) \end{aligned} \tag{8}$$

These functions are then combined using the KS function of equation (4) to form a single continuous composite KS function. The composite KS function represents the envelope of all objective functions and constraints in the optimization problem, and is used with an unconstrained optimization technique to find its minimum. Several iterations are usually required to find the optimum solution to a problem. Since every objective function's scale and offset values are selected at the beginning of each iteration, the composite KS function surface will be different for every iteration.

KSOPT GENERAL PURPOSE OPTIMIZER

The KS method has been incorporated into the general purpose constrained minimization computer program KSOPT, written in the FORTRAN programming language. The program was written with the requirements of providing a simple user interface, have the ability to be restarted in the middle of an optimization problem, and be written using a modular approach. Modular programming techniques allow portions of a program to be replaced easily as new and improved methods and algorithms are developed. In KSOPT the search direction algorithm and the one-dimensional minimization algorithm are examples of modules which might be replaced. The restarting requirement satisfies a need to be able to switch to a large, complex analysis procedure which would require all of the computing resources available, and then to restart the optimization procedure later using the analysis results. The user interface consists of two subroutines, one for initializing the optimization system and the other to perform the optimization (see appendix A for a complete description of each subroutine). This program is used much like any other constrained minimization program, where the user supplies arrays containing the design variables, objective functions, and constraints. The restarting capability is provided by the user supplying a WORK array, with sufficient computer storage space allocated, which will contain all of the constants and arrays used by the optimizer. The user may save this array to some external storage device when leaving the optimizer and restore the array before continuing with the optimization problem.

The user interface is shown in block diagram form in figure 3 and consists of the subroutines KSINIT and KSOPT. Subroutine KSINIT initializes the optimizer with constant parameters and initial values of the design variables. The user must supply the desired parameter values and dimension the arrays sufficiently large for the specific problem being solved. The minimum size requirements for each array are specified in appendix A in the description of subroutine KSINIT. Subroutine KSOPT is called iteratively, where KSOPT returns the current design variable vector and a flag, ISEL, indicating what action the user should take. If the flag is set to 1, the user must compute the objective functions and constraints for the current design variables. If the flag is 2, the user must compute the gradients of the objective functions and

constraints with respect to the current design variables. After either one of these computations is complete, the user returns the computed quantities to subroutine KSOPT. When the flag ISEL is zero, the optimization is terminated and the optimum design variable vector is returned to the user. For the user that wishes to know what the internal parameters of the KSOPT program are while solving their optimization problem, the common block KSCOMM can be included in the user's analysis program. A description of each variable in this common block is given in appendix A. The user must be careful not to alter the values of any parameters in this common block. Appendix B shows a sample FORTRAN program outline for solving an optimization problem, and is intended only as a guide.

Figure 4 is a flow chart detailing the major functional blocks of the optimization program. First, subroutine KSINIT is called to initialize the optimization problem, allocate space for internal arrays, and test the initial design variable vectors for bounds violations. Then subroutine KSOPT is called in a loop with the user supplied analysis procedures until the optimization problem is solved. Internally KSOPT converts the constrained minimization problem to an unconstrained problem using the KS method, and solves the unconstrained problem iteratively. Each iteration, as shown in figure 4, involves first obtaining the initial objective function and constraint values from user supplied analyses. Then gradients of the objective functions and constraints with respect to the current design variables are obtained either from the user explicitly or by finite differences using the user supplied analysis results. KSOPT then forms the composite KS function described earlier in the constrained to unconstrained conversion technique, and the gradients of the composite KS function with respect to the design variables. This single composite KS function and its gradients define the unconstrained optimization problem to be solved. The unconstrained problem for each iteration is then solved by first determining a search direction vector S using the Davidon-Fletcher-Powell[7] (DFP) algorithm. With the search direction determined, a one-dimensional search for the minimum of the composite KS function is performed using

$$X^q = X^{q-1} + \alpha^* S^q \qquad (9)$$

where q denotes the current iteration of the optimization procedure. The optimum step length α^* is defined as the distance travelled in the S^q direction that makes the composite KS function a minimum without violating any side constraints. The optimum length α^* is found by selecting a trial value for α, using equation (9) to compute a new design variable vector X, having the user supply the new objective functions and constraints, and computing the new composite KS function as seen in figure 4. The procedure continues stepping in the S^q direction until the composite KS function no longer decreases in value. Then a three point quadratic polynomial approximation[3] using the equation

$$A\alpha^2 + B\alpha + C = KS(\alpha) \tag{10}$$

is formed using the last three trial α values and their corresponding composite KS function values. When the set of three simultaneous linear algebraic equations are solved for the coefficients A, B, and C, the value of α^* may be computed by differentiating equation (10) and setting the result equal to zero

$$2A\alpha^* + B = 0 \tag{11}$$

and the corresponding design variable vector is found using equation (9). The quadratic polynomial approximation is repeated with additional trial design points until convergence is achieved for the one-dimensional minimization, ending the current iteration. At the end of each iteration a set of termination criteria are tested, and if they are not satisfied another iteraton is performed using the final design variables from the previous iteration as the starting point. The three termination criteria available in KSOPT are

1) Relative change of the composite KS function is less than some prescribed amount for three consecutive iterations.
2) Absolute change of the composite KS function is less than some prescribed amount for three consecutive iterations.
3) The maximum number of iterations is exceeded.

When any one of these termination criteria is satisfied, the optimization is terminated and the final set of design variables are returned to the user.

This implementation of the unconstrained minimization procedure treats side constraints separately from the other constraints. Often it is desirable to approach side constraints as closely as possible without violating them. Forming side constraints the same way as those of equations (2) do not allow many optimization methods to approach the side constraints closely. In KSOPT, side constraints are not included in the composite KS function but instead are tracked individually and the one-dimensional minimization discussed above is terminated when a side constraint is encountered. At the beginning of the next iteration a new search direction is computed. If the proposed search direction would violate any currently active side constraints, the components of the search direction vector corresponding to those design variables are set to zero and the DFP algorithm is restarted using a steepest descent gradient. As long as no new side constraints become active, the DFP algorithm continues to perform as usual while essentially using a subset of the design variable vector. This method allows the DFP scheme to continue updating the Hessian matrix, thus gaining the benefits of that algorithm, while following side constraints exactly. If a design variable which was previously on a side constraint is directed back into the feasible design space, its contribution to the Hessian matrix is once again included.

EXAMPLE PROBLEMS

A simple one-dimensional optimization problem using a single design variable, one objective function, and two constraints will be used for the first example. This problem was chosen because its optimization can be clearly shown graphically to aid in understanding the KS method. The objective function is defined as

$$F(x) = \frac{x^2}{20} - \frac{3x}{5} + \frac{5}{2} \tag{12}$$

and the two constraints are defined as

$$g_1(x) = \frac{5}{\log_e(x)} - \frac{x}{5} - 4 \tag{13}$$

$$g_2(x) = \frac{x^2}{40} + \frac{x}{5} - 2 \tag{14}$$

Analytical gradients are used in the optimization of this problem because both the objective function and constraints are easily differentiated with respect to the single design variable x. The optimization is started at point a of figure 5a, and the first iteration begins by computing the modified objective function of equation (7). The scaled and offset objective function and the two constraints are used to compute the composite KS function in figure 5b. The unconstrained DFP search direction algorithm is then employed along with a one-dimensional minimization procedure to find the minimum of the composite KS function curve, resulting in the iteration terminating at point b where the composite KS function is minimum. Since the maximum constraint at point a was violated, the KS method essentially removes the objective function from consideration for this iteration and works to minimize the constraint violation. The minimum of the composite KS function in figure 5b is the point where the two constraints intersect. The next iteration is started at this point by scaling and offsetting the objective function again and computing the new composite KS function shown in figures 6a and 6b. At the beginning of this iteration the constraints are satisfied, and the objective function guides the design. The composite KS function shown in figure 6b for this case is minimized, terminating at point c. The next two iterations are shown in figures 7a through 8b. The design is being driven toward the point where constraint g_2 is zero,

and the design oscillates between being dominated by the objective function and constraint g_2. At the optimum point e in figure 9a, which for this example is the point where g_2 is zero, the scaled and offset objective function and constraint g_2 are coincident because g_{max} is zero. The composite KS function is influenced equally by the objective function and constraint g_2, as seen in figure 9b. At this point the composite KS function is a minimum, and constraint g_2 is zero. Three optimizations were performed for this example problem, all starting at the same initial design point but using different initial values and ranges of the multiplying factor ρ. Table 1 summarizes the results for each case showing the influence of the multiplier ρ on the optimum solution and the number of iterations required. As can be seen in table 1, larger values of ρ result in a smaller optimum objective function and a value of constraint g_2 closer to zero, but at the price of additional iterations.

The second example is the classical three bar truss problem[8] modified to use steel and titanium truss members, with two design variables, two objective functions, and six constraints[9]. This design problem was chosen to demonstrate the interaction of different material types in designing a truss for multiple objectives. A description of the analysis procedure for the three bar truss is given by Vanderplaats[3] starting on page 252. A diagram of the three bar truss, including the material properties for steel and titanium and the two independently applied loads, is shown in figure 10. The desired optimum solution is the three bar truss design which will simultaneously minimize truss weight and material costs. The truss is symmetrical with the outer members made of steel and the inner member made of titanium. The design variables are the cross sectional areas of the truss members, denoted as A1 and A2 in figure 10. These areas are not allowed to be less than 0.001 square inches. The constraints are yield stress limits on the members, with three constraints for tensile loads and three for compressive loads. The first objective function is the total weight of the three truss members. Since titanium is lighter than steel the minimum weight design would use a larger titanium center member and smaller steel outer members. The second objective function is total material cost. Here titanium is 60 times more expensive than steel, leading to a design having a small titanium center member and larger outer steel members. The

optimum design when both objective functions are considered will be some compromise between low weight and low cost. To graphically illustrate this design problem, contour plots will be used over a portion of the design space. The weight objective function alone is shown in figure 11 as a contour plot of total truss weight versus inner and outer rod areas. Figure 12 shows the contour plot of the cost objective function alone versus inner and outer rod areas. Only the tensile yield stress constraint on the outer truss members is critical in this optimization problem, and a contour plot of its zero value is shown in figure 13.

In order to see what the limits of the multiple objective optimum design were, the three bar truss was optimized for each objective function separately. Table 2 shows the results of optimizing the truss for weight only, for cost only, and for weight and cost simultaneously. Shown along with the final results are the total number of iterations and the total number of analyses required. Each optimization problem was started with both the inner and outer members having a cross sectional area of one square inch. The final results of table 2 show that the multiple objective function design is identical to the minimum cost design. The cost of titanium is 60 times greater than steel, but steel weighs only 1.76 times greater than titanium. Therefore, the optimizer accepted a weight penalty of 0.24 pounds in order to reduce the cost of the truss by 15.50 dollars. Although the results are not presented in this report, reducing the cost of titanium to 4 dollars per pound of weight starts to move the cross sectional area of the titanium member away from its minimum value. Further reduction of the cost of titanium produces optimum solutions which use more titanium to reduce weight. When the cost of titanium is reduced to that of steel, the optimum solution is identical to the weight only solution. It is therefore possible to change the multiobjective optimum solution by weighting the objective functions.

To illustrate the minimization of the multiple objective function design problem, contour plots of the design space of the first six iterations are shown in figures 14 through 19. The first iteration starts in figure 14 with the cross sectional areas of the truss members at one square inch. The line shown is the direction computed by the DFP algorithm for a one-dimensional search, and terminates at the arrowhead where a minimum is found in this direction. Figure 15 shows a

contour plot of the design space for the second iteration. Note that the contours of the design space change from one iteration to the next. This is because in the KS method the modified objective function of equation (7) is computed at the beginning of each iteration, and, since the objective functions are dependent on the current design variable values, the composite KS function is different for every iteration. Iterations 3 through 6 are shown in figures 16 through 19. Iteration 6 terminates at the point where the titanium inner truss member reaches its minimum size. The remaining iterations make only small refinements to this design. The optimum solution is shown in figure 20 as a contour plot of the design space about the optimum point. No further progress can be made without either violating a yield stress constraint or increasing one of the objective functions.

CONCLUDING REMARKS

A new technique for converting a constrained optimization problem into an unconstrained one using the Kreisselmeier-Steinhauser function was presented. This technique, referred to as the KS method, can be categorized as a Sequential Unconstrained Minimization Technique similar to penalty function methods. This new approach uses the KS function to combine one or more objective functions with all of the problem constraints to form a single continuously differentiable envelope function. This envelope function is then minimized using unconstrained optimization techniques. This method offers several advantages compared to penalty function methods. One advantage of the KS method is the inherent ability to optimize for multiple objectives, without the need to perform an optimization on each objective function separately. Also, the design may be started from a feasible or infeasible region of the design space, and the "draw-down" factor typical of penalty function methods is eliminated.

The general purpose optimization computer program KSOPT, which solves constrained optimization problems using the KS method, was also described. The KSOPT program can solve single objective function and multiple objective function constrained minimization problems. It is written in the FORTRAN programming language. A description of each subroutine in KSOPT is provided. The program was developed with ease of use, a restarting capability, and modularity as design goals, providing users with a simple interface to the KS optimization method. The restarting capability provides the user with a convenient way to save the state of the optimization procedure in the middle of a design, and continue with the optimization later from the point where it was saved. Modularity of the program code allows the replacement of portions of the optimization program such as an alternate search direction method or one-dimensional minimization algorithm. The KSOPT program currently uses the DFP algorithm to find the unconstrained one-dimensional search direction, and a three point quadratic polynomial approximation to find the unconstrained minimum for each iteration. The user selectable termination criteria available are relative and absolute change in the unconstrained KS function and an upper limit on the number of optimization iterations that are allowed.

Two sample optimization problems were presented to demonstrate the ability of the KS method to solve single and multiple objective function problems. The first example was an academic problem with a single objective function, two constraints, and one design variable. It was chosen because the optimization method could be clearly displayed graphically. The second example was the classical three bar truss structural optimization problem extended to include both weight and cost as objective functions, with six constraints and two design variables. This problem demonstrated the KS method's ability to find a compromise optimum solution while considering opposing objectives and using truss members with different material properties. The results of both example problems were shown graphically step by step to further describe the KS method.

APPENDIX A

KSOPT SUBROUTINE DESCRIPTIONS

This appendix describes each FORTRAN subroutine used in the general purpose optimization program KSOPT. The first two subroutine descriptions, KSINIT and KSOPT, are intended as a reference for the user in choosing what values to assign parameters and what storage requirements are needed for the arrays. Most of the constant parameters have default values indicated, and the default selections are always taken when the user supplies a zero value for a parameter. The remaining subroutine descriptions are provided so that a user may gain an understanding of the internal processes of the optimization program. Enough information is provided for each subroutine so that modifications such as using an alternate one-dimensional search algorithm are possible. This appendix corresponds to version 2.1 of the software, dated February 7, 1989.

SUBROUTINE KSINIT

PURPOSE: This subroutine initializes all of the optimization parameters based on user supplied input and sets the default values of parameters the user does not set. It also determines the amount of temporary storage required for these parameters plus all of the arrays required during optimization and computes the pointers to the start of each array. Finally, this subroutine scales the initial design variable set, tests the design variables for side constraint violations, and prints the initial design data.

USAGE: CALL KSINIT (X, XLB, XUB, SCALE, WORK, NDV, NCON, NOBJ, NSIDE, NSCALE, IPRNT, ITMAX, IGRAD, ISDRST, RDFUN, ADFUN, FDELT, FDMIN, RHOMIN, RHOMAX, RHODEL, NUNIT, IREQ)

INPUT PARAMETERS:

X -- Initial design variable array, dimensioned at least NDV words long in the calling program.

XLB -- Array of lower bounds on the design variables, dimensioned at least NDV words long in the calling program.

XUB -- Array of upper bounds on the design variables, dimensioned at least NDV words long in the calling program.

SCALE -- Array of user supplied design variable scale values, which are used to scale the design variable vector every NSCALE iterations. It is dimensioned at least NDV in the calling program.

WORK -- A working array which holds the 56 values contained in the KSCOMM common block, plus all of the temporary arrays used during the optimization. WORK must be dimensioned large enough in the calling program to hold all of these values. The parameter IREQ, computed here, holds the minimum size in single precision words that are required for the proposed optimization problem. No test is made to determine whether or not the WORK array is large enough, so the user must check this value and adjust the dimension of the WORK array accordingly. The array should be dimensioned in the calling program according to the following formula:

SIZE = 58 + 3*NOBJ + NCON + 11*NDV + NDV*(NDV-1)/2 + NOBJ*NDV + NCON*NDV + 2*MAX(2*NDV,NOBJ+NCON)

NDV -- The number of design variables, which must be at least one.

NCON -- The number of constraints, which may be zero.

NOBJ -- The number of objective functions, which must be at least one.

NSIDE -- A non-zero value indicates that side constraints are to be considered during optimization.

NSCALE -- A flag indicating which design variable scaling option is to be used.
- =0 -- No scaling.
- <0 -- Scale the design variables using the user supplied vector SCALE.
- >0 -- Scale the design variables using the current design variable set as the scale factors. The design variables are re-scaled every NSCALE iterations.

IPRNT -- A flag indicating the print level, a 3 digit number.

Hundreds digit - One-dimensional search print:
- 0 = no print
- 1 = print alpha and KS function at each proposed alpha
- 2 = print design variable vector at each proposed alpha

Tens digit - Gradient print:
- 0 = no print
- 1 = print gradients of objective functions and constraints for every iteration that design variable data is printed

Ones Digit - Iteration data print:
- 0 = no print
- 1 = print initial and final iteration data
- 2 = print all iterations
- 3 = also include the proposed search direction and the slope
- 4 = also include the approximate Hessian matrix

ITMAX -- The maximum number of iterations. If a zero is input, the default of 20 iterations is used.

IGRAD -- A flag selecting the method of gradient computation.
- 0 = gradients of the objective functions and constraints are computed by finite differences from within KSOPT.
- 1 = the user supplies all gradient information.

ISDRST -- The number of iterations that occur before the search direction finding process is restarted using a steepest descent method. If input as zero, the default of NDV+1 is used.

RDFUN -- Termination criteria. When the relative change in the KS function is less than this value for three consecutive iterations, the optimization is terminated. If a zero is input, the default of 0.0001 is used.

ADFUN -- Termination criteria. When the absolute change in the KS function is less than this value for three consecutive iterations, the optimization is terminated. The default is zero, which means this termination criteria is not used.

FDELT -- The step size used when computing finite difference derivatives. If a zero is input, the default value of 0.01 is used, meaning a 1 percent step size.

FDMIN -- The minimum absolute step size used for finite difference computations. If a zero is input, the default value of 0.0001 is used.

RHOMIN -- The initial value of the RHO multiplier used in the KS function. If a zero is input, the default value of 5.0 is used.

RHOMAX -- The maximum value of the RHO multiplier used in the KS function. If a zero is input, the default value of 100.0 is used.

RHODEL -- This value is used to increment RHO during optimization. If a zero is input, the default is computed based on the values of RHOMIN and RHOMAX. In this case, RHODEL will be between 10.0 and 40.0. The user however may set RHODEL to any positive value. If the user wants RHODEL to be zero, then the values for RHOMIN and RHOMAX should be set equal to each other.

NUNIT -- This is the FORTRAN file unit number to be used for all printed output.

OUTPUT PARAMETERS:

IREQ -- The required length of the WORK array.

NOTES: This subroutine is called by the user to set up an optimization problem. The WORK array is loaded with initial values, and must not be altered until after the completion of the optimization.

SUBROUTINES CALLED: KSCOMP

SUBROUTINE KSOPT

PURPOSE: This subroutine is the main interface between the optimizer and the user. It is called by the user, who supplies values of the objective functions, constraints, and gradients if the appropriate options are selected. KSOPT then performs the optimization, returning an updated vector of design variables to the user whenever the objective functions, constraints, or gradients need to be computed.

USAGE: CALL KSOPT (ISEL, X, OBJ, G, DF, DG, NOMAX, NGMAX, WORK)

INPUT PARAMETERS:

OBJ -- Array of objective functions, dimensioned at least NOBJ words long in the calling program.

G -- Array of constraints dimensioned at least NCON words long in the calling program.

DF -- Matrix of the gradients of the objective functions. A two-dimensional array with the first dimension being the maximum number of objective functions and the second dimension being the number of design variables.

DG -- Matrix of the gradients of the constraints. A two-dimensional array with the first dimension being the maximum number of constraints and the second dimension being the number of design variables.

NOMAX -- The first dimension of the DF array as defined in the calling program.

NGMAX -- The first dimension of the DG array as defined in the calling program.

WORK -- The work array as defined in subroutine KSINIT. This array must not be changed by the user during the optimization.

OUTPUT PARAMETERS:

ISEL -- Flag requesting user supplied information.
- 0 = Optimization terminated, user should exit.
- 1 = User computes objective functions and constraints corresponding to the current X-vector.
- 2 = User computes gradients of the objective functions and constraints corresponding to the current X-vector. This option will only be used if IGRAD is set to 1 (see KSINIT).

X -- Current set of design variables.

NOTES: This subroutine is called in a loop combined with the user's analysis procedures. The output parameter ISEL should be tested to determine what action the user's program should take. An example is:

```
      PROGRAM TEST
      DIMENSION ...

      CALL KSINIT (...)
10    CONTINUE

      CALL KSOPT (ISEL,X,OBJ,G,DF,DG,NOMAX,NGMAX,WORK)

      IF (ISEL .EQ. 1) THEN

                  ... USER SUPPLIED ANALYSIS ...

      GO TO 10

      ENDIF

      IF (ISEL .EQ. 2) THEN

                  ... USER SUPPLIED GRADIENTS ...


      GO TO 10

      ENDIF

      OTHERWISE ASSUME ISEL IS ZERO, CONTINUE WITH THE REST
      OF YOUR PROGRAM.
```

SUBROUTINES CALLED:

KSANDO
KSCOMG
KSCOMP
KSDFP
KSDFUN
KSFUN
KSGRAD
KSONED
KSPRNT
KSSCAL
KSSIDE
KSUNSC
KSXLIM

SUBROUTINE KSCOMG

PURPOSE: Copy values from the WORK array to the common block KSCOMM. The common block values are stored in the WORK array for ease of restarting an optimization by requiring that only the WORK array be saved.

USAGE: CALL KSCOMG (WORK)

INPUT PARAMETERS:

WORK -- The work array as defined in subroutine KSINIT.

OUTPUT PARAMETERS:

none

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSCOMP

PURPOSE: Copy values from the common block KSCOMM to the WORK array. The common block values are stored in the WORK array for ease of restarting an optimization by requiring that only the WORK array be save.

USAGE: CALL KSCOMP (WORK)

INPUT PARAMETERS:

WORK -- The work array as defined in subroutine KSINIT.

OUTPUT PARAMETERS:

none

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSPRNT

PURPOSE: This subroutine contains most of the print statements used in the optimizer. It prints the user selected data pertaining to each iteration at the beginning of the iteration.

USAGE: CALL KSPRNT (IPFLAG, IPRNT1, IPRNT2, X, OBJ, G, DF, DG, ISID, SCALE, NOMAX, NGMAX, ITEMP, WORK)

INPUT PARAMETERS:

IPFLAG -- A flag to indicate whether to print normal iteration data or the final optimization data.
1 = print the iteration data
2 = print the final optimization data

IPRNT1 -- Ones digit of the IPRNT flag from subroutine KSINIT.

IPRNT2 -- Tens digit of the IPRNT flag from subroutine KSINIT.

X -- Current set of design variables.

OBJ -- Current vector of objective function values.

G -- Current set of constraints.

DF -- Current matrix of objective function gradients.

DG -- Current matrix of constraint gradients.

ISID -- Integer array containing flags indicating whether side constraints have been encountered for each design variable.

SCALE -- Current design variable scale values.

NOMAX -- Number of objective functions.

NGMAX -- Number of constraints.

ITEMP -- A temporary integer array dimensioned at least 2 * NDV in the calling program.

WORK -- The work array as defined in subroutine KSINIT.

OUTPUT PARAMETERS:

none

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: KSUNSC

SUBROUTINE KSSIDE

PURPOSE: This subroutine checks the side constraints imposed on the design variables and sets a vector of flags to indicate bounds activity. In addition, if a design variable is at a side constraint, and the gradient of the KS function for that particular design variable would move the design past the side constraint, then that term of the gradient vector is set to zero.

USAGE: CALL KSSIDE (X, XLB, XUB, ISIDE, DFUN, NDV, NSIDE)

INPUT PARAMETERS:

X -- Current scaled design variable vector.

XLB -- Scaled lower bounds values for design variables.

XUB -- Scaled upper bounds values for design variables.

DFUN -- Array of gradients of the KS function.

NDV -- Number of design variables.

NSIDE -- Flag indicating whether side constraints are to be considered (see KSINIT).

OUTPUT PARAMETERS:

ISIDE -- Integer array containing flags indicating whether side constraints have been encountered for each design variable.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSSCAL

PURPOSE: This subroutine computes a new design variable scaling vector, if automatic scaling was selected in subroutine KSINIT, and re-scales the design variables and the upper and plower bounds.

USAGE: CALL KSSCAL (X, X0, XLB, XUB, SCALE, NDV, NSIDE, NSCALE)

INPUT PARAMETERS:

X	-- Current scaled design variable vector.
X0	-- Scaled design variable vector from beginning of the current iteration.
XLB	-- Scaled lower bounds values for design variables.
XUB	-- Scaled upper bounds values for design variables.
SCALE	-- Vector of current design variable scale values.
NDV	-- Number of design variables.
NSIDE	-- Flag indicating whether side constraints are to be considered (see KSINIT).
NSCALE	-- Flag indicating which design variable scaling method is being used (see KSINIT).

OUTPUT PARAMETERS:

X	-- Current design variable vector with new scale factors applied.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSUNSC

PURPOSE: This subroutine converts the scaled design variable vector used in KSOPT to the unscaled vector used by the user for analysis.

USAGE: CALL KSUNSC (X, SX, SCALE, NDV)

INPUT PARAMETERS:

SX -- Vector of scaled design variables.

SCALE -- Vector of design variable scale factors.

NDV -- Number of design variables.

OUTPUT PARAMETERS:

X -- Current design variable vector.

NOTES: This subroutine in not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSXLIM

PURPOSE: This subroutine tests the current design variable vector against its upper and lower bounds, and adjusts the design variables so that they do not violate those bounds. This adjustment is primarily to compensate for numerical errors that might result in a very slight violation of a bound.

USAGE: CALL KSXLIM (X, XLB, XUB, NDV, NSIDE)

INPUT PARAMETERS:

X -- Current scaled design variable vector.

XLB -- Scaled lower bounds values for design variables.

XUB -- Scaled upper bounds values for design variables.

NDV -- Number of design variables.

NSIDE -- Flag indicating whether side constraints are to be considered (see KSINIT).

OUTPUT PARAMETERS:

X -- Scaled design variable vector adjusted for bounds violations.

NOTES: This subroutine in not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KS

PURPOSE: This subroutine computes the Kreisselmeier-Steinhauser function value given a vector of input values. The algorithm used here offsets the input values by the maximum input value to eliminate numerical overflow problems.

USAGE: CALL KS (F, G, NG, RHO)

INPUT PARAMETERS:

G -- Vector of input values used to compute the KS function.

NG -- The number of input values in the G vector.

RHO -- The multiplier used in the KS function.

OUTPUT PARAMETERS:

F -- The value of the KS function.

NOTES: This subroutine is not called by the user, but since it is a general purpose subroutine for evaluating the KS function it may prove useful in other applications.

SUBROUTINES CALLED: none

SUBROUTINE KSD

PURPOSE: This subroutine computes the first partial derivative of the Kreisselmeier-Steinhauser function given a vector of input values and the derivatives of those values with respect to some user specified parameter. The algorithm used here offsets the input values by the maximum input value to eliminate numerical overflow problems.

USAGE: CALL KSD (DF, G, DGDX, NG, RHO)

INPUT PARAMETERS:

G -- Vector of input values used to compute the KS function.

DGDX -- Vector of first partial derivatives of the input values with respect to some user specified parameter.

NG -- The number of input values in the G vector.

RHO -- The multiplier used in the KS function.

OUTPUT PARAMETERS:

DF -- The value of the first partial derivative of the KS function with respect to some user specified parameter.

NOTES: This subroutine is not called by the user, but since it is a general purpose subroutine for evaluating the first partial derivative of the KS function it may prove useful in other applications.

SUBROUTINES CALLED: none

SUBROUTINE KSFUN

PURPOSE: This subroutine computes the Kreisselmeier-Steinhauser function consisting of the objective functions and constraints of the optimization problem. This is the subroutine that computes the unconstrained function which is minimized to solve the optimization problem.

USAGE: CALL KSFUN (FUN, OBJ, G, RHO, FSCALE, OFFSET, NCON, NOBJ, TEMP)

INPUT PARAMETERS:

OBJ -- Current vector of objective function values.

G -- Current vector of constraint values.

RHO -- The multiplier used in the KS function.

FSCALE -- Vector of objective function scale values.

OFFSET -- Vector of objective function offset values.

NCON -- Number of constraints.

NOBJ -- Number of objective functions.

TEMP -- Temporary array dimensioned at least NOBJ + NCON in the calling program.

OUTPUT PARAMETERS:

FUN -- The value of the KS function based on the objective functions and constraints.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: KS

SUBROUTINE KSDFUN

PURPOSE: This subroutine computes the first partial derivative of the Kreisselmeier-Steinhauser function which consists of the objective functions and constraints of the optimization problem. This is the subroutine that computes the gradients of the unconstrained function which is minimized to solve the optimization problem.

USAGE: CALL KSDFUN (DFUN, OBJ, FSCALE, OFFSET, DF, G, DG, RHO, NDV, NCON, NOBJ, TEMP1, TEMP2)

INPUT PARAMETERS:

OBJ -- Current vector of objective function values.

FSCALE -- Vector of objective function scale values.

OFFSET -- Vector of objective function offset values.

DF -- Current matrix of objective function gradients.

G -- Current vector of constraint values.

DG -- Current matrix of constraint gradients.

RHO -- The multiplier used in the KS function.

NDV -- Number of design variables.

NCON -- Number of constraints.

NOBJ -- Number of objective functions.

TEMP1 -- Temporary array dimensioned at least NOBJ + NCON in the calling program.

TEMP2 -- Temporary array dimensioned at least NOBJ + NCON in the calling program.

OUTPUT PARAMETERS:

DFUN -- Vector of the first derivatives of the KS function based on the gradients of the objective functions and constraints.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: KSD

SUBROUTINE KSGRAD

PURPOSE: This subroutine computes the gradients of the objective functions and constraints by finite differences, if that option was selected in subroutine KSINIT. Only a forward difference scheme is used, but tests are made to determine if the finite difference step would violate an upper bound. In this case, backward differencing is used for the particular design variable.

USAGE: CALL KSGRAD (INEXT, X, X0, XLB, XUB, G, G0, OBJ, OBJ0, DF, DG, SCALE, DELX, NDV, NCON, NOBJ, NSIDE, FDELT, FDMIN)

INPUT PARAMETERS:

INEXT -- Flag indicating which design variable is currently being perturbed for finite differencing. If zero, indicates the initial call to this subroutine in which initial values of the design variables, constraints, and objective functions are saved.

X -- Current scaled design variable vector.

X0 -- Vector holding a copy of the current design variables.

XLB -- Scaled lower bounds values for design variables.

XUB -- Scaled upper bounds values for design variables.

G -- Current vector of constraint values.

G0 -- Vector holding a copy of the current constraints.

OBJ -- Current vector of objective functions.

OBJ0 -- Vector holding a copy of the current objective functions.

DF -- Current matrix of objective function gradients.

DG -- Current matrix of constraint gradients.

SCALE -- Current design variable scale values.

NDV -- Number of design variables.

NCON -- Number of constraints.

NOBJ -- Number of objective functions.

NSIDE -- Flag indicating whether side constraints are to be considered (see KSINIT).

FDELT -- Step size used to compute finite differences.

FDMIN -- Minimum step allowed for finite differences.

OUTPUT PARAMETERS:

INEXT	-- Flag indicating which design variable has been perturbed for finite difference calculations. If zero, then the finite differencing is complete.
X	-- Vector of scaled design variables with one variable perturbed for finite difference calculations. If INEXT is zero, then this is the current scaled design variable vector.
DELX	-- The design variable step size being used for finite differences on the current design variable.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSANDO

PURPOSE: This subroutine computes the scale factor and offset for each objective function, which are used when assembling the KS function for the unconstrained minimization problem.

USAGE: CALL KSANDO (OBJ, G, FSCALE, OFFSET, NCON, NOBJ)

INPUT PARAMETERS:

OBJ -- Current vector of objective functions.

G -- Current vector of constraints.

NCON -- Number of constraints.

NOBJ -- Number of objective functions.

OUTPUT PARAMETERS:

FSCALE -- Vector of objective function scale values.

OFFSET -- Vector of objective function offset values.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSONED

PURPOSE: This subroutine performs a one-dimensional minimization on the unconstrained KS function given search direction. Termination of the minimization occurs when the minimum of a quadratic curve placed through three points is found to within a preset tolerance.

USAGE: CALL KSONED (INEXT, JTRY, X, X0, XLB, XUB, FUN, FUN0, S, SLOPE, ALPHA, ALPMAX, NDV, A1, A2, A3, F1, F2, F3, FTEST, NSIDE, LIMIT, NUNIT, IPRNT3, SCALE, TEMP, ISDFLG)

INPUT PARAMETERS:

INEXT -- Flag indicating which part of this subroutine is to be executed at this time. This is used to control what phase of the one-dimensional search is currently in use.

X -- Current scaled design variable vector.

X0 -- Scaled design variables at start of the current iteration.

XLB -- Scaled lower bounds values for design variables.

XUB -- Scaled upper bounds values for design variables.

FUN -- Current value of the unconstrained KS function.

FUN0 -- Value of KS function at start of the current iteration.

S -- Vector containing the search direction to be used for the one-dimensional minimization of the unconstrained KS function.

SLOPE -- Initial slope of the search direction vector.

NDV -- Number of design variables.

NSIDE -- Flag indicating whether side constraints are to be considered.

NUNIT -- FORTRAN file unit number to be used for printed output.

IPRNT3 -- Hundreds digit of the IPRNT flag from subroutine KSINIT.

SCALE -- Current design variable scale values.

TEMP -- Temporary array dimensioned at least NDV in the calling program.

OUTPUT PARAMETERS:

INEXT -- Flag indicating which part of this subroutine is to be executed the next time it is entered. If zero, the one-dimensional minimization is complete.

JTRY -- A counter to track how many times a quadratic interpolation has been performed during the one-dimensional minimization. If this counter reaches 15, the minimization is terminated for this iteration.

X -- The scaled design variable vector which has been stepped in the search direction. This is the next trial design point.

ALPHA -- The distance in the one-dimensional search direction that the design variables have been moved.

ALPMAX -- The maximum distance that can be moved in the search direction before encountering a side constraint.

A1 -- The first abscissa of the three point quadratic curve which is interpolated for a minimum. This variable holds the distance alpha for point 1.

A2 -- The second abscissa of the three point quadratic curve which is interpolated for a minimum. This variable holds the distance alpha for point 2.

A3 -- The third abscissa of the three point quadratic curve which is interpolated for a minimum. This variable holds the distance alpha for point 3.

F1 -- The first ordinate of the three point quadratic curve which is interpolated for a minimum. This variable holds the unconstrained KS function value corresponding to point A1.

F2 -- The second ordinate of the three point quadratic curve which is interpolated for a minimum. This variable holds the unconstrained KS function value corresponding to point A2.

F3 -- The third ordinate of the three point quadratic curve which is interpolated for a minimum. This variable holds the unconstrained KS function value corresponding to point A3.

FTEST -- The value of the unconstrained function interpolated from a three point quadratic curve.

LIMIT -- A flag indicating when a new side constraint is encountered.
0 = No new side constraints encountered
1 = One or more new side constraints encountered

ISDFLG -- A flag indicating whether or not to restart the DFP algorithm with the steepest descent method. A value of zero means to restart the DFP.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: KSQMIN
KSQUAD
KSUNSC

SUBROUTINE KSDFP

PURPOSE: This subroutine computes the one-dimensional search direction for the unconstrained minimization problem using a modified version of the Davidon-Fletcher-Powell (DFP) search algorithm. The modified method keeps track of side constraints and restarts the DFP algorithm with a steepest descent method whenever a side constraint is encountered. As long as the design stays on a side constraint, the DFP algorithm is used as usual but the component of the search direction corresponding the design variable which is at a side constraint is forced to be zero.

USAGE: CALL KSDFP (X, ISIDE, ISACT, DFUN, NDV, S, SLOPE, Y, P, H, ISDFLG)

INPUT PARAMETERS:

X -- Current scaled design variable vector.

ISIDE -- Integer array containing flags indicating whether side constraints have been encountered for each design variable.

ISACT -- Integer array which is the same as the ISIDE array except that this array indicates which side constraints were active the last time the DFP algorithm was restarted with the steepest descent method.

DFUN -- Vector of the first derivatives of the KS function based on the gradients of the objective functions and constraints.

NDV -- Number of design variables.

S -- The search direction vector from the previous iteration.

Y -- The vector of gradients of the unconstrained function being minimized from the previous iteration.

P -- The vector of scaled design variables from the previous iteration.

H -- The lower triangular part of the partial Hessian matrix, stored as a vector in row order.

ISDFLG -- A flag indicating whether or not to restart the DFP algorithm with the steepest descent method. A value of zero means to restart the DFP.

OUTPUT PARAMETERS:

X -- The next trial scaled design variable vector.

ISACT -- If a steepest descent restart was performed, this vector contains the currently active side constraint flags.

S	--	The current search direction vector, normalized to unit length.
SLOPE	--	The slope of the normalized search direction vector.
Y	--	A copy of the vector of gradients of the unconstrained function being minimized.
P	--	A copy of the current scaled design variable vector.
H	--	The lower triangular part of the partial Hessian matrix, stored as a vector in row order.
ISDFLG	--	A flag indicating whether or not to restart the DFP algorithm with the steepest descent method. This flag is reset to zero if a steepest descent restart was performed.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: KSHMUL
KSVPRD

SUBROUTINE KSQUAD

PURPOSE: This subroutine computes the minimum of a quadratic curve passing through three points. The middle point of the curve must be less than the two end points.

USAGE: CALL KSQUAD (A1, A2, A3, F1, F2, F3, ASTAR, FTEST)

INPUT PARAMETERS:

A1 -- The abscissa of the first point on the curve.

A2 -- The abscissa of the second point on the curve.

A3 -- The abscissa of the third point on the curve.

F1 -- The ordinate of the first point on the curve.

F2 -- The ordinate of the second point on the curve.

F3 -- The ordinate of the third point on the curve.

OUTPUT PARAMETERS:

ASTAR -- The interpolated value of the abscissa.

FTEST -- The interpolated value of the ordinate.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSQMIN

PURPOSE: This subroutine determines which three points of the four provided to use for the next quadratic interpolation.

USAGE: CALL KSQMIN (A1, A2, A3, ALPHA, F1, F2, F3, FUN)

INPUT PARAMETERS:

A1 -- The abscissa of the first point on the previously interpolated curve.

A2 -- The abscissa of the second point on the previously interpolated curve.

A3 -- The abscissa of the third point on the previously interpolated curve.

ALPHA -- The previously interpolated value of the abscissa.

F1 -- The ordinate of the first point on the previously interpolated curve.

F2 -- The ordinate of the second point on the previously interpolated curve.

F3 -- The ordinate of the third point on the previously interpolated curve.

FUN -- The previously interpolated value of the ordinate.

OUTPUT PARAMETERS:

A1 -- The new abscissa of the first point of the quadratic curve.

A2 -- The new abscissa of the second point of the quadratic curve.

A3 -- The new abscissa of the third point of the quadratic curve.

F1 -- The new ordinate of the first point of the quadratic curve.

F2 -- The new ordinate of the second point of the quadratic curve.

F3 -- The new ordinate of the third point of the quadratic curve.

NOTES: This subroutine is not called by the user.

SUBROUTINES CALLED: none

SUBROUTINE KSHMUL

PURPOSE: This subroutine multiplies a lower triangular matrix by a vector, returning a vector. The matrix is usually the Hessian matrix, and the multiplied vector is usually the vector of gradients of the unconstrained function being minimized. The resulting vector would then be the search direction vector.

USAGE: CALL KSHMUL (A, B, X, NROW)

INPUT PARAMETERS:

A -- The lower triangular part of the matrix, stored as a vector in row order. This is the Hessian matrix for this application.

B -- The vector to be multiplied by A. For this application, it is a vector of gradients of the unconstrained function being minimized.

NROW -- The number of rows and columns in the matrix. This is the same as the number of design variables for this application.

OUTPUT PARAMETERS:

X -- The vector resulting from the multiplication of matrix A by vector B. For this application, it would be the search direction vector.

NOTES: This subroutine is not called by the user, but is a generic routine which may be useful in other applications.

SUBROUTINES CALLED: none

SUBROUTINE KSVPRD

PURPOSE: This subroutine performs the dot product of two vectors.

USAGE: CALL KSVPRD (X, Y, PROD, NROW)

INPUT PARAMETERS:

X -- The first vector to by multiplied.

Y -- The second vector to be multiplied.

NROW -- The number of points in each vector.

OUTPUT PARAMETERS:

PROD -- The dot product of vectors X and Y.

NOTES: This subroutine is not called by the user, but is a generic routine which may be useful in other applications.

SUBROUTINES CALLED: none

COMMON BLOCK KSCOMM

PURPOSE: This common block holds all of the parameters required by the optimization routine. The common block is copied into the WORK array every time subroutine KSOPT returns to the user, and saving the contents of the WORK array will allow the optimization problem to be restarted at a later time from where it left off. This is particularly important when the optimizer and the analysis procedure cannot reside in the computer's memory at the same time.

PARAMETERS:

IFSCL -- Pointer to the beginning of the objective function scaling vector.

IFOFF -- Pointer to the beginning of the objective function offset vector.

ISX -- Pointer to the beginning of the scaled design variable vector.

ISX0 -- Pointer to the beginning of the scaled design variable vector saved at the beginning of the current iteration. Used to compute finite difference gradients.

ISXLB -- Pointer to the beginning of the scaled lower bounds vector.

ISXUB -- Pointer to the beginning of the scaled upper bounds vector.

ISCL -- Pointer to the beginning of the design variable scaling vector.

IG0 -- Pointer to the beginning of the constraint vector saved at the beginning of the current iteration. Used to compute finite difference gradients.

IDF -- Pointer to the beginning of the gradient vector of the unconstrained KS function being minimized.

ISLP -- Pointer to the beginning of the one-dimensional search direction.

IOBJ0 -- Pointer to the beginning of the objective function vector saved at the beginning of the current iteration. Used to compute finite difference gradients.

IY -- Pointer to the beginning of a vector where the gradients of the unconstrained KS function from the previous iteration are stored. Used to compute the search direction from the DFP algorithm.

IP -- Pointer to the beginning of a vector where the scaled design variables from the previous iteration are stored. Used to compute the search direction from the DFP algorithm.

IH -- Pointer to the beginning of the vector where the lower triangular part of the approximate Hessian matrix is stored.

ISIDE -- Pointer to the beginning of an integer vector containing flags indicating whether side constraint boundaries have been encountered for each design variable.

ISACT -- Pointer to the beginning of an integer vector containing the same flags as the ISIDE vector, except that this vector corresponds to the side constraint activity the last time the DFP algorithm was restarted with the steepest descent method.

IDOBJ -- Pointer to the beginning of the vector containing the gradients of the objective functions with respect to the scaled design variables.

IDG -- Pointer to the beginning of the vector containing the gradients of the constraints with respect to the scaled design variables.

ITMP1 -- Pointer to the beginning of a temporary scratch vector dimensioned as the maximum of 2 * NDV or NCON + NOBJ.

ITMP2 -- Pointer to the beginning of a temporary scratch vector dimensioned as the maximum of 2 * NDV or NCON + NOBJ.

INEXT -- Flag usually used to indicate which design variable is begin perturbed for finite difference gradient calculations. During a one-dimensional search, it is used to indicate which part of subroutine KSONED is to executed next.

JNEXT -- A counter indicating how many quadratic minimizations have been attempted in the current one-dimensional search.

JSEL -- Flag indicating which part of subroutine KSOPT is to be executed next.

ITCNT -- The number of the current iteration.

ICNTR -- A counter indicating how many consecutive iterations have satisfied the termination criteria for relative function change.

ICNTA -- A counter indicating how many consecutive iterations have satisfied the termination criteria for absolute function change.

ISDFLG -- Flag indicating when the DFP algorithm is to be restarted with a steepest descent method. When the flag is zero, a restart is performed.

ISDRST -- The number of iterations between steepest descent restarts.

IFNCL -- A counter indicating the total number of times the objective functions and constraints have been computed by the user.

NUNIT -- The FORTRAN unit number that all printed output goes to.

NDV -- The number of design variables.

NCON -- The number of constraints.

NOBJ -- The number of objective functions.

NSIDE -- A flag indicating whether side constraints are to be considered. Zero means no side constraints.

NSCALE -- A flag indicating which design variable scaling option is to be used (see KSINIT).

IPRNT -- A flag indicating what level of printed output to use (see KSINIT).

ITMAX -- The maximum number of iterations allowed.

IGRAD -- A flag selecting the method of gradient computation (see KSINIT).

RDF -- The same as RDFUN (see KSINIT).

ADF -- The same as ADFUN (see KSINIT).

FDL -- The same as FDELT (see KSINIT).

FDM -- The same as FDMIN (see KSINIT).

RHO -- The current value of the multiplying factor used in the KS function.

DRHO -- The amount by which RHO is incremented during the optimization.

RHOMAX -- The maximum value the factor RHO may have.

FUN0 -- The value of the unconstrained KS function at the beginning of the current iteration.

SLOPE -- The slope of the search direction obtained with the DFP algorithm.

DELX -- The increment applied to a design variable for computing finite difference gradients.

ALPHA -- The distance moved during the current one-dimensional search.

ALPMAX -- The maximum distance that may be travelled during the current one-dimensional search before a side constraint is encountered.

A1 -- Holds a value of ALPHA used for quadratic minimization.

A2 -- Holds a value of ALPHA used for quadratic minimization.

A3 -- Holds a value of ALPHA used for quadratic minimization.

F1 -- Holds a value of the unconstrained KS function used for quadratic minimization.

F2 -- Holds a value of the unconstrained KS function used for quadratic minimization.

F3 -- Holds a value of the unconstrained KS function used for quadratic minimization.

FTEST -- Holds the interpolated value of the unconstrained KS function resulting from a quadratic minimization.

LIMIT -- A flag indicating when a side constraint has been encountered during the current one-dimensional search.

APPENDIX B

GENERIC FORTRAN EXAMPLE PROGRAM OUTLINE USING KSOPT

```
      PROGRAM EXAMPLE

      DIMENSION WORK(500)

**    The WORK array holds all of KSOPT's constants and arrays for restarting
**    purposes.  The user should never alter this array.

      DIMENSION X(5), OBJ(3), G(12)

**    X is the design variable array.
**    OBJ is the objective function array.
**    G is the constraint array.

      DIMENSION XLB(5), XUB(5), SCALE(5)

**    XLB is the design variable lower bounds array.
**    XUB is the design variable upper bounds array.
**    SCALE is the design variable scaling array.

      DIMENSION DF(3,5), DG(12,5)

**    DF is the matrix of gradients of the objective functions.
**    DG is the matrix of gradients of the constraints.

      NDV    = 5
      NCON   = 12
      NOBJ   = 3
      NSIDE  = 1
      NSCALE = 4
      IPRNT  = 012
      IGRAD  = 1

**    The above parameters do not have default values.  See appendix A
**    for a complete description of each one.

      ITMAX  = 0
      ISDRST = 0
      RDFUN  = 0.0
      ADFUN  = 0.0
      FDELT  = 0.0
      FDMIN  = 0.0
      RHOMIN = 0.0
      RHOMAX = 0.0
      RHODEL = 0.0

**    The above parameters are all set to zero, and their default values
**    will be used by KSOPT.  See appendix A for a complete description
**    of each one.
```

```
      NUNIT      = 6

**    Printed output is directed to FORTRAN unit number 6

            X(1)       ... X(5)       = ...
            XLB(1)     ... XLB(5)     = ...
            XUB(1)     ... XUB(5)     = ...
            SCALE(1)   ... SCALE(5)   = ...

**    Initialize the design variable vector, the upper and lower
**    bounds, and if necessary, the design variable scaling vector.

      CALL KSINIT (X, XLB, XUB, SCALE, WORK, NDV, NCON, NOBJ, NSIDE,
                   NSCALE, IPRNT, ITMAX, IGRAD, ISDRST, RDFUN, ADFUN,
                   FDELT, FDMIN, RHOMIN, RHOMAX, RHODEL, NUNIT, IREQ)

**    KSINIT is called to initialize the KSOPT optimization procedure.

            NOMAX    = 3
            NGMAX    = 12

**    NOMAX is the first dimension of array DF above.
**    NGMAX is the first dimension of array DG above.

10    CONTINUE

      CALL KSOPT  (ISEL, X, OBJ, G, DF, DG, NOMAX, NGMAX, WORK)

**    KSOPT is called in a loop to solve the optimization problem.

            IF (ISEL .EQ. 0)    THEN
                                    STOP " The optimum has been found "
                                ENDIF

            IF (ISEL .EQ. 1)    THEN
                                    Compute the objective functions and
                                    constraints for the current design variables.
                                ENDIF

            IF (ISEL .EQ. 2)    THEN
                                    Compute the gradients of the objective
                                    functions and constraints with respect to the
                                    current design variables.
                                ENDIF

      GO TO 10

      END
```

Example Problem 1	Case 1	Case 2	Case 3
Number of Iterations	12	17	19
Number of Analyses	121	156	211
Design Variable Value	5.6518	5.7562	5.7781
Constraint 1	-2.2512	-2.2974	-2.3087
Constraint 2	-0.0635	-0.0176	-0.0061
Objective Function	0.7061	0.7030	0.7025
Initial RHO Value	20	20	50
Final RHO Value	20	80	200

Table 1. Final Optimization Results for Example Problem 1 for Three Ranges of Rho.

Example Problem 2	Weight Only	Cost Only	Weight and Cost
Number of Iterations	9	6	14
Number of Analyses	96	66	171
Outer Member Area (sq. in.)	0.4456	0.5553	0.5553
Inner Member Area (sq. in.)	0.3977	0.0010	0.0010
Yield Stress Constraint	0.0007	-0.0004	0.0068
Truss Weight (pounds)	4.19	4.43	4.43
Truss Material Cost (dollars)	17.36	1.86	1.86

Table 2. Final Optimization Results for the Three Bar Truss Example Problem.

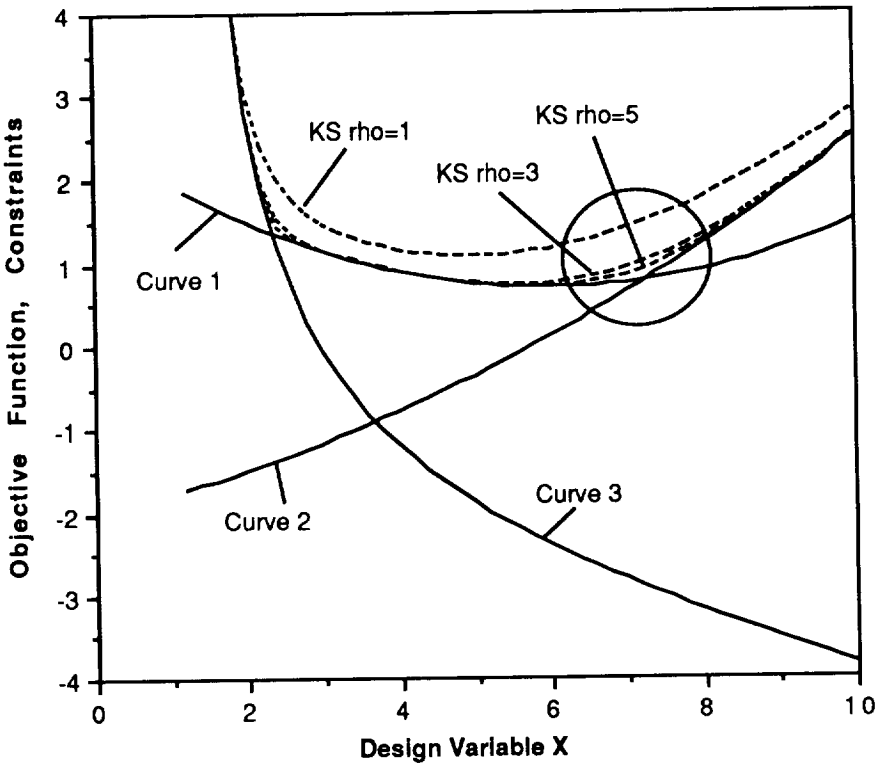


Figure 1a. Example of KS Function Characteristics for Various Rho Values.

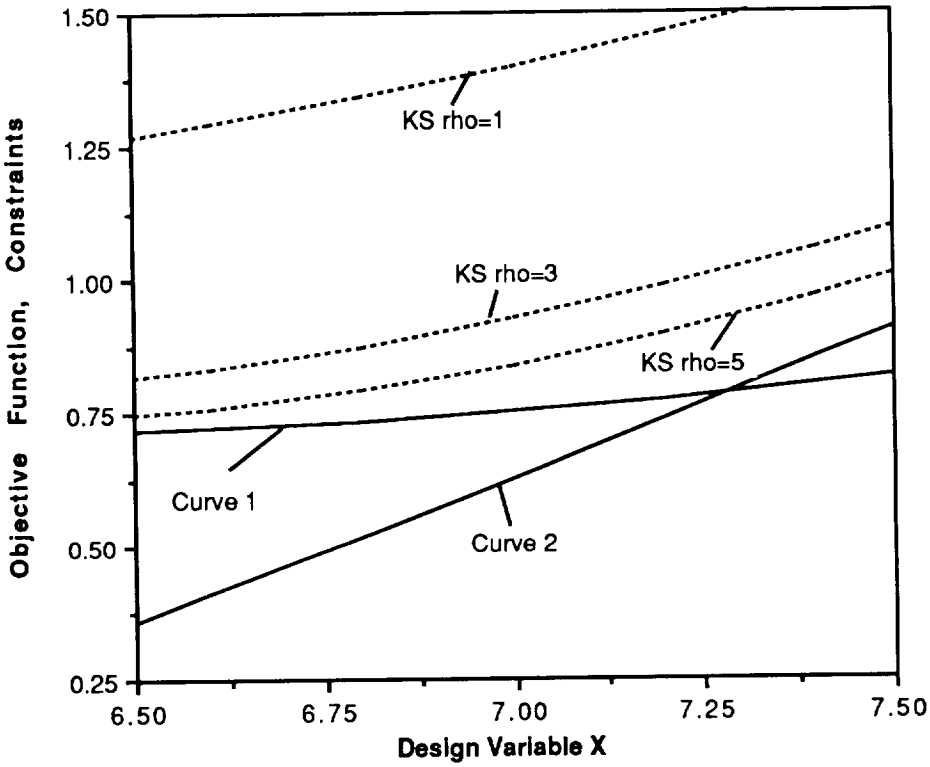


Figure 1b. Detail of KS Function for Increasing Values of Rho.

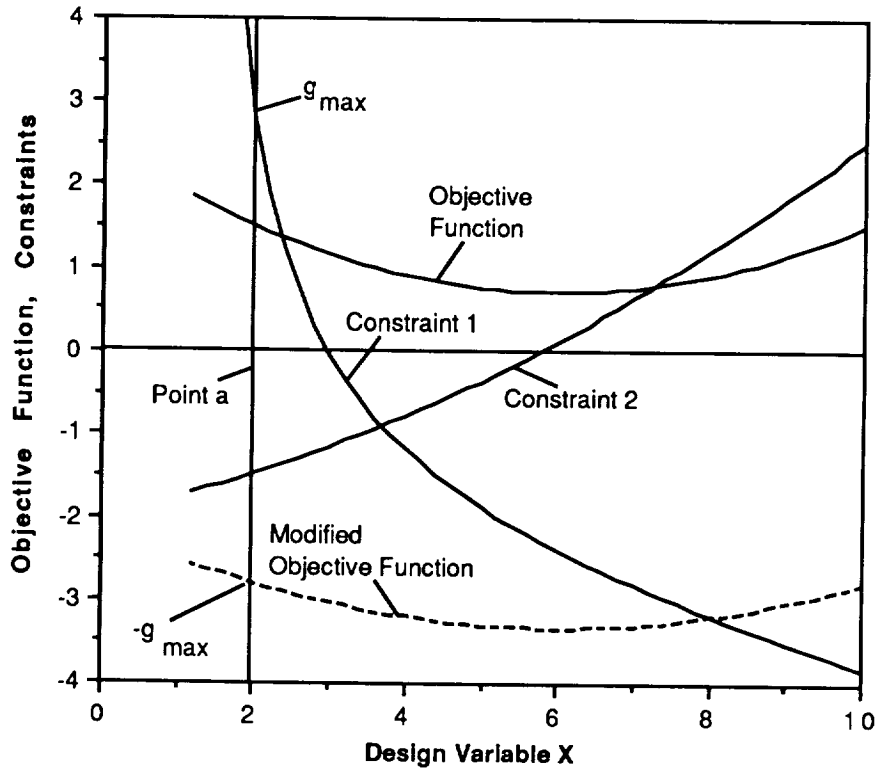


Figure 2a. Single Modified Objective Function Evaluated at Point a.

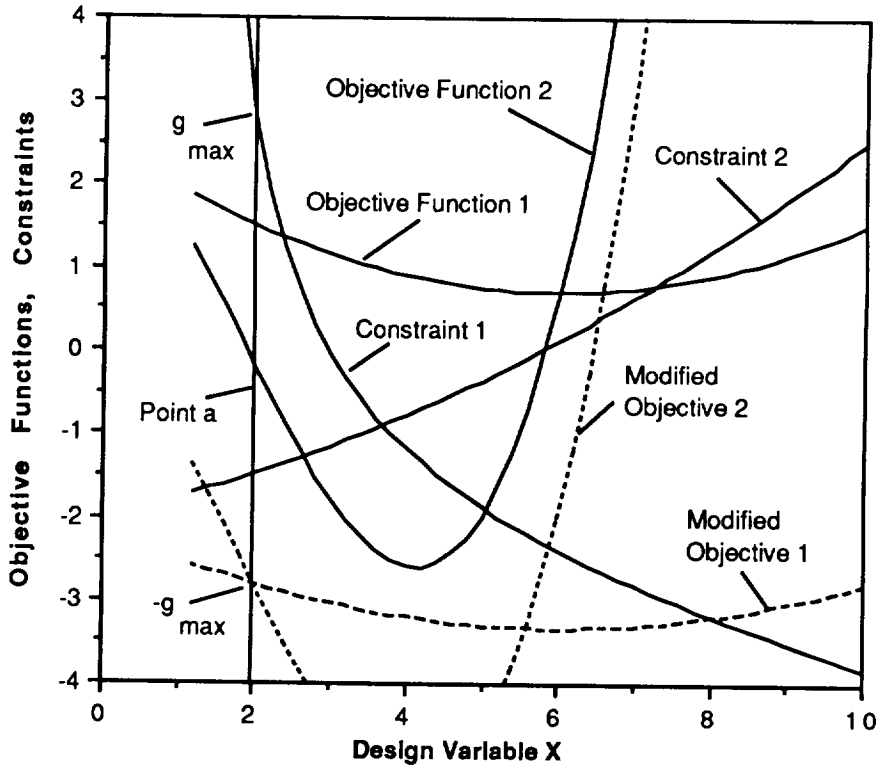


Figure 2b. Two Modified Objective Functions Evaluated at Point a.

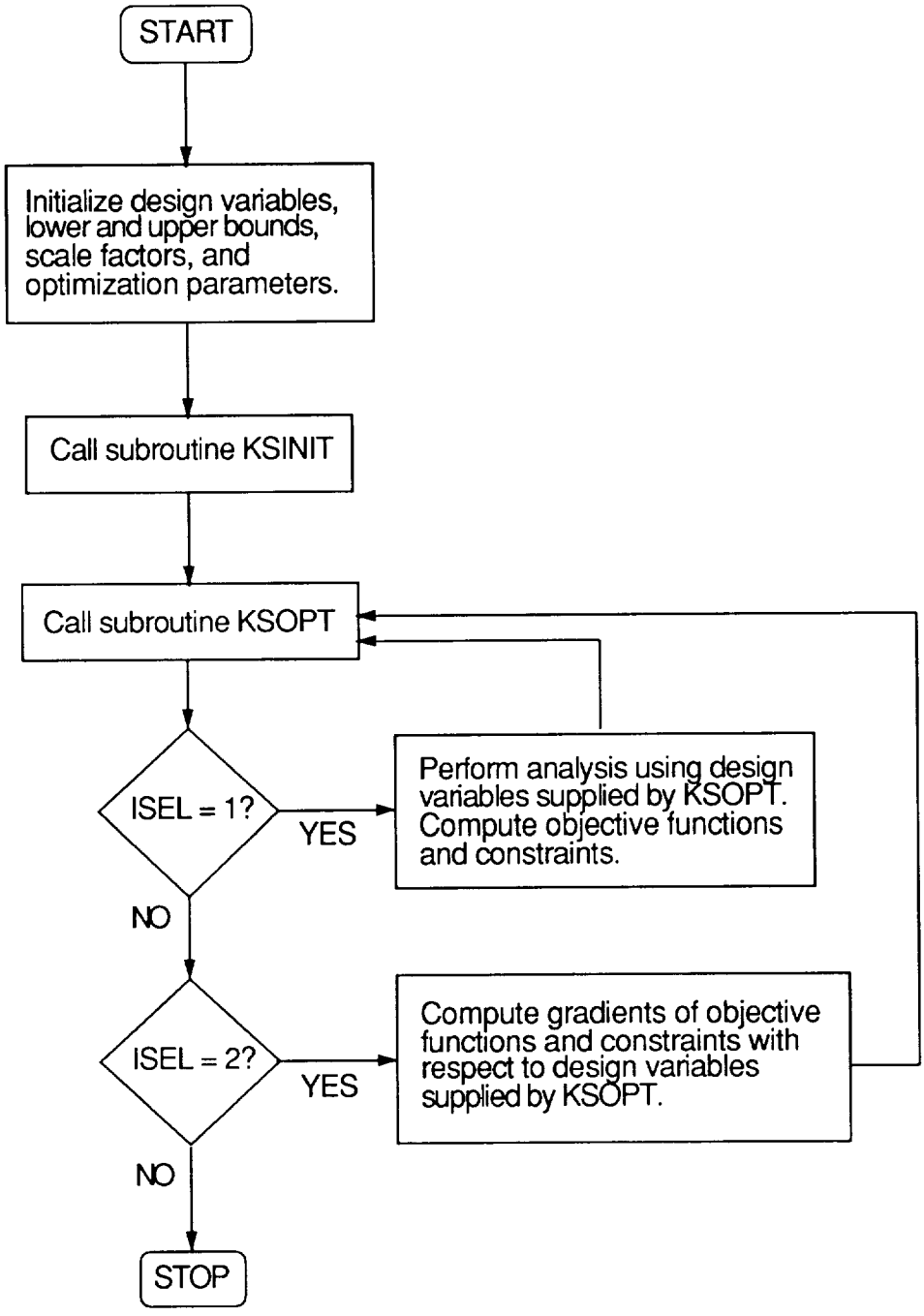


Figure 3. Block Diagram of an Optimization Procedure Using the KSOPT Program.

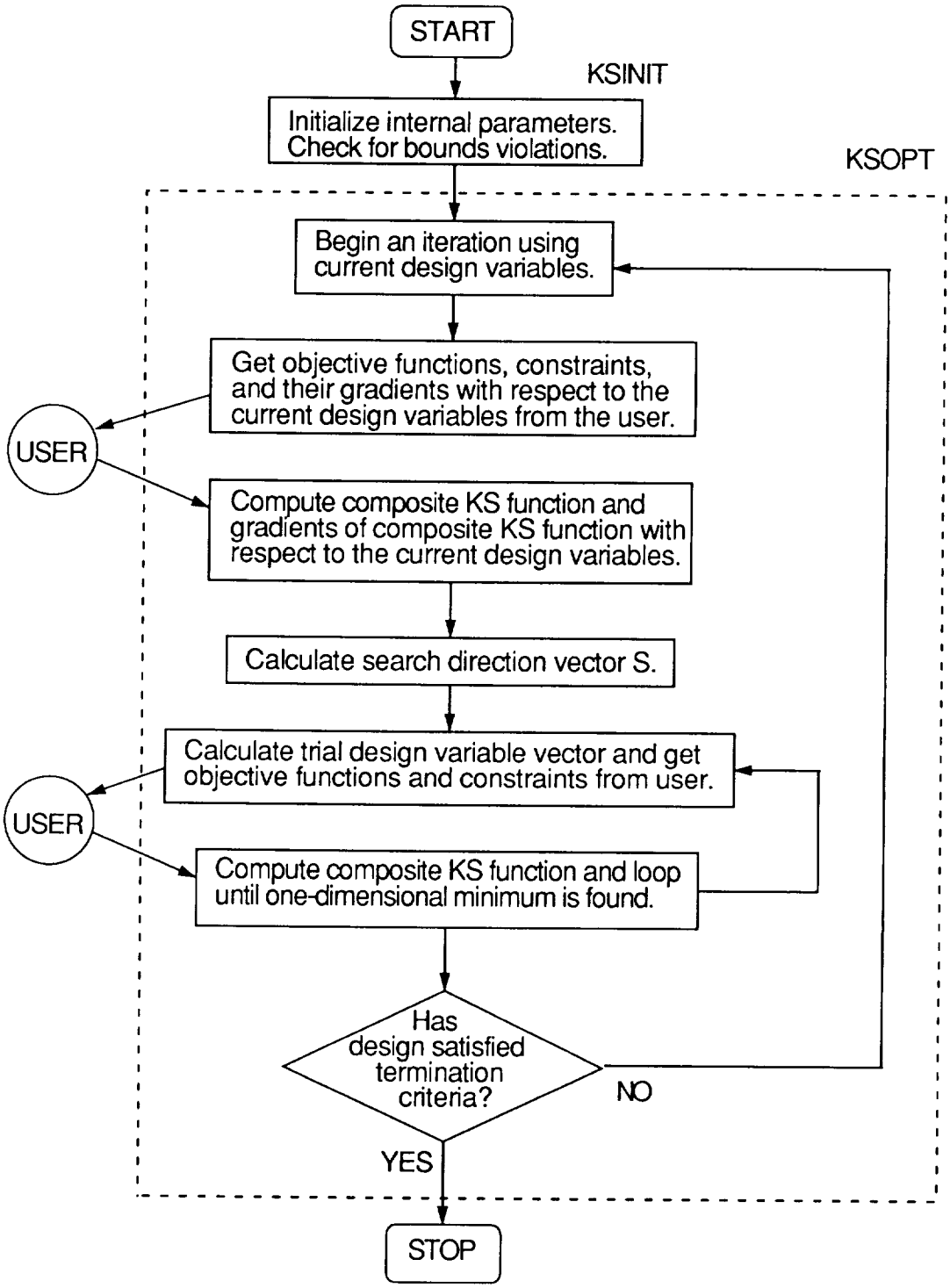


Figure 4. Flow Chart of the KSOPT Optimization Program.

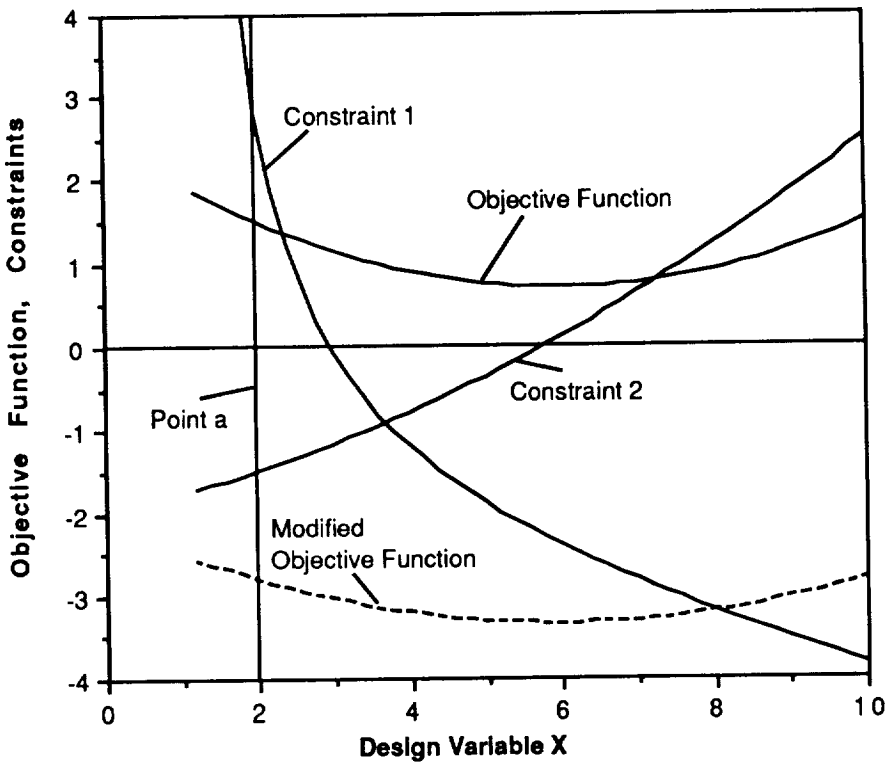


Figure 5a. One-Dimensional Design Problem Starting at Point a.

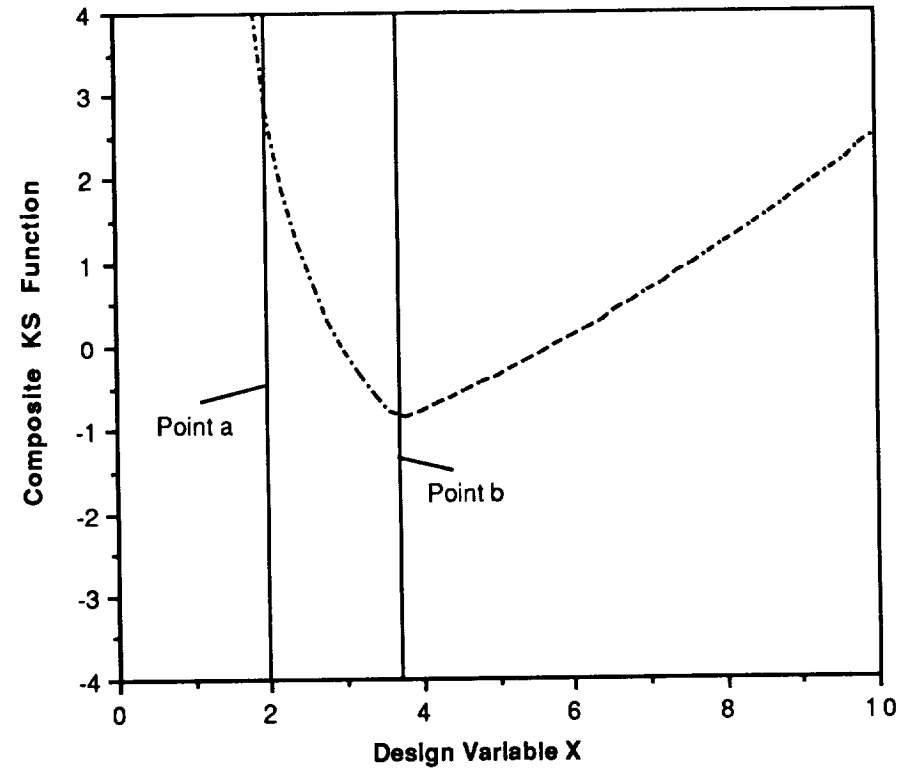


Figure 5b. Composite KS Function for Iteration 1.

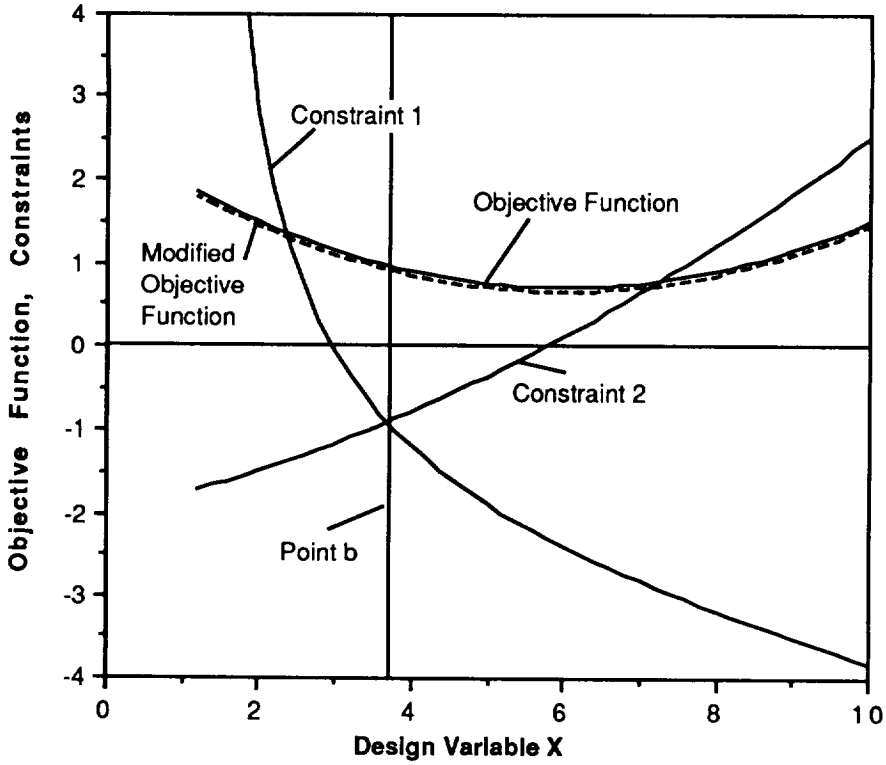


Figure 6a. One-Dimensional Design Problem Starting at Point b.

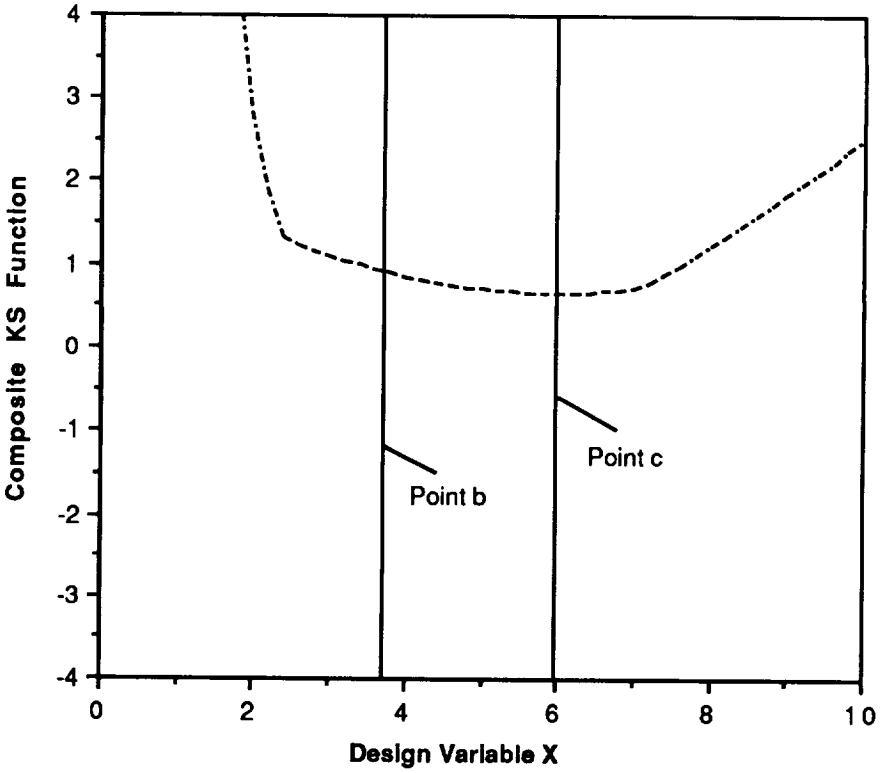


Figure 6b. Composite KS Function for Iteration 2.

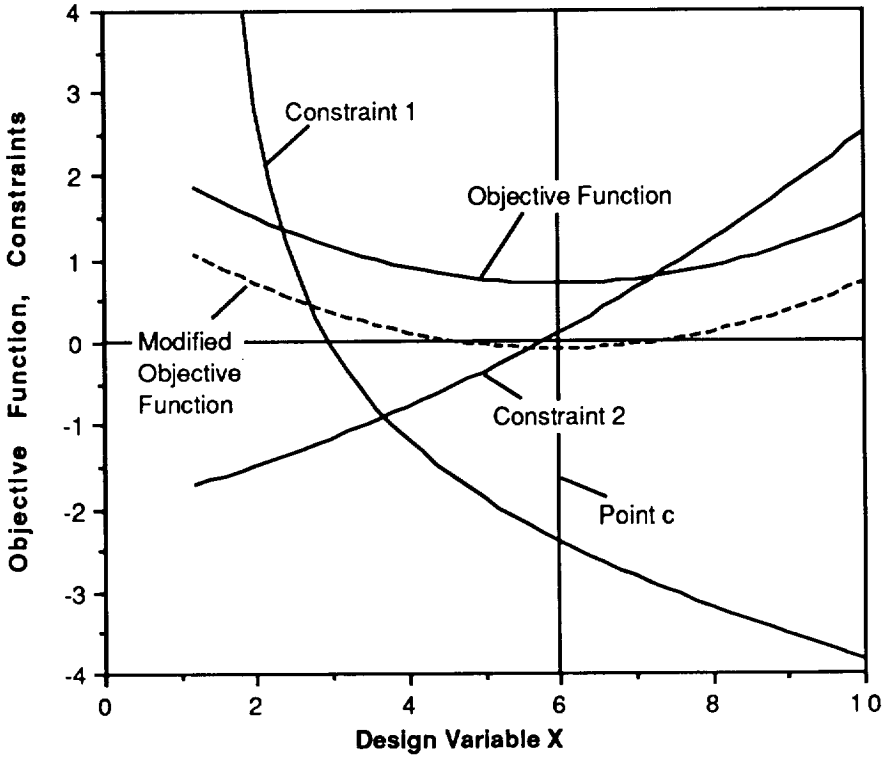


Figure 7a. One-Dimensional Design Problem Starting at Point c.

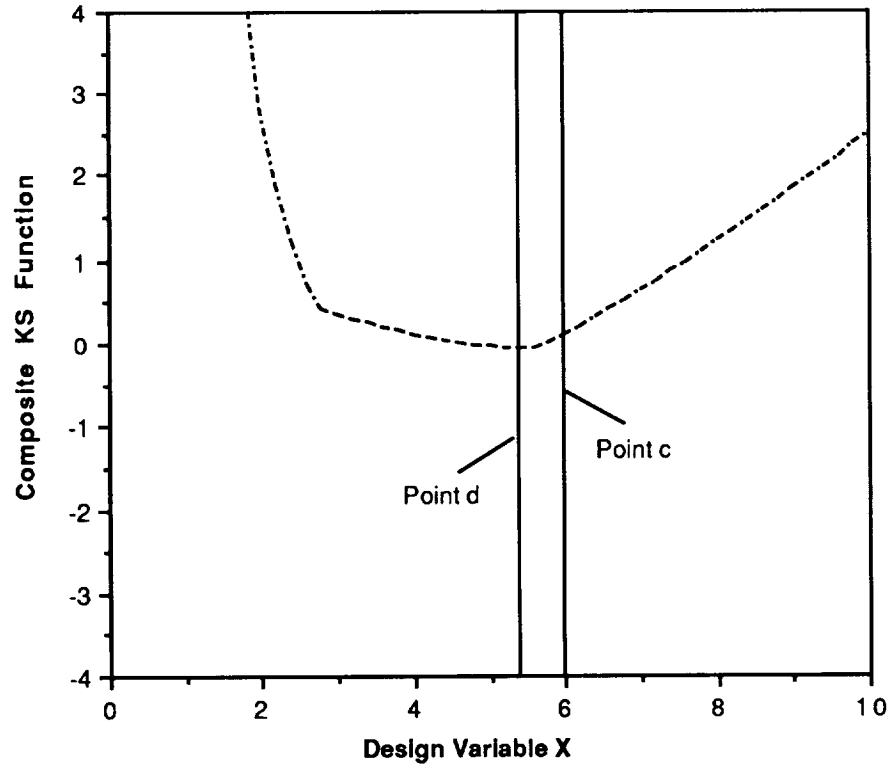


Figure 7b. Composite KS Function for Iteration 3.

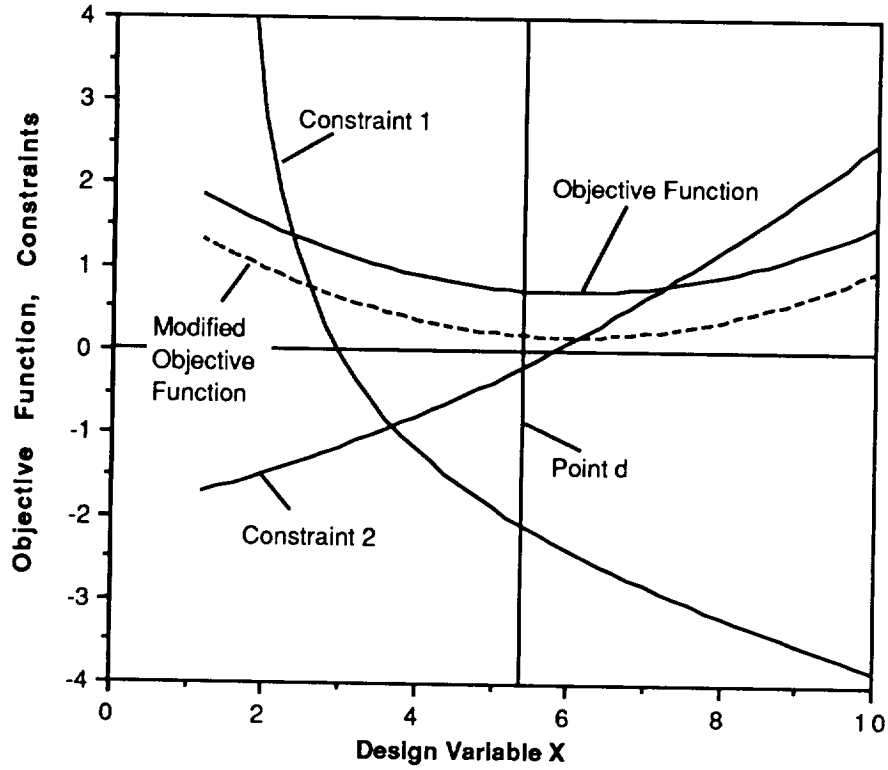


Figure 8a. One-Dimensional Design Problem Starting at Point d.

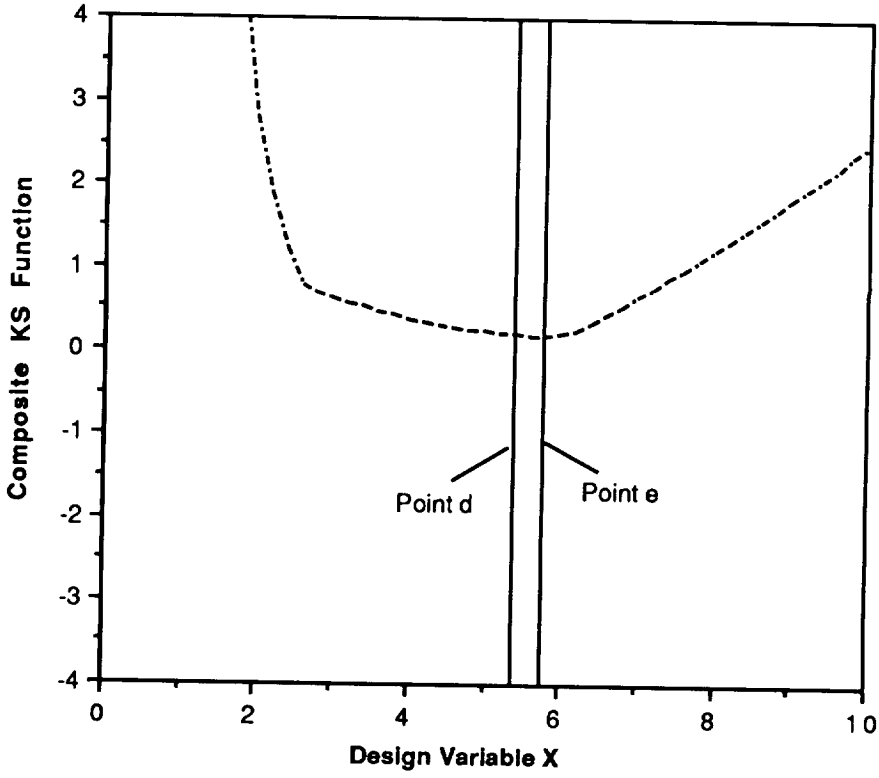


Figure 8b. Composite KS Function for Iteration 4.

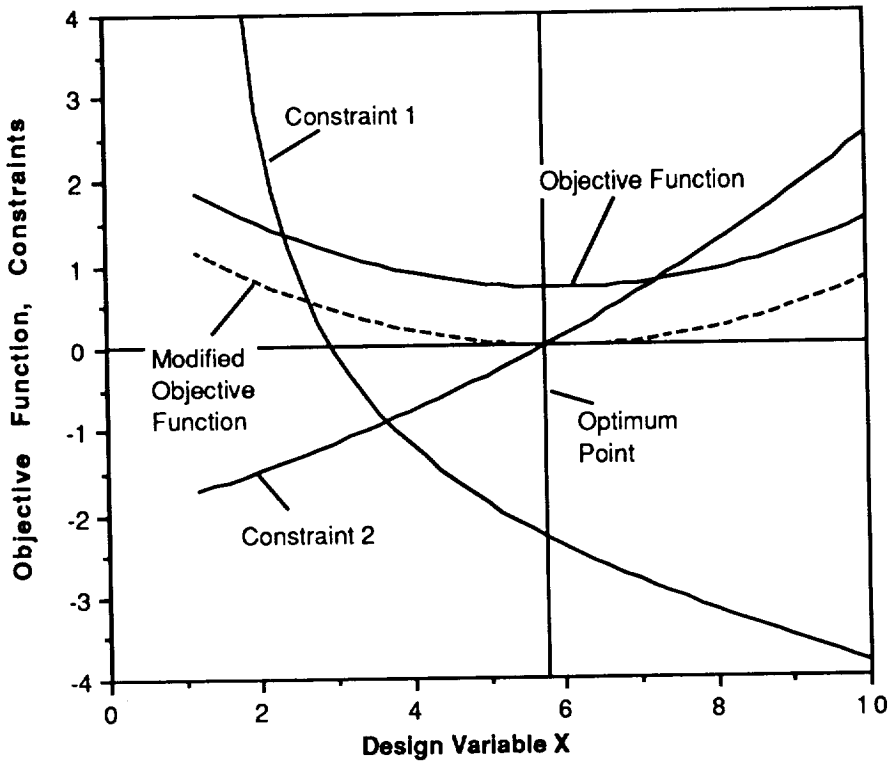


Figure 9a. One-Dimensional Design Problem at the Optimum Point.

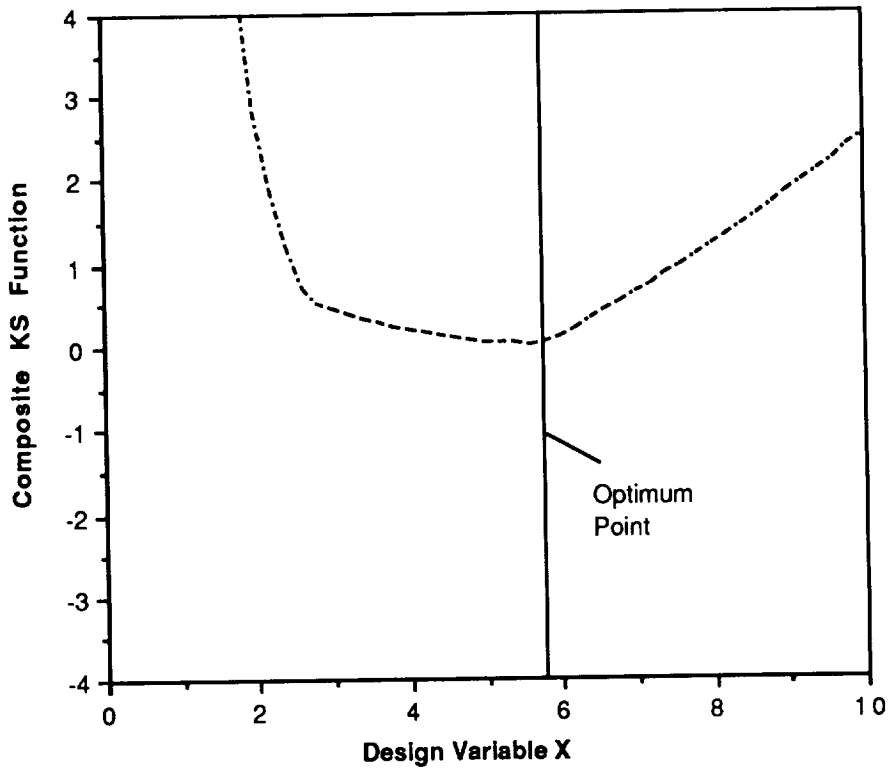
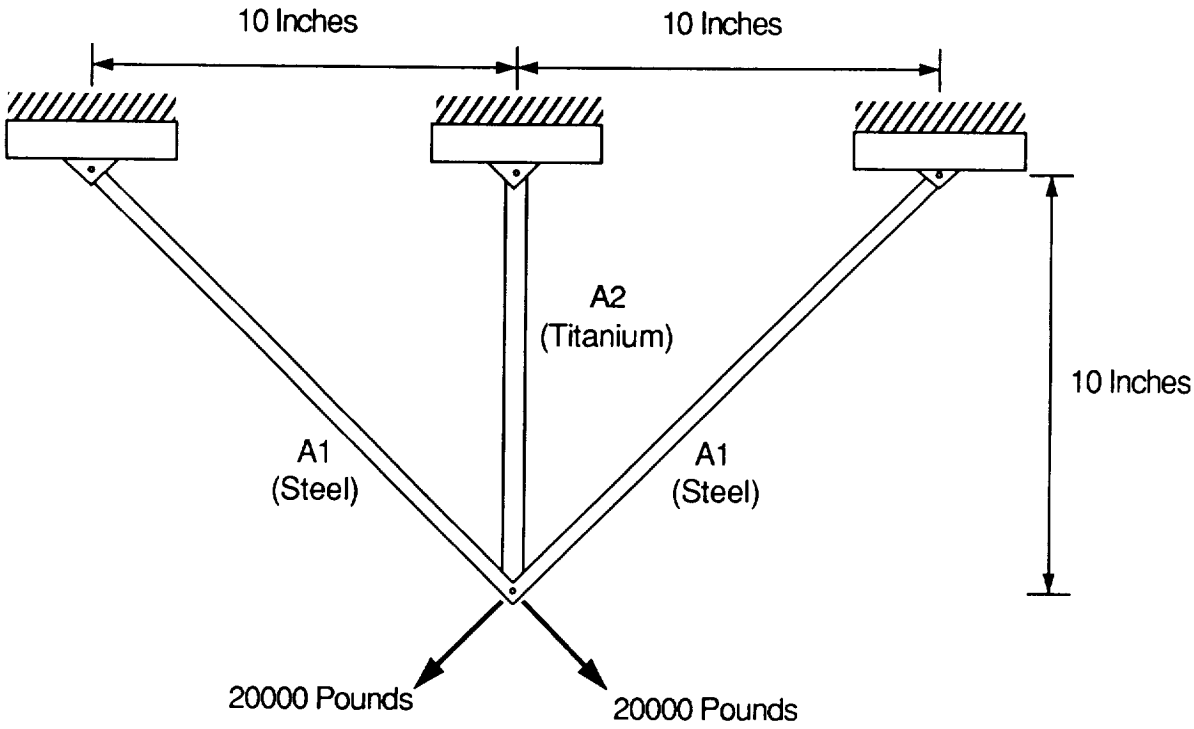


Figure 9b. Composite KS Function at the Optimum Point.



Material Properties	Steel	Titanium
Young's Modulus (psi)	30,000,000	15,500,000
Density (pounds/cu. in.)	0.282	0.160
Cost (dollars/pound)	0.41	25.00
Tensile Yield Stress (psi)	36,000	110,000
Compressive Yield Stress (psi)	27,000	82,500

Figure 10. Three Bar Truss Example Problem with Material Properties Specified.

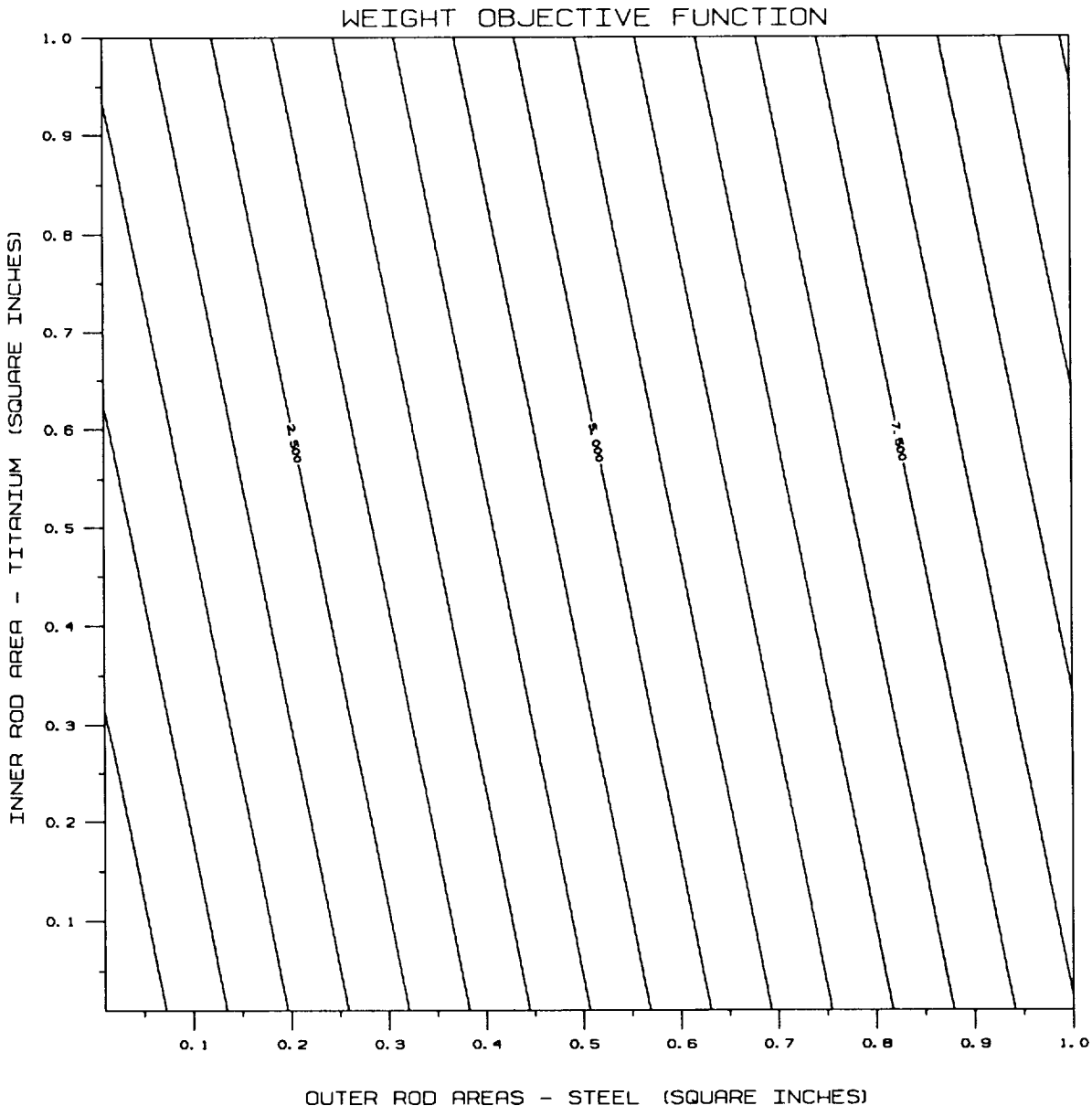


Figure 11. Contour Plot of Three Bar Truss Member Weight.

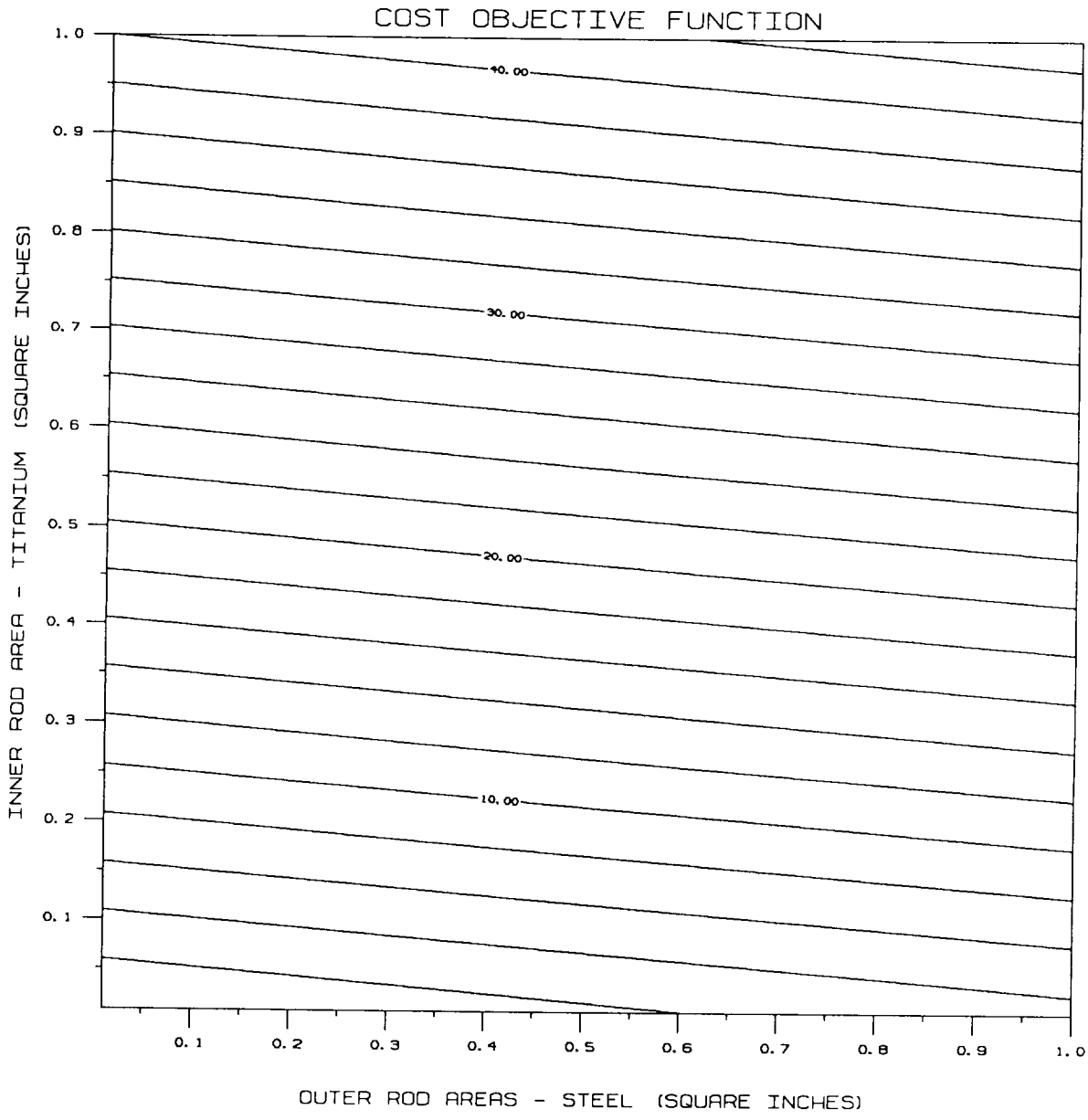


Figure 12. Contour Plot of Three Bar Truss Member Cost.

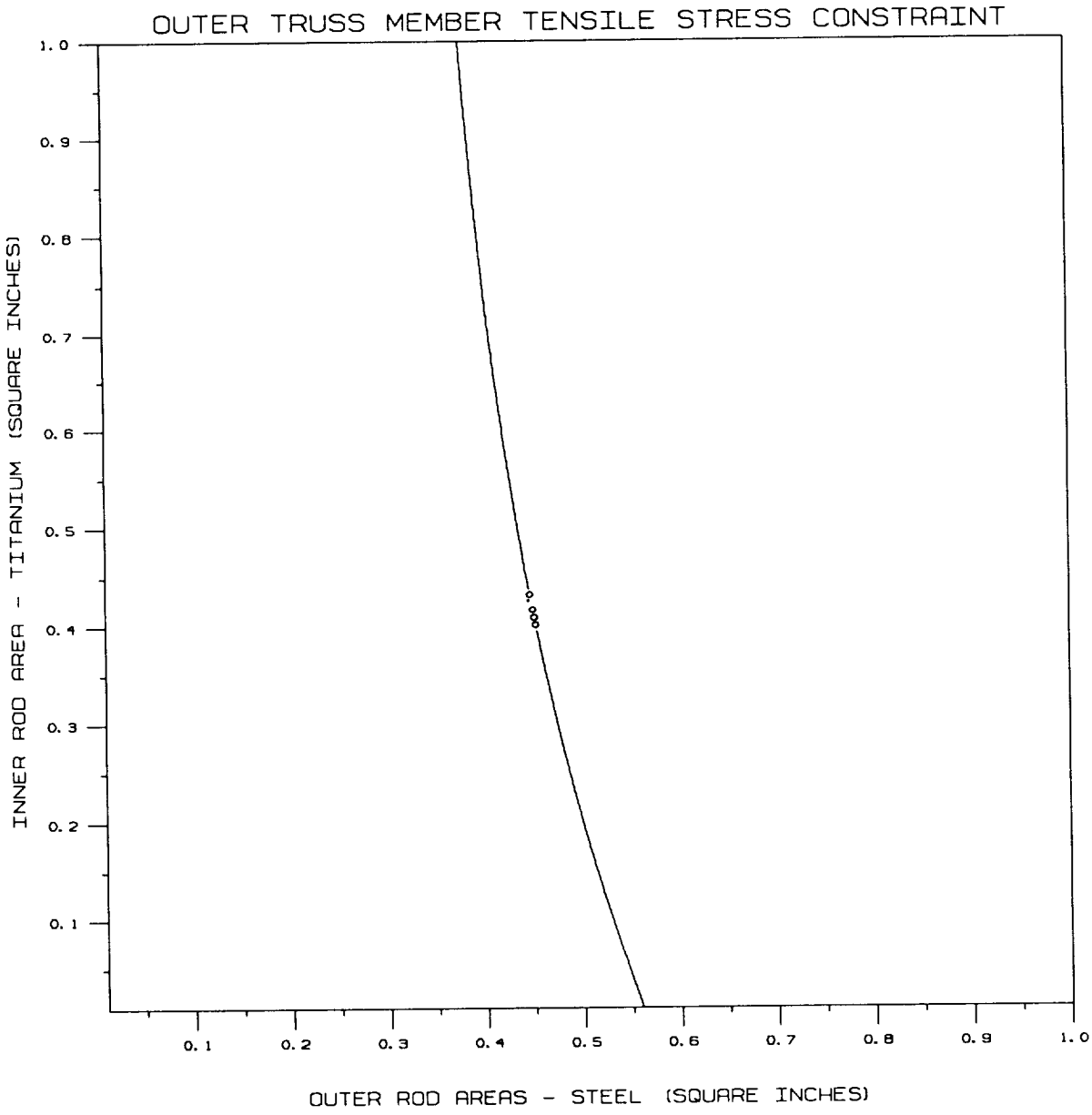


Figure 13. Contour Plot of Outer Truss Member Tensile Stress Constraint.

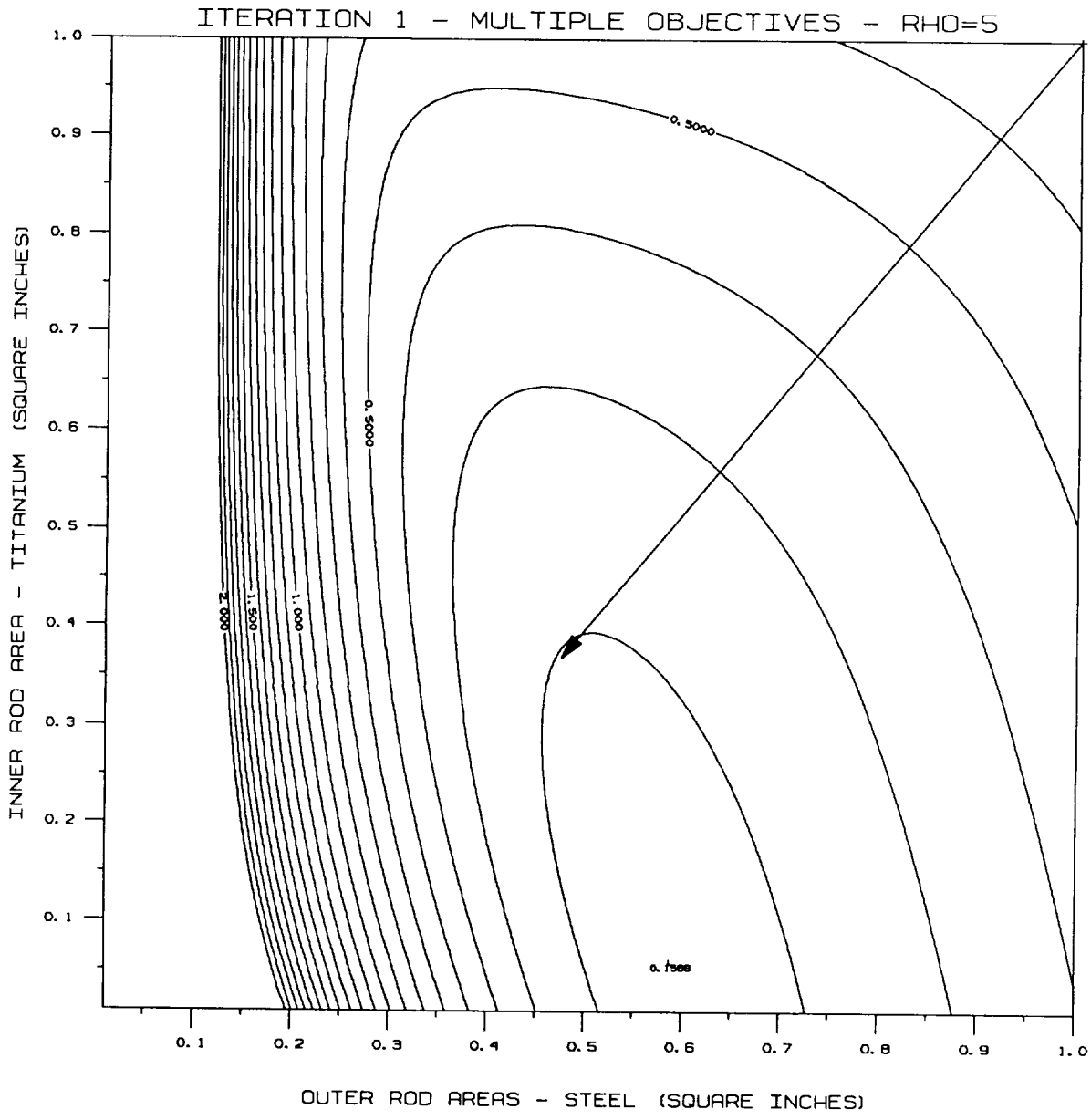


Figure 14. Contour Plot of the Three Bar Truss Design Space for Iteration 1.

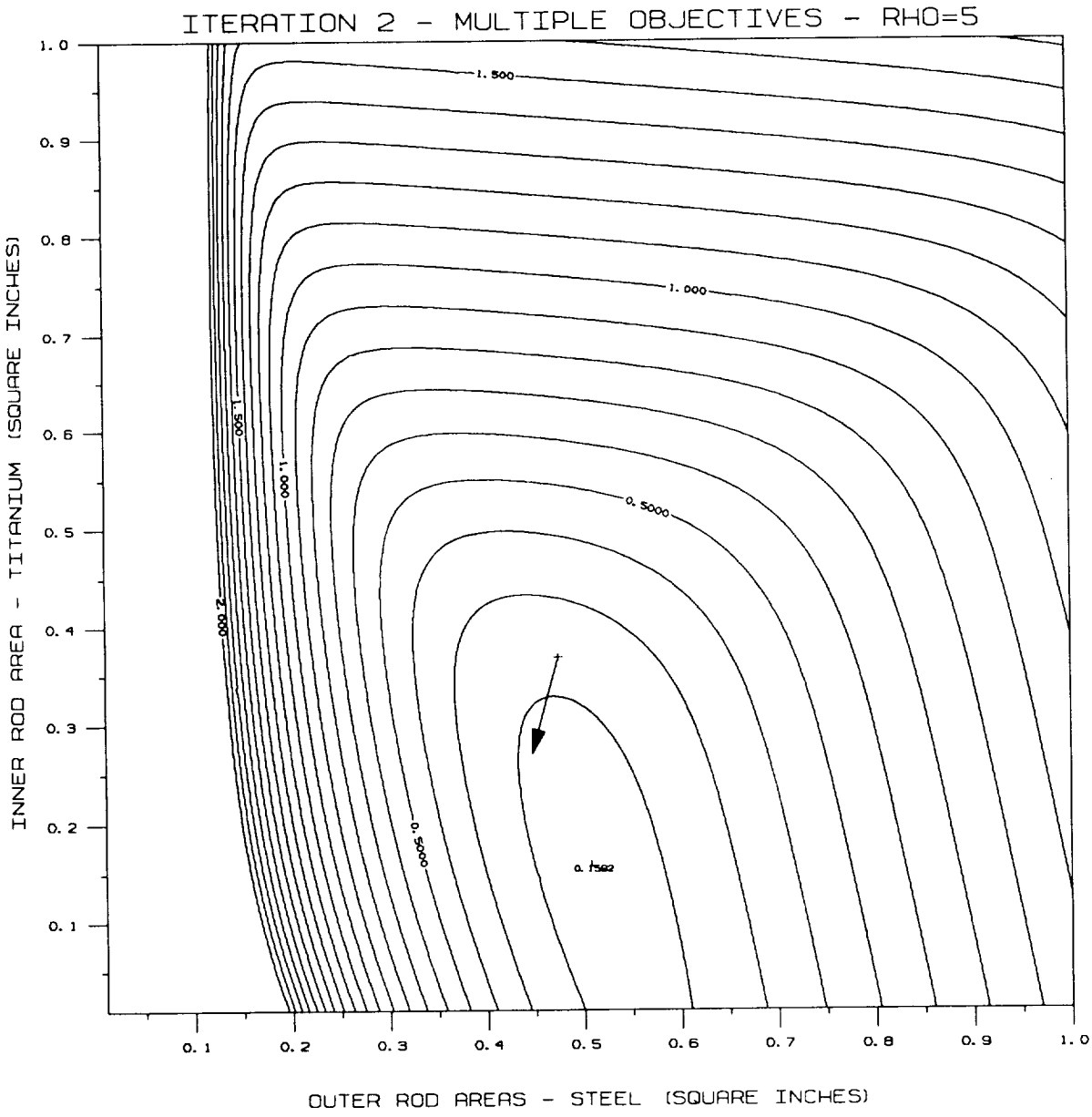


Figure 15. Contour Plot of the Three Bar Truss Design Space for Iteration 2.

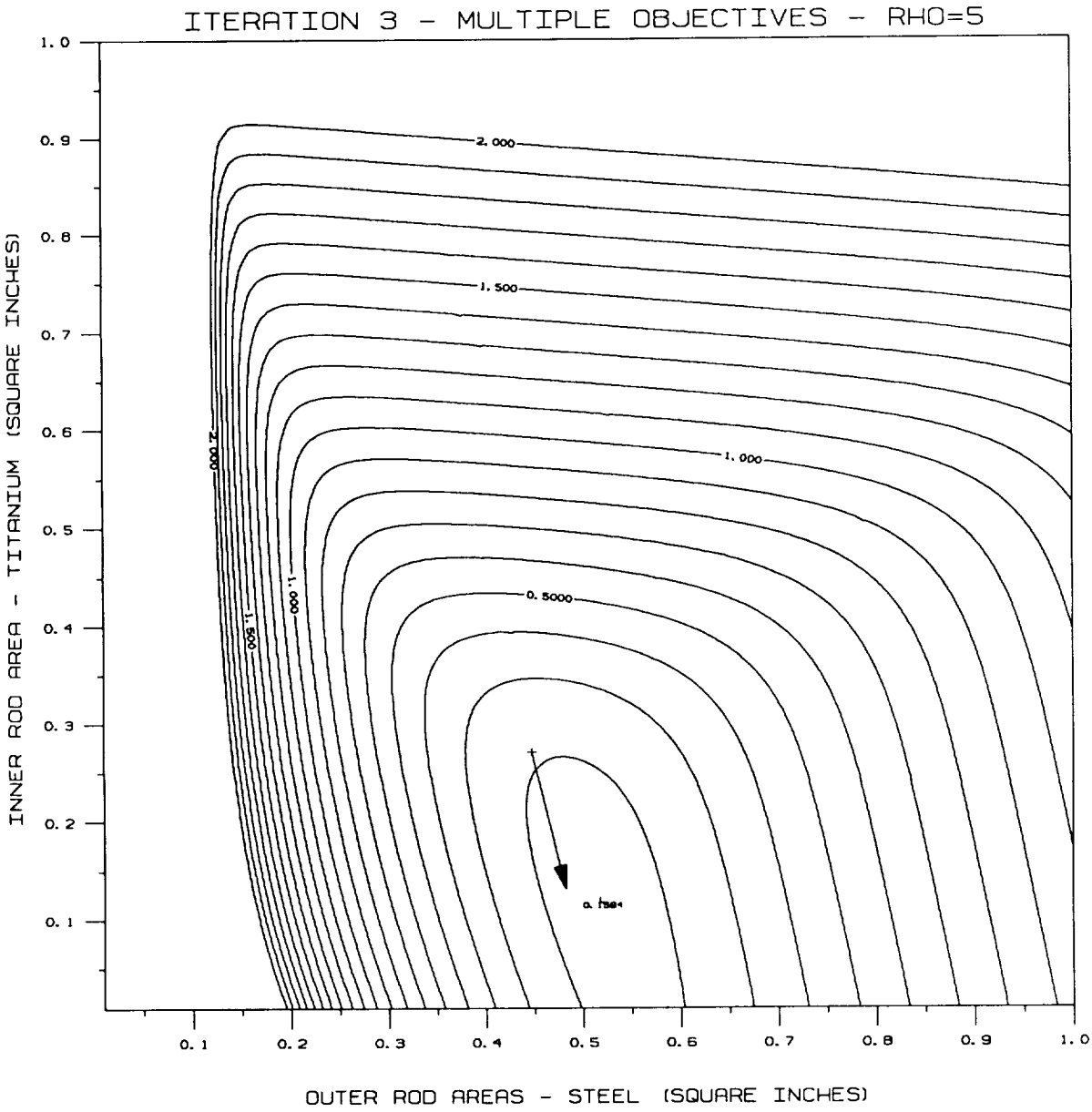


Figure 16. Contour Plot of the Three Bar Truss Design Space for Iteration 3.

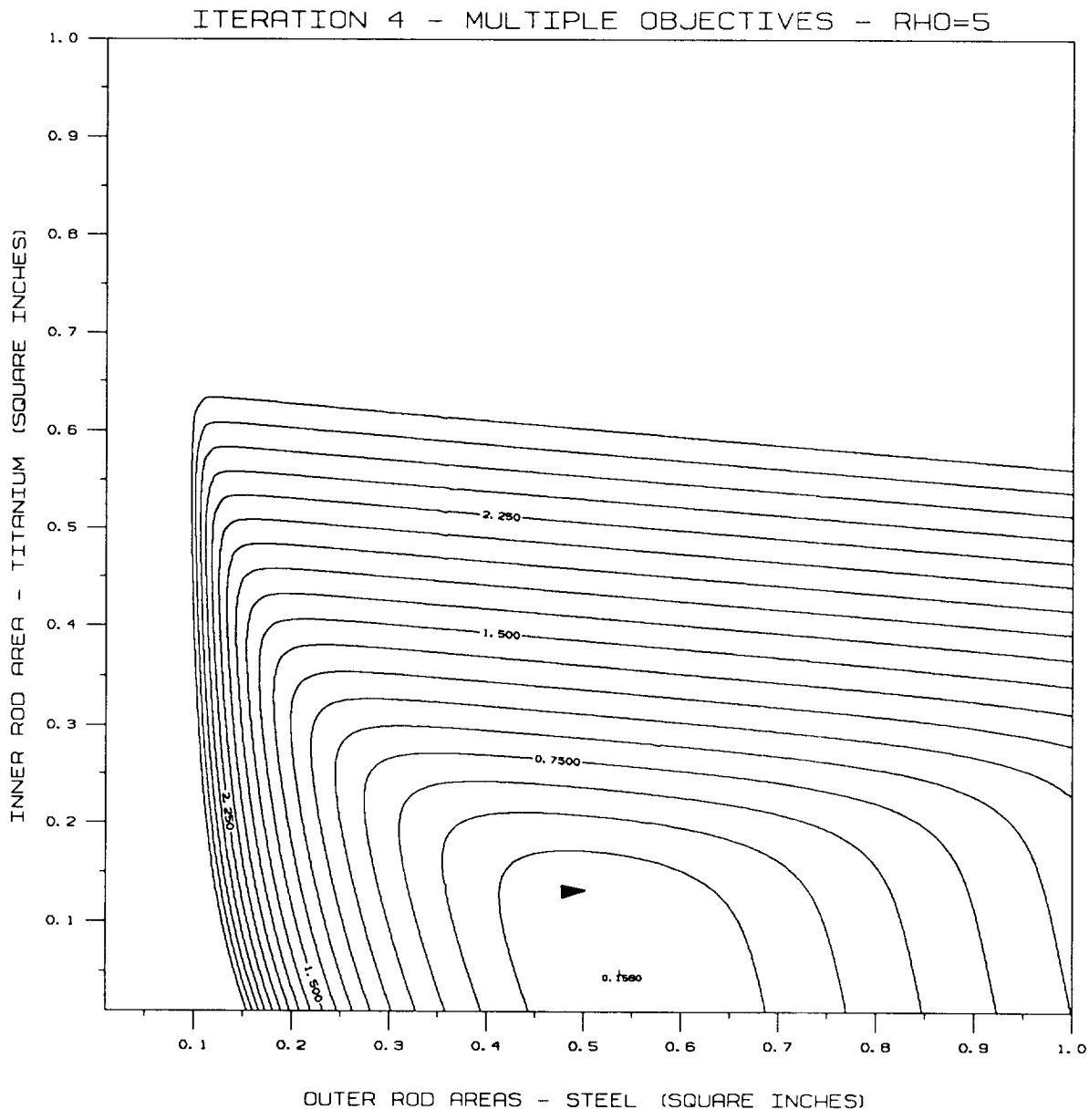


Figure 17. Contour Plot of the Three Bar Truss Design Space for Iteration 4.

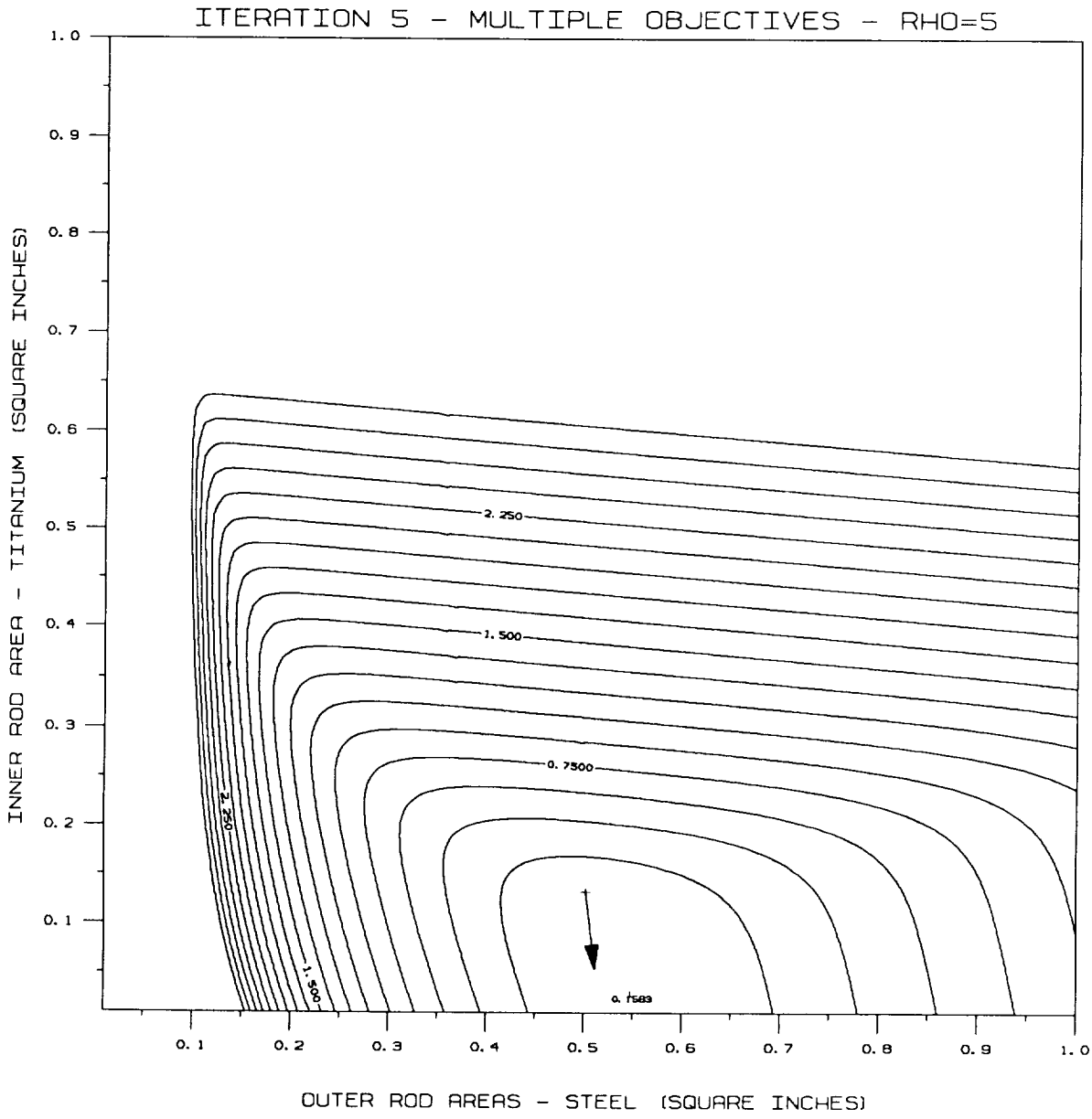


Figure 18. Contour Plot of the Three Bar Truss Design Space for Iteration 5.

REFERENCES

1. Kreisselmeier, G., and Steinhauser, R., "Systematic Control Design by Optimizing a Vector Performance Index," International Federation of Active Controls Symposium on Computer-Aided Design of Control Systems, Zurich, Switzerland, August 29-31, 1979.

2. Fiacco, A. V., McCormick, G. P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley, New York, 1968.

3. Vanderplaats, Garret N., Numerical Optimization Techniques for Engineering Design, McGraw-Hill Book Company, New York, 1984.

4. Fox, R. L., Optimization Methods for Engineering Design, Addison-Wesley, Reading, Mass., 1971.

5. Haftka, R. T., Starnes, J. H., Jr., "Applications of a Quadratic Extended Interior Penalty Function for Structural Optimization," AIAA Journal, Vol. 14, June 1976, pp. 718-724.

6. Rao, S. S., "Multiobjective Optimization in Structural Design with Uncertain Parameters and Stochastic Processes," AIAA Journal, Vol. 22, November 1984, pp. 1670-1678.

7. Davidon, W. C., "Variable Metric Methods for Minimization," Argonne National Laboratory, ANL-5990 Rev., University of Chicago, 1959.

8. Schmit, L. A.,"Structural Design by Systematic Synthesis," Proceedings of the Second National Conference on Electronic Computation, Structural Division, ASCE, Pittsburgh, PA, September 1960, pp. 105-132.

9. Sobieszczanski-Sobieski, Jaroslaw, Dovi, Augustine R., Wrenn, Gregory A.,"A New Algorithm for General Multiobjective Optimization," NASA TM-100536, March, 1988.

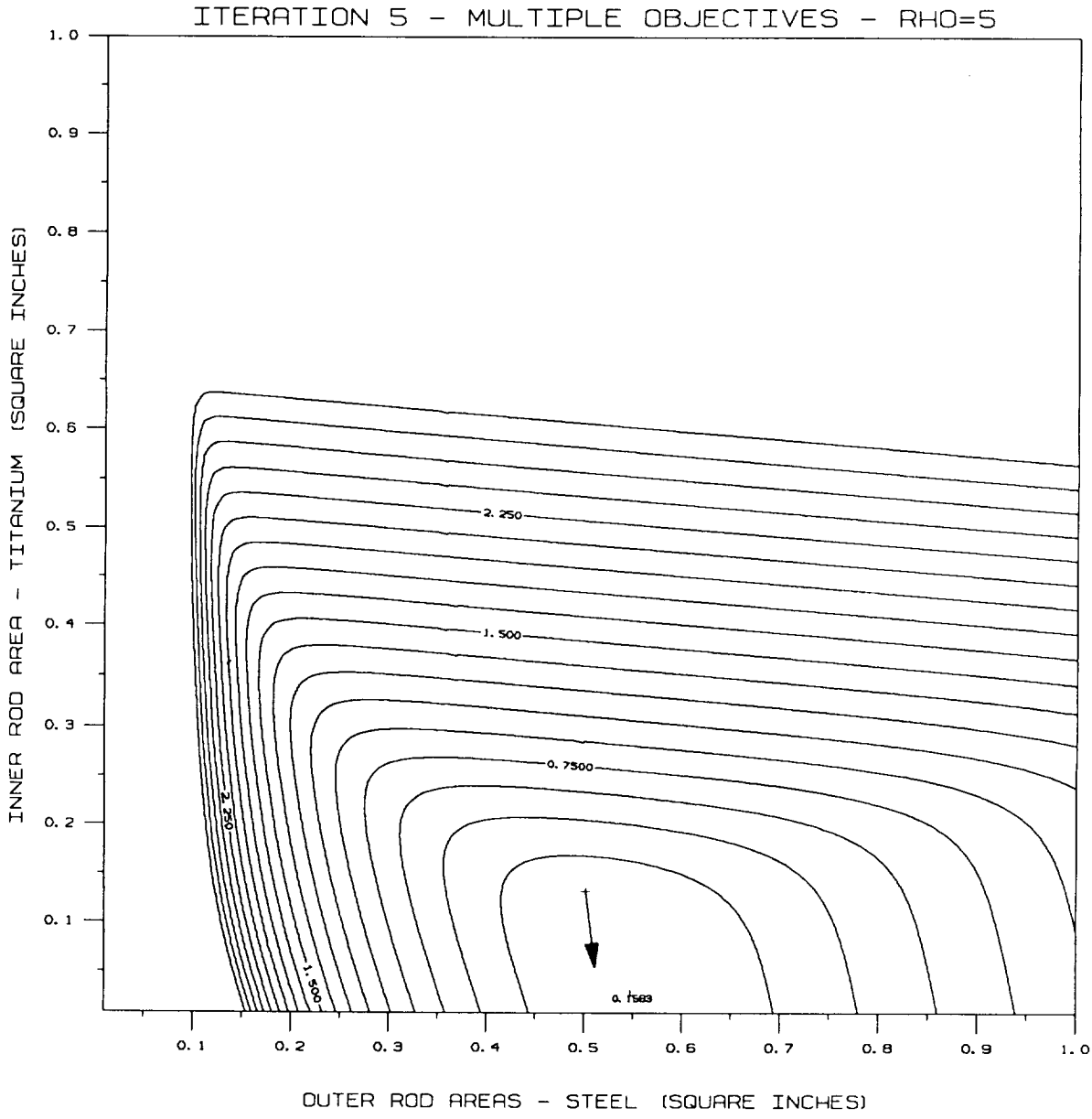


Figure 18. Contour Plot of the Three Bar Truss Design Space for Iteration 5.

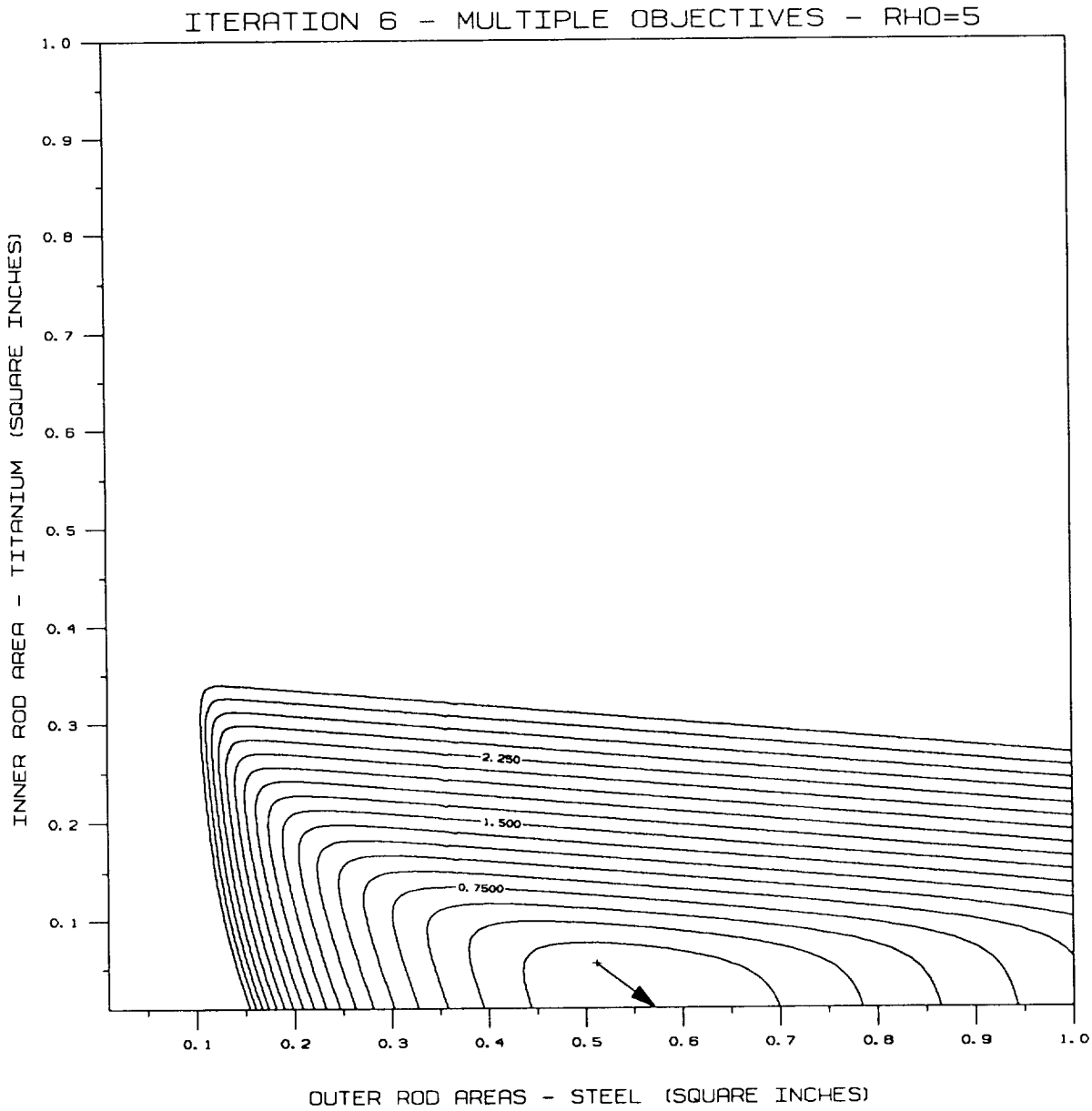


Figure 19. Contour Plot of the Three Bar Truss Design Space for Iteration 6.

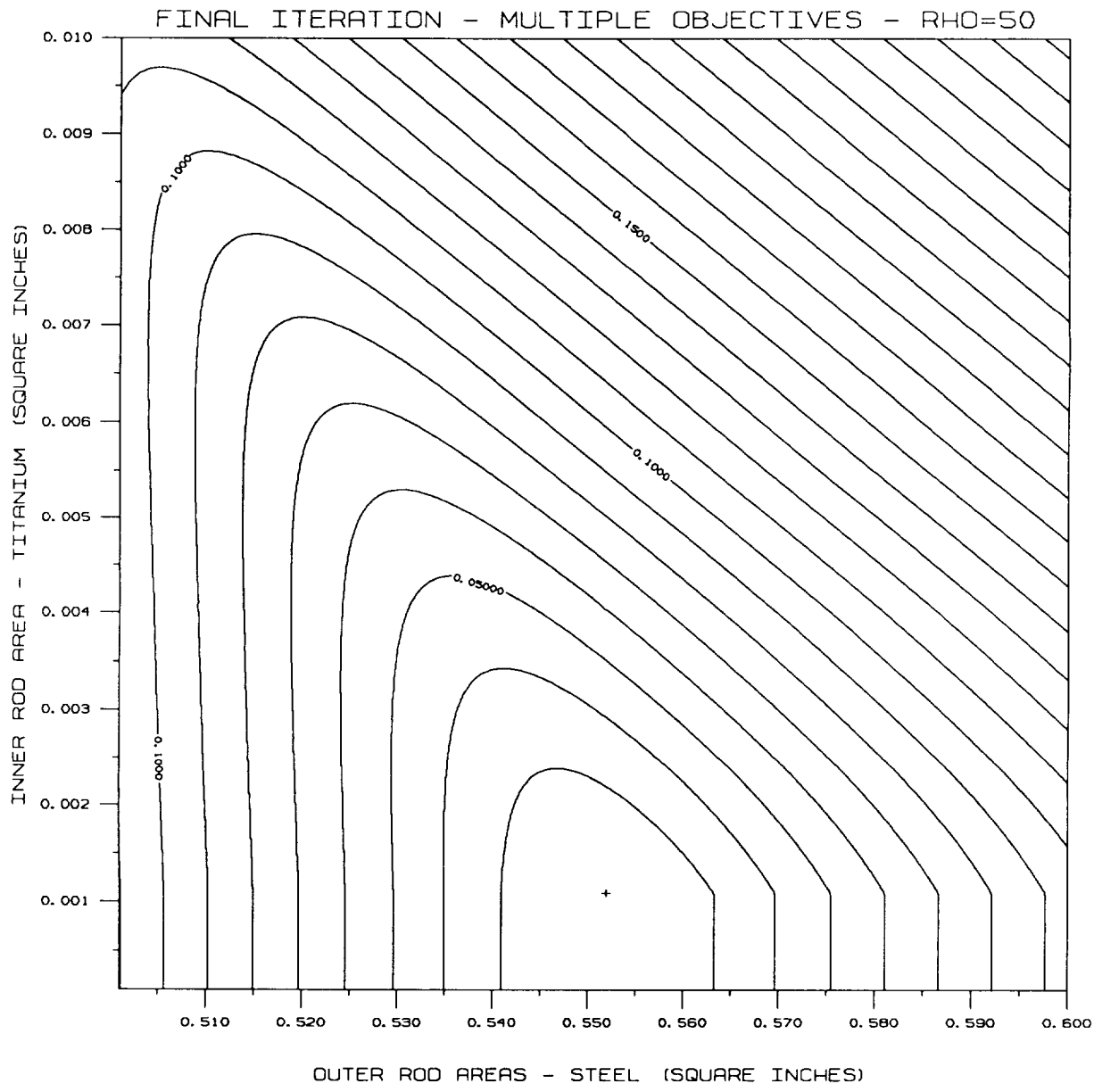


Figure 20. Contour Plot of the Three Bar Truss Design Space at the Optimum Solution.

REFERENCES

1. Kreisselmeier, G., and Steinhauser, R., "Systematic Control Design by Optimizing a Vector Performance Index," International Federation of Active Controls Symposium on Computer-Aided Design of Control Systems, Zurich, Switzerland, August 29-31, 1979.

2. Fiacco, A. V., McCormick, G. P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley, New York, 1968.

3. Vanderplaats, Garret N., Numerical Optimization Techniques for Engineering Design, McGraw-Hill Book Company, New York, 1984.

4. Fox, R. L., Optimization Methods for Engineering Design, Addison-Wesley, Reading, Mass., 1971.

5. Haftka, R. T., Starnes, J. H., Jr., "Applications of a Quadratic Extended Interior Penalty Function for Structural Optimization," AIAA Journal, Vol. 14, June 1976, pp. 718-724.

6. Rao, S. S., "Multiobjective Optimization in Structural Design with Uncertain Parameters and Stochastic Processes," AIAA Journal, Vol. 22, November 1984, pp. 1670-1678.

7. Davidon, W. C., "Variable Metric Methods for Minimization," Argonne National Laboratory, ANL-5990 Rev., University of Chicago, 1959.

8. Schmit, L. A.,"Structural Design by Systematic Synthesis," Proceedings of the Second National Conference on Electronic Computation, Structural Division, ASCE, Pittsburgh, PA, September 1960, pp. 105-132.

9. Sobieszczanski-Sobieski, Jaroslaw, Dovi, Augustine R., Wrenn, Gregory A.,"A New Algorithm for General Multiobjective Optimization," NASA TM-100536, March, 1988.

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-4220		

4. Title and Subtitle	5. Report Date
An Indirect Method for Numerical Optimization Using The Kreisselmeier-Steinhauser Function	March 1989
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Gregory A. Wrenn	
	10. Work Unit No. 506-43-41-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Planning Research Corporation Aerospace Technologies Division Hampton, VA 23666	NAS1-18000
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Jaroslaw Sobieski
Final Report

16. Abstract

This paper describes a new technique for converting a constrained optimization problem into an unconstrained problem. The technique transforms one or more objective functions into reduced objective functions, which are analogous to goal constraints used in the goal programming method. These reduced objective functions are appended to the set of constraints and an envelope of the entire function set is computed using the Kreisselmeier-Steinhauser function. This envelope function is then searched for an unconstrained minimum. The technique may be categorized as a SUMT algorithm. Advantages of this approach are the use of unconstrained optimization methods to find a constrained minimum without the "draw down" factor typical of penalty function methods, and that the technique may be started from the feasible or infeasible design space. In multiobjective applications, the approach has the advantage of locating a compromise minimum design without the need to optimize for each individual objective function separately.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Constrained optimization Unconstrained optimization methods Objective functions	Unclassified - Unlimited Subject Category 05

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	84	A05

NASA FORM 1626 OCT 86

NASA-Langley, 1989

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171