Physical Programming: Effective Optimization for Computational Design

Achille Messac

Corresponding Author
Achille Messac, Ph.D.
Distinguished Professor and Department Chair
Mechanical and Aerospace Engineering
Syracuse University, 263 Link Hall
Syracuse, New York 13244, USA

Email: messac@syr.edu
Tel: (315) 443-2341
Fax: (315) 443-3099
https://messac.expressions.syr.edu/

Bibliographical Information
Physical Programming: Effective Optimization for Computational Design

Achille Messac*
Northwestern University, Boston, Massachusetts 02115

A new effective and computationally efficient approach for design optimization, hereby entitled physical programming, is developed. This new approach is intended to substantially reduce the computational intensity of large problems and to place the design process into a more flexible and natural framework. Knowledge of the desired attributes of the optimal design is judiciously exploited. For each attribute of interest to the designer (each criterion), regions are defined that delineate degrees of desirability: unacceptable, highly undesirable, undesirable, tolerable, desirable, and highly desirable. This approach completely eliminates the need for iterative weight setting, which is the object of the typical computational bottleneck in large design optimization problems. Two key advantages of physical programming are 1) once the designer’s preferences are articulated, obtaining the corresponding optimal design is a noniterative process—in stark contrast to conventional weight-based methods and 2) it provides the means to reliably employ optimization with minimal prior knowledge thereof. The mathematical infrastructure that supports the physical programming design optimization framework is developed, and a numerical example provided. Physical programming is a new approach to realistic design optimization that may be appealing to the design engineer in an industrial setting.

I. Introduction

The endeavor of computational design has witnessed significant evolution over the past decade. In the early eighties, the realization that the sequential treatment of structure and control failed to comprehensively recognize their interdependence prompted researchers to explore related integrated design approaches.1-22 The area of multidisciplinary design optimization (MDO) has grown to the point of gaining near universal recognition in its ability to lead to better designs.23 Unfortunately, the full power of MDO has only selectively been exploited. Government organizations and academia have developed and successfully employed MDO. At the same time, industry has generally failed to accept the quantum change in its product development infrastructure that would be required for the routine exploitation of MDO technology.

In addition to the disciplines of structures and control, MDO has been employed in conjunction with such diverse areas as trajectory determination, aeroelastic tailoring, and automobile active suspension design. Knowledge-based methods have also increasingly been used in the MDO context.24 MDO-specific software is growing in numbers: (ASTROS,25 ASCYNT,26 and ENGINEOUS27). Other commercially available software have recently begun to incorporate MDO capabilities: PATRAN3, SDRC’s IDEAS, ProENGINEER, and Intergraph are among such software. At the heart of the MDO problem is a multicriteria optimization problem. A number of books and other publications have been published on the subject.

It was mentioned earlier that industry has thus far underutilized MDO technology. The reasons for this situation are many.

1) The computational resources needed for the routine application of MDO are still beyond the reach of most organizations. For realistically complex designs, supercomputing, parallel computing, or distributed computing is needed if one is to keep computation time down to accustomed levels.

2) The use of full-featured analysis software tools within an optimization-based design environment remains an awkward process that often requires significant interface-coding efforts.

3) The application of optimization in the solution of large problems is too often an undertaking that frustrates all but the most experienced in the art. Without the benefit of significant experience on the part of the designer, the optimization problem formulation often fails to include important constraints. This omission usually leads the optimization process to converge to inappropriate design points.

4) The existence of several local minima, many of which might be highly undesirable, often poses severe difficulties.

5) The rigid framework for expressing design preferences can be considered singularly undesirable. That framework is typically limited to the use of such phrases as minimize C, need A = B, need A < B, and need A > B, where the quantities A, B, and C denote important attributes of the design. Moreover, the quantity C is usually an aggregate representation of the design preferences. The constituent components of this representation typically form a set of competing objectives. Methods for reconciling the competing nature of these objectives often require repeated optimization runs, thereby further exacerbating the computational nature of MDO. The latter point is discussed in detail later.

A. Problem Statement

To illustrate the conventional approach to optimization problem statements, consider the generic problem of designing a passenger car. The problem statement might read as follows: 1) Price must be less than $11,000. 2) Weight must be minimized as much as possible. 3) Mileage must be above 26 mpg and 18 mpg for highway and city driving, respectively. 4) Vibration response due to a given road disturbance must be below a given level. 5) ....

B. Shortcomings of Problem Statement

It is too inflexible, and it provides information within the context of a limited scenario. Because the solution is that of a narrowly posed problem, the designer must ask the critical question: How sensitive is the solution to both the design specifications and the design parameters? Insensitivity to a particular design specification suggests that the designer can relax that specification with little or no consequence. Strong sensitivity to a design parameter could alert the designer of possible fabrication problems regarding dimensional tolerance. The robustness properties of the design can become a concern.

C. Archimedian vs Preemptive Formulation Approaches

All real world design problems are implicitly or explicitly multi-objective, and approaches to finding the compromise (optimal)
The choice of utility functions in utility theory, too, can be interpreted as a choice of weights. (Note that while utility theory clearly defines the properties of utility functions, it fails to provide practical means to develop such functions.) Physical programming is a method that requests physically motivated information from the part of the designer and produces a problem statement that reflects the realistic texture of the designer’s preferences.

D. Objectives of This Paper

The following specific objectives are pursued in this paper: 1) develop a general framework for problem formulation that offers the flexibility required of real-world design problems (the framework allows for deliberate imprecision in the problem statement: a flexible infrastructure for expressing how good is good and how bad is bad), 2) develop a design process that aims to reduce the notoriously high MDO computational burden, and 3) ensure that the design process developed in objective 2 is readily compatible with existing optimization software.

This paper is organized as follows. The physical programming approach is presented in Sec. II. A numerical example is provided in Sec. III, and Sec. IV presents the conclusion.

Section II is divided into seven components. In Sec. II.A, the physical programming approach is compared with the conventional weight-based optimization approach. The superiority of the former is discussed from mathematical, algorithmic, and computational perspectives. Section II.B classifies the expression of preference under the physical programming methodology. In Sec. II.C, a method for the expression of intracriteria preference is developed, where an expansive design optimization lexicon is defined. This new lexicon brings both new flexibility and rigor to the design process by employing class functions. Section II.D develops the mathematical representation of the generic class functions. That development involves the creation of a new spline that possesses certain required properties. In Sec. II.E, the class function development algorithm is presented. Section II.F develops the aggregate preference function. The physical programming problem statement model is presented in Sec. II.G.

II. Physical Programming

A. Physical Programming vs Weight-Based Approach

1. Mathematical Relationship

This section develops the mathematical relationship that exists between the physical programming based preference function, \( J_{pf} \), and the usual weighted-sum objective, \( J_w \). This relationship will help establish the superiority of the former. The difficulty of expressing a general set of preferences using the weighted-sum approach is made evident.

The weight-based objective function is expressed as

\[
J_w(g) = \sum_i w_i[g(x)]g_i(x)
\]  

(1)

where \( w_i[g(x)] \) is the weight associated with the \( i \)th criterion \( g_i \). (In many applications, the weight is chosen to be a constant.) In the preceding equation, the individual criteria are often squared before summation. That possibility does not materially impact the following discussion; the criterion is simply replaced by (or redefined as) its square. Note that \( g_i \) may represent objectives (criteria) that will be optimized or quantities that will be constrained.

The preference function based objective is written as

\[
J_{pf}(g) = J[g(x)]
\]  

(2)

The preceding two expressions can be considered equivalent if they both have the same extrema. To establish that requirement, the first variation of each is evaluated:

\[
\delta J_w(g) = \sum_i p_i(g) \delta g_i
\]  

(3)

where

\[
p_i(g) = \sum_j \frac{\partial w_j}{\partial g_i} g_j + w_i
\]  

(4)
and

$$\delta J_{pf}(g) = \sum \delta q_i(g) \delta g_i$$  \hspace{1cm} (5)$$

where

$$q_i(g) = \frac{\partial J(g)}{\partial g_i}$$  \hspace{1cm} (6)$$

Note that both \( J_p \) and \( J_{pf} \) reflect the same preference when the following differential equation is satisfied:

$$\sum \frac{\partial w_i}{\partial g_i} g_j + w_j = \frac{\partial J(g)}{\partial g_i}$$  \hspace{1cm} (7)$$

or, in matrix form,

$$\left[ \frac{\partial w}{\partial g} \right]^T g + w = \frac{\partial J(g)}{g}$$  \hspace{1cm} (8)$$

where \( w \) is a column vector whose \( i \)th entry is \( w_i \). For the case where the minimum is at the boundary of a criterion domain—and the associated criterion variation does not vanish—the appropriate row in Eq. (8) is omitted.

Some useful observations can be made from the preceding development. Let \( p \) and \( q \) be column vectors whose \( i \)th elements are denoted by \( p_i \) and \( q_i \).

1) From Eqs. (3) and (5), note that (because the first variation of the preference function vanishes at the optimum) the vectors \( p \) and \( q \) are parallel and are perpendicular to a hyperplane that is tangent to the hypersurface that borders the feasible hypervolume of the objective space (or criteria space): an intuitively satisfying geometrical interpretation of the preceding development.

2) In the case where each weight function depends only on the criterion it multiplies, and the preference function is a decoupled function of the criteria, Eq. (8) takes the form of \( m \) decoupled differential equations. Even in this simplified case, the correct weight is a complex function of the pertaining criterion.

3) In the common case where constant weight is assumed, Eq. (8) is further simplified, revealing the strict limit associated with constant weights. The preference expressed can only be a linear function of the criteria [so that the right-hand-side of Eq. (7) can also be constant].

4) To express arbitrary preferences, the correct weight vector would have to be a complex function of the criterion vector.

5) The preference function, the criteria, and the weights are, in general, related by a set of coupled differential equations. The typical practice of assigning constant weights is therefore subject to gross inadequacies.

In light of the preceding observations, the author suggests that in the typical case where competing objectives are nonlinearly dependent on the design variables, iteratively guessing the appropriate values of the weights is—at best—a poor and inefficient way to engage in the design of complex systems.

Physical programming provides the means for direct expression of the preference, which fundamentally impacts the design process. Rather than expending substantial efforts tweaking weights and re-optimizing until a given set of preferences is achieved, the designer is allowed to concentrate more on the physical problem at hand and less on the art of converging to satisfactory weights. This issue is further discussed in the following. As is discussed in Ref. 28, several attempts at forming preference functions (utility functions) have been made. All have revealed the formidable difficulty of uncovering a general approach. This publication seeks to redress that situation.

2. Generic Design Loops

The following is a discussion of the design loops involved in the weight-based methods vs those of physical programming, where a general method for obtaining the preference function is available.

Figure 1a depicts the two generic loops involved in weight-based methods. First, the designer/decision maker expresses what he/she thinks the preferences should be. Based on those stated preferences (and possibly on initial analysis), the analyst chooses a set of weights and obtains the potentially optimal design. (This step is typically computationally intensive.) The next step is to assess whether the design obtained adequately reflects the stated preferences. Then, if the answer is no, choose new weights (inner loop); if yes, evaluate adequacy of preference statement (outer loop).

The basic deficiency of this approach is the existence of the inner-loop, the iterative weight setting, which constitutes a computational bottleneck for real (nonacademic) problems. Explicitly, the problem is twofold. First, obtaining a solution for each chosen set of weights may take hours to days of computer time for complex designs. Second, it may not be evident how to obtain an adequate convergence rate even for reasonably complex problems.

It is also important to observe that although the frequency of the inner loop is potentially high, that of the outer loop is typically not. The preference statement might be changed either because it was incorrect or because the designer wishes to explore other possibilities.

The preceding discussion suggests that the design optimization process would benefit greatly from two distinct developments: 1) the means for creating a mathematical function that directly expresses a priori preferences (removal of the inner loop described earlier) and 2) the ability to express preferences in fashions that are less than rigid and deliberately imprecise (as discussed in the sequel).

Physical programming is a new method that breaks new grounds on both counts.

B. Qualitative Classification of Preference

This section presents the physical programming method for expressing preference for each generic criterion. Preference generally falls under three classes, each comprising two cases.

As is depicted in Fig. 2, the preference classes are referred to as follows: 1) class-1: smaller-is-better (SIB), 2) class-2: larger-is-better (LIB), and 3) class-3: center-is-better (CIB).

For each class, the cases hard (H) and soft (S) are defined. Exactly what the shape of the preference function for the soft curves should be is the subject of later discussions.

These classes refer to the behavior of the preference function with respect to each generic criterion. For each criterion, a class function \( g \) is formed that constitutes a component of the aggregate preference function to be minimized. A lower value of the class function is considered better than a higher value. The utopian value of the class function is zero. For example: 1) If a criterion represents purchase price—to be minimized—class 1-S would apply. 2) If a criterion represents car mileage—to be maximized—class 2-S would apply. 3) In the cases where one only cares not to exceed some limits, the hard column would apply (Fig. 2-S).

Physical programming provides the means for direct expression of the preference, which fundamentally impacts the design process. Rather than expending substantial efforts tweaking weights and re-optimizing until a given set of preferences is achieved, the designer is allowed to concentrate more on the physical problem at hand and less on the art of converging to satisfactory weights. This issue is further discussed in the following. As is discussed in Ref. 28, several attempts at forming preference functions (utility functions) have been made. All have revealed the formidable difficulty of uncovering a general approach. This publication seeks to redress that situation.

2. Generic Design Loops

The following is a discussion of the design loops involved in the weight-based methods vs those of physical programming, where a general method for obtaining the preference function is available.

Figure 1a depicts the two generic loops involved in weight-based methods. First, the designer/decision maker expresses what he/she thinks the preferences should be. Based on those stated preferences (and possibly on initial analysis), the analyst chooses a set of weights and obtains the potentially optimal design. (This step is typically computationally intensive.) The next step is to assess whether the design obtained adequately reflects the stated preferences. Then, if the answer is no, choose new weights (inner loop); if yes, evaluate adequacy of preference statement (outer loop).

The basic deficiency of this approach is the existence of the inner-loop, the iterative weight setting, which constitutes a computational bottleneck for real (nonacademic) problems. Explicitly, the problem is twofold. First, obtaining a solution for each chosen set of weights may take hours to days of computer time for complex designs. Second, it may not be evident how to obtain an adequate convergence rate even for reasonably complex problems.

It is also important to observe that although the frequency of the inner loop is potentially high, that of the outer loop is typically not. The preference statement might be changed either because it was incorrect or because the designer wishes to explore other possibilities.

The preceding discussion suggests that the design optimization process would benefit greatly from two distinct developments: 1) the means for creating a mathematical function that directly expresses a priori preferences (removal of the inner loop described earlier) and 2) the ability to express preferences in fashions that are less than rigid and deliberately imprecise (as discussed in the sequel).

Physical programming is a new method that breaks new grounds on both counts.

B. Qualitative Classification of Preference

This section presents the physical programming method for expressing preference for each generic criterion. Preference generally falls under three classes, each comprising two cases.

As is depicted in Fig. 2, the preference classes are referred to as follows: 1) class-1: smaller-is-better (SIB), 2) class-2: larger-is-better (LIB), and 3) class-3: center-is-better (CIB).

For each class, the cases hard (H) and soft (S) are defined. Exactly what the shape of the preference function for the soft curves should be is the subject of later discussions.

These classes refer to the behavior of the preference function with respect to each generic criterion. For each criterion, a class function \( g \) is formed that constitutes a component of the aggregate preference function to be minimized. A lower value of the class function is considered better than a higher value. The utopian value of the class function is zero. For example: 1) If a criterion represents purchase price—to be minimized—class 1-S would apply. 2) If a criterion represents car mileage—to be maximized—class 2-S would apply. 3) In the cases where one only cares not to exceed some limits, the hard column would apply (Fig. 2-S).

It is important to observe that if the proper shape of the soft curves can be determined, the preceding classification offers significantly more flexibility than the typical weighted-criterion approach. Figure 3 depicts the limitations of the weighted-criterion approach. As can be seen, the weighted-criterion approach simply allows the designer to modify the slope of a straight line, whereas the class functions (Fig. 2) allow the preference function to be almost arbitrarily shaped. Section II.C develops the method for prescribing the appropriate shape of the class function.
for quantitatively forming the generic class functions. This is activities of the designer. This section presents the methodical approach on its ability to create class functions that truly reflect the priori-

C. Intracriteria Preference

The effectiveness of physical programming rests substantially on its ability to create class functions that truly reflect the priorities of the designer. This section presents the methodical approach for quantitatively forming the generic class functions. This is accomplished by defining a physical programming lexicon that departs from conventional mathematical programming formalism. The development of mathematical expressions for the class functions is the subject of the next section.

1. Physical Programming Lexicon

Conventional mathematical formalism employs such terms as minimize A, maximize B, D less than E, F greater than G, or E equals F, where A–F denote quantities of interest in the design process. Physical programming explicitly recognizes the limitations of such a problem formulation framework by employing a new expansive and flexible lexicon. This lexicon comprises terms that characterize the degree of desirability of 6 ranges/domains/regions for each generic criterion for classes 1-S and 2-S, and 10 regions for class 3-S. To fix ideas, consider the case of class 1-S (Fig. 4); the criterion may not take. The property of the class function discussed earlier is expressed by the relation

\[
\bar{g}_k(g_k) \equiv \bar{g}^k \quad \forall i \quad (1 \leq k \leq 5) \quad (9)
\]

In light of the preceding discussion, it is noted that the change/improvement that takes place as one travels across the kth region reads \(\bar{g}^k\) (see Fig. 4).

The second feature of the class function that is independent of both the criterion under consideration and of the region type is as follows: To make 1) the highly undesirable region significantly worse than the undesirable region, 2) the undesirable region significantly worse than the tolerable region, 3) the tolerable region significantly worse than the desirable region, and 4) the desirable region somewhat worse than the highly desirable region, the following relations are enforced:

\[
\bar{g}^k \text{ is a small positive number} \quad (10)
\]

\[
\bar{g}^k > n_{sc} \bar{g}^{k-1}, \quad 2 \leq k \leq 5 \quad (11)
\]

Undesirable range (region 4) \(g_{l4} \leq g_i \leq g_{u4}\): an acceptable range that is undesirable.

Tolerable range (region 3) \(g_{l2} \leq g_i \leq g_{u2}\): an acceptable range that is tolerable.

Desirable range (region 2) \(g_{l1} \leq g_i \leq g_{u1}\): an acceptable range that is desirable.

Highly Desirable range (region 1) \(g_i \leq g_{l1}\): an acceptable range over which the improvement that results from further reduction of the criterion is desired but is of minimal additional value.

The preceding terms shall be understood in the following context.

2. One vs Others Criteria Rule (OVO Rule)

Consider the following two options concerning the class function values:

Option 1: Full reduction for one criterion across a given region (region \(k = 3, 4, 5\)).

Option 2: Full reduction for all of the other criteria across the next better region (region \(k = 1\)).

Option 1 shall be preferred over option 2.

In other words, it is considered better for one criterion to travel across, say, the tolerable region than it is for all the other criteria to travel across the desirable region. [See inequality (11) later.]

It is observed that, whereas the region limits define intracriterion preference, the OVO rule is a form of intercriteria preference.

3. Qualitative Discussion

First, consider the issues that are particular to a given criterion. For each criterion, one must first define the class type. This is followed by the prescription of scalar values, for each criterion, that quantitatively define the criterion’s preferred behavior via region limits (see Fig. 4). 1) For classes 1-H or 2-H, the designer must provide a single scalar value that defines the boundary of the feasible space. 2) For class 3-H, two values are needed. 3) For classes 1-S or 2-S, five values are required. 4) For classes 3-S, nine values are needed.

Second, the issues that are independent of both class type and criterion are discussed.

As is shown in Fig. 4, the value of the class function is the same at each of the region boundaries, regardless of class type or criterion (only the location of the boundary changes from criterion to criterion). As a consequence, as one travels across a given region type (say, tolerable), the change in the class function will always be of the same magnitude (\(\bar{g}^k\)). This behavior of the class function values at the boundaries is the critical factor that makes each generic region type have the same numerical consequence for different criteria. This same behavior also has a normalizing effect, and results in favorable numerical conditioning properties. For example, consider the possible behavior of two distinct criteria over the tolerable region. The first criterion varies between 900 and 15,000, whereas the second varies between 1.5 and 3.8. Since both of those regions are tolerable, the respective class functions vertical excursion over that region is the same (\(\bar{g}^k\)) regardless of the class type.

The property of the class function discussed earlier is expressed by the relation

\[
\bar{g}_k(g_k) \equiv \bar{g}^k \quad \forall i \quad (1 \leq k \leq 5) \quad (9)
\]
D. Mathematical Representation of Generic Preference

This section develops an approach for developing the mathematical representation of the generic class functions. The class functions must have the following properties:

All soft classes

(P1) strictly positive
(P2) continuous first derivative
(P3) strictly positive second derivative
(P4) all properties must hold for arbitrary choices of region boundaries

Class 1-S

(P5) strictly positive first derivative
(P6) limit \( \tilde{g}_i(g) = 0, g_i \to -\infty \)

Class 2-S

(P7) strictly negative first derivative
(P8) limit \( \tilde{g}_i(g) = 0, g_i \to \infty \)

Class 3-S

(P9) first derivative possesses one-and-only-one zero
(P10) \( \frac{d^2 g_i}{d g_i^2} = 0 \) at \( g_i = g_{i+1} \)

The justifications for the preceding properties are either obvious or provided in the next section. With these properties in mind, the objective is to develop a practical procedure for creating the generic class functions. Note that it is possible to remove the word strictly in the preceding properties in many instances by suffering only minor consequences.

In searching for a method to form the generic class function, the following observations are of interest. 1) Using a high-order polynomial with free coefficients (to be determined) is in general unacceptable, as this approach might easily violate property (P3), without extraordinary effort. 2) Conventional application of cubic spline methods is also unacceptable, as property (P3) might again be violated. The usual continuity of first and second derivative across spline segments does not necessarily satisfy property (P3).

To address the preceding difficulty, the necessary and sufficient conditions for the cubic spline method to lead to strictly positive curvature were developed [see Eqs. (34) and (35)]. Analysis of these conditions reveals that they are unduly restrictive. That is, the range of values of the criteria that lead to positive curvature for the class function is too limited. For that reason, a new spline is developed in the left-hand side of Eqs. (12) and (13): The right-hand side is the same for all criteria. The inequality (11) reflects the OVO criteria rule.

The preceding discussion offers some similarities to the goal programming method, where a good value for each criterion is defined. That good value is closely associated with the \( g^{/i} \) value defined earlier. However, the similarities between goal programming and physical programming are largely limited to the preceding observation. The more comprehensive framework of physical programming could be of pivotal value to the practical designer. In a similar vein, many readers will observe a philosophical connection between physical programming and fuzzy optimization where membership functions play a role similar to that of class functions in physical programming. A comprehensive discussion and analysis of the relative strengths of the two methods is beyond the scope of this publication.

\[ \tilde{g}_i(k) = \tilde{g}_i(g_i(k)) \quad 2 \leq k \leq 5 \quad (12) \]

\[ \tilde{g}_1 = \tilde{g}_1(g_1(k)) \quad (13) \]

and \( n_c \) is the number of soft criteria. Note the absence of the index \( i \) in the left-hand side of Eqs. (12) and (13): The right-hand side is the same for all criteria. The inequality (11) reflects the OVO criteria rule.

The development of the spline segment that possesses a strictly positive second derivative begins by defining its second derivative in the form (for region \( k \))

\[ \frac{d^2 g_i}{d g_i^2} = \left( \lambda_i \right)^2 \left[ a \left( \xi_i \right)^2 + b \left( \xi_i - 1 \right)^2 \right], \quad 0 \leq \xi_i \leq 1 \quad (14) \]

where \( a \) and \( b \) are real, strictly positive constants, and where

\[ \xi_i = \frac{g_i - g_i(k-1)}{g_i(k) - g_i(k-1)} \quad (15) \]

\[ \lambda_i = \frac{g_i(k) - g_i(k-1)}{g_i(k) - g_i(k-1)} \quad (16) \]

It follows that the first derivative reads

\[ \frac{d g_i}{d g_i} = \left( \lambda_i \right)^2 \left[ \left( \frac{a}{3} \right)^2 \left( \xi_i \right)^3 + \left( \frac{b}{3} \right)^2 \left( \xi_i - 1 \right)^3 \right] + c \lambda_i \xi_i + d \quad (17) \]

and the class function takes the form

\[ \tilde{g}_i = \left( \lambda_i \right)^2 \left[ \left( \frac{a}{12} \right)^2 \left( \xi_i \right)^4 + \left( \frac{b}{12} \left( \xi_i - 1 \right)^4 \right) \right] + c \lambda_i \xi_i + d \quad (18) \]

Equation (18) possesses four constants \( a, b, c, \) and \( d \) that are determined by prescribing four quantities: the two displacements and slopes at the region boundaries (the algorithm for obtaining these boundary conditions is described in the following).

With the knowledge of the four constants \( a, b, c, \) and \( d, \) it is convenient to express the class function in region \( k \) as

\[ \tilde{g}_i = T_0 \left( \xi_i \right) g_i(k-1) + T_1 \left( \xi_i \right) g_i(k) \]

\[ + \tilde{g}_i \left( \xi_i; \lambda_i \right) S_i(k-1) + \tilde{g}_i \left( \xi_i; \lambda_i \right) S_i(k) \quad (19) \]

where \( 0 \leq \xi_i \leq 1, k = 2, \ldots, 5, \) and

\[ S_i(k) = \frac{d g_i}{d g_i} \quad (20) \]

\[ T_0 \left( \xi_i \right) = \frac{1}{2} \left( \xi_i - 1 \right)^4 - 2 \xi_i + \frac{3}{2} \quad (21) \]

\[ T_1 \left( \xi_i \right) = \frac{1}{2} \left( \xi_i - 1 \right)^4 + 2 \xi_i - \frac{3}{2} \quad (22) \]

\[ \tilde{g}_i \left( \xi_i, \lambda_i \right) = \lambda_i \left[ \frac{1}{3} \xi_i^3 - \frac{1}{4} \left( \xi_i - 1 \right)^3 \right] + \frac{3}{2} \xi_i + \frac{1}{2} \quad (23) \]

Equations (19–24) define the class function (class 1-S) for the \( i \)th criterion in the domain of the 4th region (\( k = 2, \ldots, 5 \)), if indeed the constants \( a \) and \( b \) are strictly positive. It is interesting to note the similarities between the preceding functions [Eqs. (21–24)] and the Hermite polynomials. The former more readily satisfy the present curvature requirements.

To establish the conditions under which this curvature positivity holds, observe from Eq. (14) that the constants \( a \) and \( b \) must be positive. To uncover the conditions under which those conditions are satisfied, \( a \) and \( b \) are evaluated as

\[ a = \frac{3 \left[ 3 S_i(k) + 3 S_i(k-1) \right] - 12 \lambda_i^2}{2 \left( \lambda_i^2 \right)^3} \quad (25) \]

and

\[ b = \frac{12 \lambda_i^2 - 3 \left[ 3 S_i(k) + 3 S_i(k-1) \right]}{2 \left( \lambda_i^2 \right)^3} \quad (26) \]

where

\[ \lambda_i^2 = \frac{\Delta k^2}{\lambda_i^2} \quad (k = 2, 3, 4, 5) \quad (27) \]

Note that the variable \( \lambda_i^2 \) is a characteristic slope for the \( k \)th region of the \( i \)th criterion. It comprises two components: the first \( (\lambda_i^2) \) represents the improvement that results from traveling across the \( k \)th region for any criterion; the second \( (\lambda_i^2) \) represents the length...
of the same region for a given criterion. The first has a normalizing effect, whereas the second takes into account the preferred behavior of a given criterion.

Using Eqs. (25) and (26), it can be shown that the requirement that $a$ and $b$ be strictly positive is satisfied when the following two relations hold:

$$[s_{(k-1)}]_{\text{min}} \leq s_k \leq [s_{(k-1)}]_{\text{max}}, \quad k = 2, \ldots, 5 \quad (28)$$

$$[s_k]_{\text{min}} \leq [s_k]_{\text{max}}, \quad k = 2, \ldots, 5 \quad (29)$$

where

$$[s_{(k-1)}]_{\text{min}} = \frac{4s_k^2 - 3s_k}{3}, \quad [s_{(k)}]_{\text{max}} = \frac{4s_k^2 - s_k}{3} \quad (30)$$

$$[s_k]_{\text{min}} = \frac{4s_{(k-1)}^2 - 3s_{(k-1)}}{3}, \quad [s_k]_{\text{max}} = 4s_{(k-1)}^2 - 3s_{(k-1)} \quad (31)$$

The preceding relations are used in the class function algorithm. They represent the constraints on the left boundary slope in terms of the right boundary slope [Eq. (28)] and vice versa [Eq. (29)]. To verify the effectiveness of this new spline, the magnitudes of the range of acceptability of the boundary slopes are evaluated and expressed as

$$\Delta s_{(k-1)} = [s_{(k-1)}]_{\text{max}} - [s_{(k-1)}]_{\text{min}} = \frac{2}{3}(s_k - s_k^2) \quad (32)$$

$$\Delta s_k = [s_k]_{\text{max}} - [s_k]_{\text{min}} = \frac{2}{3}(s_k^2 - s_{(k-1)}) \quad (33)$$

The flexibility offered by this new spline is quantified by comparing these results with those obtained by the cubic spline method. By performing the same analytical development for the case of the cubic spline, one finds the magnitude of the range of acceptable slopes to be

$$\Delta s_{(k-1)} = \frac{2}{3}(s_k - s_k^2) \quad (34)$$

$$\Delta s_k = \frac{2}{3}(s_k^2 - s_{(k-1)}) \quad (35)$$

Observe that the $\frac{2}{3}$ factor is replaced by $\frac{4}{3}$. Therefore, the new spline offers a 78% improvement in range of acceptable slope over the cubic spline. The author finds that this improvement is of great significance in the practical implementation of physical programming.

Finally, for region 1, the class function expression is represented by an exponential function. An acceptable mathematical expression reads

$$g_i = g_i \exp \left[ \left( s_k^2 - g_i \right) / g_i \right] \quad \text{for} \quad (g_i \leq g_i) \quad (36)$$

As stated earlier, the preceding discussion applies to class 1-S. The case of classes 2-S and 3-S follows readily from the previous discussion, with minor modifications. A few comments regarding those changes follow.

1) The case of class 2-S is the mirror-image of class 1-S.
2) The case of class 3-S requires the inclusion of nine criteria. These values need not possess any symmetry about the minimum of the class function. However, as discussed earlier, the class function vertical excursion across any given region is of the same magnitude for all classes (see Fig. 4).

E. Class Function Evaluation Algorithm

In Sec. II.D, the mathematical expressions for the generic class function are developed within the context of a single class function. For a given optimization/design problem, it is necessary to fully define all of the class functions before the aggregate preference function can be formed.

The following outlines the algorithm for determining the class function value for a generic criterion value $g_i$:

- Provide the input values:
  - $i$: criterion number, under consideration
  - $g_i$: value of $i$th criterion
  - $g_i$: region boundaries (see Fig. 4)
  - $n_{sc}$: number of soft criteria

Evaluate

$$g_i^1 = \frac{g_i}{g_i} = 0.1 \quad (37)$$

$$g_k^k = \beta n_k \frac{g_i}{g_i^k} \quad (2 \leq k \leq 5) \quad (38)$$

$$g_i^k = g_i(g_i^k) + g_i^k \quad (2 \leq k \leq 5) \quad (39)$$

[See Eqs. (9), (12), and (13).]

$$\lambda_k^k = g_i(g_k) - g_i(g_k) \quad (40)$$

$$z_k^k = \frac{g_k^k}{\lambda_k^k} \quad (k = 2, 3, 4, 5) \quad (41)$$

(for class 1-S)

$$s_{11} = \alpha z_{11}^2, \quad 0 < \alpha < 1 \quad (42)$$

$$s_k = (s_k)_{\text{min}} + \alpha \Delta s_k \quad (k = 2, 3, 4, 5) \quad (43)$$

[See Eqs. (31) and (33).] Obtain $g_i(g_k)$ using Eq. (19) or (36).

A few observations are in order: 1) Letting the constant $\alpha$ be small (less than 0.1) has the desired effect of maximizing the range of acceptable slopes at the region boundaries, that is, those that maintain a positive curvature. 2) The last two steps of the preceding algorithm apply to the class 1-S. A similar approach holds for the classes 2-S and 3-S. 3) The parameter $\beta$ can be considered a convexity parameter. From the inequality (11), it is seen that $\beta$ must be greater than 0. Furthermore, the value of $\beta$ is used to enforce the convexity of all of the class functions as follows. After determining all of the class function values and slopes at all the region limits, it is verified that the constants $\alpha$ and $\beta$ for each region are strictly positive using Eqs. (35) and (26). If this is not the case, $\beta$ is increased, and new slopes and class function values are computed. It can be shown that this iterative process converges and does so quickly. In practice, starting with $\beta = 1.5$ and increasing by 0.5 works well.

F. Aggregate Preference Function Development

This section develops the mathematical representation of the aggregate preference function. To promote the robust implementation of physical programming, it is important to ensure that the form of the preference function is not responsible for the unintended introduction of new local minima into the problem. Gradient-based nonlinear programming codes would perform poorly in the solution phase. For the purpose of exploring areas of potential difficulties, the case of a single criterion with a single design parameter is discussed first. This is followed by the more general case.

1. One-Criterion with One-Parameter Case

In the case of a single criterion, with a single design variable, the preference function is assumed to simply involve the relevant class function. The quantities of interest are the design parameter $x$, the criterion expression that is governed by the physics of the problem $g$, and the class function that expresses preference $g$.

Figure 5 depicts the various possibilities. In the first row (row 0), the behavior of the criterion function $g(x)$ is shown to comprise four possibilities: strictly increasing ($s_i$), strictly decreasing ($d_i$), minimum (min), and maximum (max). The first column (column 0) depicts the three preference function possibilities. With the exclusion of the first row and column, the remaining entries describe the behavior of the function $g(x)$, which is to be minimized. Figure 5 exposes the cases where $g(x)$ possesses more than one minimum or no finite minimum.

The behavior of $g(x)$ can be analyzed by evaluating the limits

$$\lim_{x \to \pm \infty} g(x) = g_0 = \lim_{x \to \pm \infty} g(x) \quad (44)$$

and by examining the derivatives

$$\frac{\partial^2 g(x)}{\partial x^2} = \frac{\partial^2 g}{\partial x^2} \quad (45)$$

Recall that $\partial^2 g / \partial x^2 > 0$. 

$$\frac{\partial^2 g(x)}{\partial x^2} = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial x^2} \quad (46)$$
The specific cases of Fig. 4 are discussed as follows, in light of Eqs. (44–46):

1) The entries (1, 1), (1, 2), (2, 1), and (2, 2) represent situations where constraints are necessary if the minimum is to be finite.
2) The entries (1, 3), (2, 4), (3, 1), and (3, 2) denote cases of a well-behaved function that possesses a single finite minimum.
3) The cases of entries (1, 4) and (2, 3) possess finite minima only when appropriate constraints are enforced. The potential for two minima, one at each endpoint, is unfortunately present.
4) The entries (3, 3) and (3, 4) represent troublesome situations, where a single-minimum or single-maximum criterion $g$ is transformed into a two-minimum function assuming no constraints.

It is important to note that the undesirable situations described in case 4 are not uncommon, as the weighted-sum-square method easily leads to those situations.

Further insight into the behavior of the preference function is obtained by analyzing the structure of the second derivative [Eq. (46)].

Equation (46) shows that when $\frac{\partial^2 g}{\partial x^2} > 0$, the quantity $\frac{\partial^2 g}{\partial x^2}$ is strictly positive in the cases of entry (1, 1), (1, 2), and (1, 3). Equation (46) also shows that when $\frac{\partial^2 g}{\partial x^2} < 0$, the quantity $\frac{\partial^2 g}{\partial x^2}$ is strictly positive in the cases of entry (2, 1), (2, 2), and (2, 4).

Moreover, from Eq. (46), it is seen that even in the case of entries (3, 1) and (3, 2), which possess single minima, there is no guarantee that $\frac{\partial^2 g}{\partial x^2} > 0$. By observing the first term of the right-hand side of Eq. (46), it is seen that the lack of guarantee for entries (3, 1) and (3, 2) can be alleviated by letting $\frac{\partial^2 g}{\partial g^2}$ be strongly positive. (The algorithm for choosing the convexity parameter $\beta$ exploits this fact.)

The aggregate preference function allows for the guarantee that $\frac{\partial^2 g}{\partial x^2} > 0$. As shown in Sec. II.F.2, the said guarantee plays a key role in the formation of the aggregate preference function.

### General Preference Function Formation

In this section, the method for forming the aggregate preference function is developed. The objective is to form a function that reflects the preferences expressed in each class function. The most direct approach is to simply add the individual class functions. But in doing so, it is important to ensure that the said addition does not lead to a multiple-minimum problem. Although this paper addresses the problem of multiparameter optimization, the following discussion follows the assumption of a single design parameter. The detailed analysis of the multiparameter case is the subject of ongoing studies.

[Recall that the previous section discussed the number of minima of $g(x)$, given those of $g_i(x)$.] This section discusses the number of minima of the sum $g = \sum g_i(x)$, given those of $g_i(x)$.

Since the present interest is to determine when the sum $g$ possesses a single minimum, it is important to examine the behavior of the second derivative of the said sum. (Again, it is recalled that a function that has strictly positive second derivative has at most one minimum.) In that light, it is observed that a sufficient condition for $\frac{\partial^2 g}{\partial x^2} > 0$ to hold is that $\frac{\partial g}{\partial x} > 0$ for at least one value of $i$, whereas $\frac{\partial^2 g}{\partial x^2} \geq 0$ for the other values of $i$. The cases where the condition $\frac{\partial^2 g}{\partial x^2} > 0$ holds have been discussed in the previous section. As was discussed in that section, the behavior of $\frac{\partial^2 g}{\partial x^2}$ depends on that of $\frac{\partial^2 g_i}{\partial x^2}$, over which the designer has no control. Fortunately, the designer does have control over the quantity $\frac{\partial^2 g_i}{\partial x^2}$. Equation (46) shows that the said flexibility can be exploited to make $\frac{\partial^2 g}{\partial x^2} > 0$, even when $\frac{\partial^2 g_i}{\partial x^2}$ does not possess the desired property.

With the preceding comments in mind, the aggregate preference function is assumed to take the form

$$\hat{g} = \frac{1}{n_e} \sum_{i=1}^{n_e} g_i \tag{47}$$

For computationally motivated reasons, the actual function minimized is

$$g = \log_{10} \hat{g} \tag{48}$$

The quantities $g$ and $\hat{g}$ offer several computationally desirable features. 1) The scalar $\hat{g}$ forces the optimization search path to be heavily governed by the criteria of the worst regions. This is largely due to enforcement of the relations Eqs. (11) and (38), which brings some mini-max features to the problem structure, where physically motivated regions govern the optimization path. 2) Because the quantity $\hat{g}$ emphasizes the worst regions, it is possible to ignore the criteria that are in the much better regions and to periodically monitor the situation. As some of the ignored criteria become important, they are included. This exclusion/inclusion of selected criteria can be automated by periodically monitoring the values of the pertaining class functions. Since the computational intensity of the evaluation of the objective function is a critical factor in the optimization of large real-world problems, this economy of computation should not be ignored. 3) The taking of the logarithm in forming $g$ has the desired result of mapping a domain that spans several orders of magnitude [largely because of Eqs. (11) and (38)] to one that typically involves one order of magnitude, $\hat{g}$. This result is desired because most optimization codes work best when the values of the objective span less than two or three orders of magnitude. In all of the cases analyzed, the taking of the logarithm had the effect of reducing the number of iterations by at least 20% (while leading to the same optimum point).

The aggregate preference function is employed in the physical programming problem statement of the next section.

### Physical Programming Problem Model

Building on the previous development, the physical programming problem statement takes the form

$$\min g(x) = \log_{10} \left\{ \frac{n_e}{n} \sum_{i=1}^{n} g_i[g_i(x)] \right\} \quad \text{(for soft classes)} \tag{49}$$

subject to

$$g_i(x) \leq g_i \quad \text{(for class 1-S)} \tag{50}$$

$$g_i(x) \geq g_i \quad \text{(for class 2-S)} \tag{51}$$

$$g_{3L} \leq g_i(x) \leq g_{3R} \quad \text{(for class 3-S)} \tag{52}$$

$$g_i(x) \leq g_M \quad \text{(for class 1-H)} \tag{53}$$

$$g_i(x) \geq g_M \quad \text{(for class 2-H)} \tag{54}$$

$$g_M \leq g_i(x) \leq g_M \quad \text{(for class 3-H)} \tag{55}$$

$$x_{im} \leq x_i \leq x_{jm}$$

where $g_{im}, g_{im}, x_{im},$ and $x_{jm}$ represent minimum and maximum values.

As can be seen, the preceding problem model conforms to the framework of most nonlinear programming codes, with possible minor rearrangements.
III. Example

A. Problem Definition

1. Plant Definition

The plant under consideration is idealized as a pinned-pinned sandwich beam that supports a motor (see Fig. 6). A vibratory disturbance (at \( v \) Hz) is imparted from the motor onto the beam, which is of length \( L \), of width \( b \), and symmetrical about its midplane. The variables \( d_i \) and \( d_j \), respectively, locate the contact of materials one and two, and two and three. The variable \( d_l \) locates the top of the beam. The mass density, Young's modulus, and cost per unit volume for materials one, two, and three are, respectively, denoted by the triplets \((\rho_1, E_1, c_1)\), \((\rho_2, E_2, c_2)\), and \((\rho_3, E_3, c_3)\). The mass of the motor is ignored in the following analysis.

In the design of the plant, the quantities of interest to the designer—or criteria—are as follows:

- **Fundamental frequency:**
  \[ g_1 \equiv f_1 = \left(\frac{\pi}{2L^2}\right)\left(EI/\mu\right)^\frac{1}{4} \]  
  (57)

  \[ E I = \left(\frac{2b}{3}\right)\left[ E_1 d_1^4 + E_2 \left(d_2^4 - d_1^4\right) + E_3 \left(d_3^4 - d_2^4\right)\right] \]  
  (58)

  \[ \mu = \left(\frac{2b}{3}\right)\left[ \rho_1 d_1 + \rho_2 (d_2 - d_1) + \rho_3 (d_3 - d_2)\right] \]  
  (59)

- **Cost:**
  \[ g_2 = c = 2b[c_1 d_1 + c_2 (d_2 - d_1) + c_3 (d_3 - d_2)] \]  
  (60)

- **Width:**
  \[ g_3 = b \]  
  (61)

- **Length:**
  \[ g_4 = L \]  
  (62)

- **Mass:**
  \[ g_5 = \mu L = m \]  
  (63)

- **Height (semiheight):**
  \[ g_6 = d_3 \]  
  (64)

- **Width of layer 1:**
  \[ g_7 = d_1 \]  
  (65)

- **Width of layer 2:**
  \[ g_8 = d_2 - d_1 \]  
  (66)

- **Width of layer 3:**
  \[ g_9 = d_3 - d_2 \]  
  (67)

The design parameters are

\[ x = \{d_1, d_2, d_3, b, L\} \]

Note that design parameters can form a subset of the criteria if certain preferences concerning their values must be expressed.

B. Results and Discussion

The optimal design results are presented in Tables 1–3. The first two present numerical results, whereas the third displays a qualitative visualization of the behavior of each criterion. In Table 3, a rightward arrow represents an improvement for the respective criterion. The frequency and the width experienced substantial improvements. The mass and the height showed moderate improvements. The cost remained in the desirable region without appreciable change. The length worsened but remained in the desirable region. The thicknesses of all of the layers remained in the acceptable region. However, the thickness of layer 2 reached its minimum, which points
to the fact that it is of dubious utility in this particular design. Its material is relatively weak and expensive. When different input constants are used, the presence of the second layer can become highly iterative weight tweaking, which does not necessarily converge (inner loop of Fig. 1a). This deterministic iteration is entirely distinct from the typical highly undesirable form the backbone of this new framework and a conventional comparison, the author would only expect physical programming to be highly competitive. But such a comparison would miss the point of this paper entirely. Such a comparison would implicitly assume that, given the designer's preference, method x provides the designer with the means to create the correct optimization problem statement—the solution of which would lead to the optimal design becomes a less elusive and more deterministic process. Second, physical programming significantly impacts the computational burden by entirely eliminating the uncertain, time-consuming, and often frustrating process of weight tweaking. In the case of large and complex problems, this new approach may bring new possibilities. Physical programming is not a new approach to nonlinear programming; rather it is a new approach to using nonlinear programming for design optimization—a way that may be appealing to the design engineer in an industrial setting.

IV. Conclusion

This paper presented a novel approach (physical programming) to design optimization that offers two key advantages over conventional approaches. First, it offers a problem formulation and solution framework that conforms to real-life design, where an expansive new lexicon is employed. The terms desirable, tolerable, undesirable, and highly undesirable form the backbone of this new framework and bring a new flexibility, rigor, and deliberate imprecision to the design process. Reaching the optimal design becomes a less elusive and more deterministic process. Second, physical programming significantly impacts the computational burden by entirely eliminating the uncertain, time-consuming, and often frustrating process of weight tweaking. In the case of large and complex problems, this new approach may bring new possibilities. Physical programming is not a new approach to nonlinear programming; rather it is a new approach to using nonlinear programming for design optimization—a way that may be appealing to the design engineer in an industrial setting.

References


