Chapter 3: An overview of Linear and Non-linear Programming methods for Structural Optimization

Chapter details

Chapter DOI:

https://doi.org/10.4322/978-65-86503-71-5.c03

Chapter suggested citation / reference style:

Choze, Sergio B., et al. (2022). "An overview of Linear and Non-linear Programming methods for Structural Optimization". *In* Jorge, Ariosto B., et al. (Eds.) *Model-Based and Signal-Based Inverse Methods*, Vol. I, UnB, Brasilia, DF, Brazil, pp. 65–106. Book series in Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity.

P.S.: DOI may be included at the end of citation, for completeness.

Book details

Book: Model-based and Signal-Based Inverse Methods

Edited by: Jorge, Ariosto B., Anflor, Carla T. M., Gomes, Guilherme F., & Carneiro, Sergio H. S.

Volume I of Book Series in:

Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity

Published by: UnB City: Brasilia, DF, Brazil Year: 2022

DOI: https://doi.org/10.4322/978-65-86503-71-5

Overview of Linear and Non-linear Programming Methods for Structural Optimization

Sergio B. Choze[1*] Rogerio R. Santos[2]
Ariosto B. Jorge[3] Guilherme F. Gomes[4]

[1]Consultant. E-mail: sergio.butkewitsch@gmail.com
[2]Division of Mechanical Engineering, Technological Institute of Aeronautics, Brazil. E-mail: rsantos9@gmail.com
[3]Post-Graduate Program - Integrity of Engineering Materials, University of Brasilia, Brazil. E-mail: ariosto.b.jorge@gmail.com
[4]Mechanical Engineering Institute, Federal University of Itajuba - UNIFEI, Brazil. E-mail: guilhermefergom@unifei.edu.br

*Corresponding author

Abstract

This chapter describes the theoretical and practical aspects of deterministic optimization methods. The introductory section describes the basic equations commonly found in numerical optimization and lists some standard approaches in structural optimization. Next, the concepts of Multi-criteria Optimization and Multidisciplinary Design Optimization are described. The rest of the text is devoted to clarifying specific variations of the methods in the context of Linear and Non-linear Programming. Unconstrained and constrained cases are handled. Algorithms and source code fragments are presented to enrich the discussion about the methodologies. Some practical considerations for applied optimization are described throughout the text.

1 Introduction

As a decision-making problem crafted with mathematical formality, an optimization procedure aims at finding the extreme point that minimize or maximize the value of a function (Vanderplaats [1998], Butenko and Pardalos [2014]) as defined in Equation (1)

$$\min, \max[F(\{X\})] \tag{1}$$

expressing the objective function F, dependent upon an n-dimensional vector of input variables $\{X\}$, which contains the parameters to be determined/modified in order to achieve the desired optimization/decision goals. As such, these parameters are usually called decision or design variables, and relate directly to the resources

available to supply a certain demand Haldar and Mahadevan [1999]. In the particular context of structural optimization, these decision variables will determine the load bearing capabilities (supply) to resist the operating conditions (demand).

In a realistic decision-making context, however, changes applied in the content of $\{X\}$ will likely affect not only the objective F, but also produce side effects in other quantities. These other quantities are most plausibly not allowed to vary freely and thus represent constraints that need to be factored into the determination of $\{X\}$ in the pursuit of improving F. Moreover, it is customary to classify these constraints into 2 categories, defined in Equations (2) and (3) as inequality and equality constraints, respectively:

$$G_j(\{X\}) \leq 0 \tag{2}$$

$$H_k(\{X\}) = 0 \tag{3}$$

Inequality constraints represent more flexible considerations corresponding to a threshold, as for example when the stress in a structural member should not exceed an allowable, but a margin of safety ($\leq$) would not be, barring any other considerations, harmful. On the other hand, if a structural failure mode is deliberately designed to undertake some pre-determined course or sequence, then the load bearing capabilities of its weakest link (and all others thereafter, for that matter) must fulfill a much more stringent target, giving rise to equality constraints. Please also note that the nominal values for the failure limits in these 2 scenarios may differ significantly, and it is standard practice to get them normalized by moving their values into the left-hand side of inequality (2) and equality (3) relationships, both then uniformly defaulted to 0, which becomes the reference for all equality and inequality constraints alike (they are considered violated once they surpass zero).

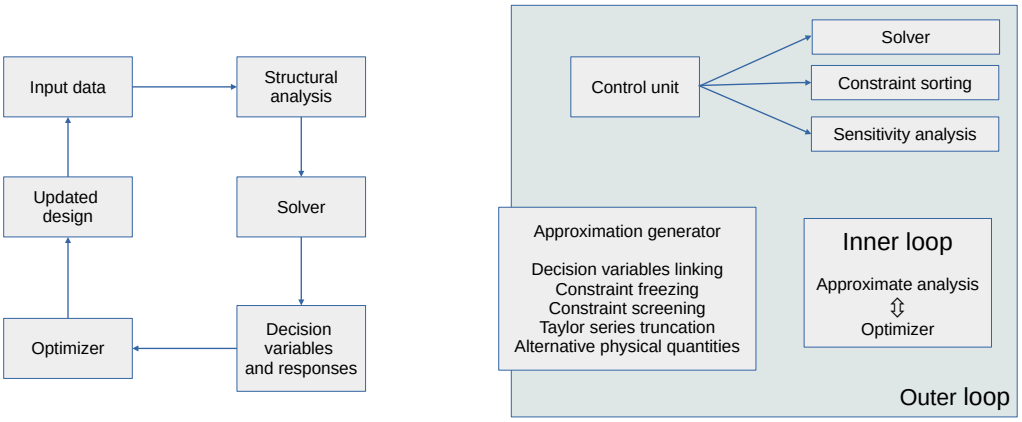
In addition to the constraints that arise in the form of functions other than the objective, a special kind of limitation acknowledges the fact that the decision variables are themselves restricted, as are the resources associated with structural sizing and material properties. These are the so-called side constraints, which complete the statement of an optimization problem by specifying limits, both lower (LB) and upper (UB) bounds, onto the decision variables, as in Equation (4):

$$\{X\}^{LB} \leq \{X\} \leq \{X\}^{UB} \tag{4}$$

Mass, strength, stiffness and natural frequencies and associated mode shapes of vibration are examples of objectives/constraints relevant to structural design, and geometry/materials/manufacturing processes are examples of the collectively shared decision variables.

2 Structural Optimization

The set of $j+k+1$ functions defined in Equations (1)-(3) all share the same status, being responses (or outcomes) that depend on the same set of decision variables (or inputs) $\{X\}$. As such, they are in principle interchangeable in terms of which response is the objective and which ones are to be constrained. The caveat is that



(a) Direct link (b) Approximate link

Figure 1: Comparison of the direct and approximate link between the Finite Element solver and the optimizer.

some of these combinations (alternative formulations for the optimization problem) are more amenable for solution by existing methods than others. Accordingly, the practice of structural optimization since its early days has relied upon specific formulations intended to streamline the solution processes. More recently, even with the emergence of powerful computational resources, these applied optimization frameworks continue to be improved with focus on formulation, most often dedicated to mediating the transfer of data between a Finite Element solver and optimization methods to be detailed in the remainder of the current chapter.

In this context, the direct coupling between analysis and optimization modules, as indicated in Figure (1a), results in the evaluation of the entire numerical model, by means of the analysis module, at all iterations performed by the optimizer along its search for an optimal solution. Since the CPU effort associated to a single analysis is multiplied by the (usually high) number of function evaluations during optimization, this scheme is often associated with a computational overhead that discourages its use.

Based on the fact that not all the parts of the model contribute all the time, and with the same intensity/relevance to the progress of the optimization procedure, a set of algorithms known as structural synthesis techniques (Schmidt, 1960) has been proposed to rationalize the use of computer resources when coupling optimizers to numerical analysis software. Figure (1b) presents the framework of the synthesis approach.

It is important to highlight the role of the approximate problem generator in the framework presented in Figure (1b). This set of algorithms is the ultimate responsible for the feasibility of coupling numerical optimizers to analysis software, since its use results in significant drop of the need for computer resources in comparison to the optimization scheme presented in Figure (1a). The methods listed therein operate described in the next 4 topics, for the duration of this section.

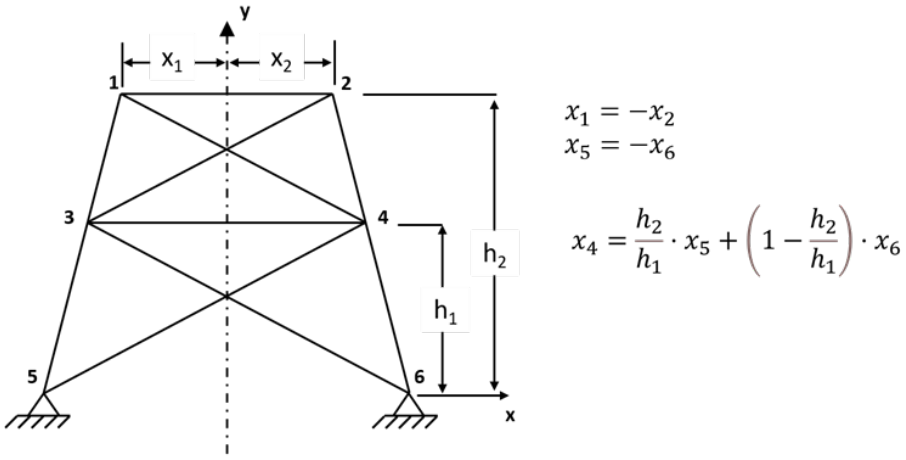


Figure 2: Schematic depiction of variable linking in a truss type of structure to generate approximations capable of reducing the computational cost of structural optimization, while simultaneously enforcing symmetry.

2.1 Decision Variable Linking

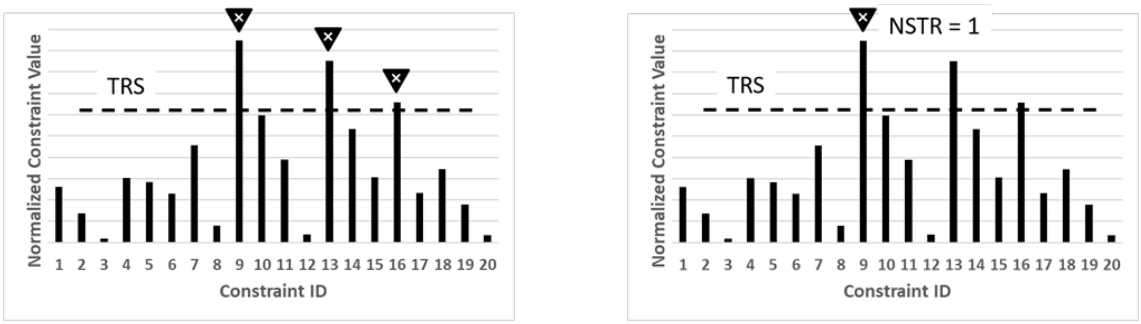
By establishing linear relations among design variables, one can reduce the number of independent variables to be evaluated. Economy of CPU resources is noticeable due to the drop in the number of partial derivatives of the responses, to be calculated with respect to the independent decision variables. Additional advantages introduced by this technique are: 1) The relations among decision variables are directly controlled by the user, which helps to keep physical insight; 2) The laws of dependence can be useful to enforce desirable design features, such as symmetry and parallelism, as shown in Figure (2).

2.2 Constraint Freezing and Screening

If a given subset of all the prescribed constraints has no risk of violation, it is useless to waste CPU time with their evaluation and the calculation of their derivatives with respect to the design variables. Hence, for the sake of feasibility, the constraints far from the violation threshold (TRS) can be neglected until their importance grows (i.e., risk of violation arises) at a different stage of the automated design process. This concept is illustrated graphically in Figure (3a).

For the purpose of further CPU economy, it should be considered that numerical models are discrete, which leads to the unfolding of the constraints prescribed in the statement of the optimization problem into a much larger number of components. For instance, consider a group of thousands of shell finite elements employed to model an aeronautic airfoil. If one desires to minimize the structure's weight keeping track of stress levels, constraints must be prescribed over all the elements of the airplane's wing. Just a few (NSTR) elements, however, can be considered at each optimization iteration, due to their superior representativity in comparison with the others. The effect of NSTR on constraint screening is illustrated in Figure (3b).

It should be noted that the procedures of constraint freezing and constraint



(a) (b)

Figure 3: Schematic illustration of the combined constraint freezing/constraint freezing procedure. Constraints are normalized.

screening are overlapped (i.e., used in conjunction) in order to avoid the heavy calculations involved in the evaluation of unnecessary constraint functions: the number of such functions is thus reduced to the least possible.

2.3 Taylor Series Truncation

Up to this point, the model reduction techniques presented acted on quantitative basis, that is, alternatives to reduce the number of independent design variables and constraints were indicated. Although these solutions are effective, more CPU power can be saved by addressing the qualitative aspects related to the functions evaluated during the optimization procedure. If several simple functions are eliminated by means of constraint deletion and screening but a few very complex, highly non-linear functions remain, still too much computer effort will result. For this reason, simplification by linearization may be beneficial, and this can be performed by expanding the functions in a Taylor Series to be truncated at a lower order/first term, as indicated in the Equations (5) and (6) below:

$$f(x^0 + \Delta x) = f(x^0) + \left.\frac{df}{dx}\right|_{x^0} \cdot \Delta x + \left.\frac{d^2 f}{dx^2}\right|_{x^0} \cdot \frac{\Delta x^2}{2!} + \left.\frac{d^3 f}{dx^3}\right|_{x^0} \cdot \frac{\Delta x^3}{3!} + \cdots \quad (5)$$

$$\hat{f}(x^0 + \Delta x) = f(x^0) + \left.\frac{df}{dx}\right|_{x^0} \cdot \Delta x \quad (6)$$

where the approximation of function f within an interval Δx around the reference point x^0 can the represented by a Taylor series expansion, with as many terms as are deemed necessary and sufficient to balance accuracy and expediency including, at the limit, just the initial linear term.

2.4 Alternative Physical Quantities

Still in the effort of simplifying the functions involved in the numerical non-linear optimization process, a special group of algorithms was developed, having in mind engineering situations often present in design optimization tasks.

As far as specific approaches go, it is useful to recall that a very common structural optimization task aims to obtain the minimum possible mass without violating stress constraints. Since the material (and consequently its density) is very seldom considered as design variables, the optimizer has to impose changes to the geometric parameters and the area is chosen, most of the times, as the design variable.

In such a formulation, the objective function displays a linear, explicit relation with respect to the design variables. The same, however, does not hold true for the constraints, because stresses and areas relate with each other by means of a reciprocal mathematical function. This situation poses a special difficulty for the optimizer because the optimization problem is usually strongly driven by the constraints, which get directly represented as nonlinear entities with respect to the decision variables in this particular kind of formulation. Indeed, one would prefer, for computational efficiency reasons, the objective function to became nonlinear, and the constraints linear with respect to the design variables. This switch can be done if the design variables became the reciprocal of the areas, which is equivalent to integrate the stresses with respect to the areas, obtaining the internal forces (Vanderplaats [1998]). Hence, it is a usual procedure to replace stresses by internal forces in structural synthesis problems.

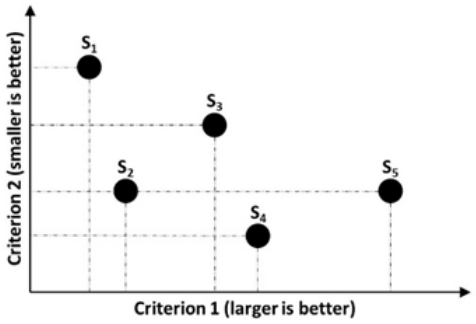
Conversely, in the case of structural dynamics problems, the corresponding computer cost mitigation approach consists of replacing the natural frequencies by the equivalent Rayleigh Coefficient (Canfield [1993]), with Equation (7) representing it as a dimensionless scalar that also relates the kinetic and elastic energies of the vibrating structure:

$$\lambda = \omega^2 = \frac{\phi^T \cdot k\phi}{\phi^T \cdot M\phi} = \frac{U}{T} \tag{7}$$

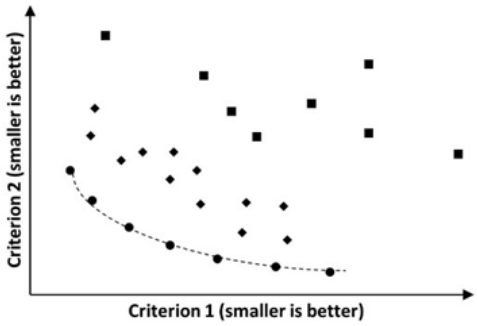
where K, M and ϕ are the stiffness, mass, and modal matrices, respectively. The scalars ω, λ, U and T stand for the natural frequency, its associated eigenvalue and the potential and kinetic energies.

3 Multi-criteria Optimization

Another noteworthy scenario with respect to possible combinations of the of $j+k+1$ functions defined in Equations (1)-(3) is that in which more than one response can participate within a set of objectives, configuring a multi-criteria optimization scenario (Gandibleux [2006], Odu and Charles-Owaba [2013]). The solutions for multi-criteria optimization problems are presented in terms of sets with multiple possible combinations of decision variables, each of them representing not strictly an optimal solution, in the absolute sense, but an optimal trade-off among the participating criteria. The solutions belonging to this set inherently group in a locus of the responses space, generally alluded to as the Pareto frontier (Wilson et al. [2001]), in a manifestation of a property called dominance or Pareto-optimality. By definition, a solution S1 to a multicriteria optimization problem dominates another solution S2 if S1 is no worse than S2 in all criteria and, furthermore, if S1 is better than S2 in at least one criterion. The concept of dominance in multicriteria decision-making can be graphically illustrated as in Figure 4. In panel (a), a two-dimensional response or output space (for 2 criteria), the best trade-offs would reside on the lower



(a) 2 criteria - larger is better AND smaller is better.



(b) 2 criteria - both smaller is better.

Figure 4: Schematic illustration of Pareto-dominance in the solution space for optimization problems with 2 criteria.

right extreme of the graph. However, driven by other constraints (explicit or not), there are cases such as S3 relative to S2 and S5 relative to S4 where Criterion 1 improves only at the expense of Criterion 2. In terms of dominance, both S1 and S5 dominate S2, whereas neither S2 nor S3 dominate any of the other solutions shown explicitly. Conversely, panel (b), shows the case of a two-dimensional output space where the 2 involved responses have to be minimized, and compete with each other. Even though the illustrations cover only two responses, for simplicity and clarity, the concept is valid for whatever arbitrary number of multiple criteria.

In the intersection between structural and multi-criteria optimization, a common situation arises when a certain region is discretized through some number of finite elements and, each of them having their own individual stress level, a single stress constraint is applied over the entire region. More generally, this situation is representative of the case in which a vector of responses that are connected/correlated to each other are subjected to the same practical constraint. In this scenario, a simple yet ineffective workaround would be to impose the constraint over the maximum value in the vector, that is, the maximum stress in this particular example, in the vein of being conservative. Because $\max(\cdot)$ is a highly discontinuous function, numerical difficulties are to be expected, therefore the alternative presented in Equations (8) and (9) is recommended (Butkewitsch and Steffen, Jr. [1999]):

$$\min(\beta),\ 0 \leq \beta \leq 1 \tag{8}$$

subject to

$$G(T, \beta) = \beta - \frac{T}{T'} \geq 0 \tag{9}$$

For numerical conditioning, the objective defined by the auxiliary variable β is normalized in the unit interval, and its value is driven down subjected to the constraint that it still remains positive while the values in the true target T (the components of the vector constraint), normalized by T' are subtracted from it. These conditions are only satisfied as the values of T itself are reduced, as initially intended.

4 Multidisciplinary Design Optimization

As described in Gandomi et al. [2013], multidisciplinary design optimization (MDO) has been an established field for as long as optimization migrated from a purely mathematical to a fully applied science. It gained significant momentum from the mid-1990s, as asserted by references Ragon et al. [2003], Giunta et al. [1997], Alexandrov and Kodiyalam [1998], Venter and Haftka [1999], mainly impelled by the challenge and opportunity of performing optimal decision making, in the mathematical sense, as in the present material, however targeting applications in aircraft design, that could leverage ever increasing computer processing power. Given that aircraft are highly complex and integrated systems themselves, a natural derivation merged MDO and Systems Engineering into MSDO (Multidisciplinary Systems Engineering), in which the mathematical framework to solve MDO problems has been applied to systems (References Neufville et al. [2004], Gu et al. [2000], Agarwal and Renaud [2004]).

While settling itself as MDO and evolving into MSDO, this entire knowledge field has consolidated a number of approaches to handle problems with competing domains, as described in the sequence.

4.1 All-at-Once (AAO)

For being the most straightforward, All-at-once is also the most usual approach for optimizing complex systems. It should be noted, however, that the clear exchange here is to gain simplicity potentially giving up some performance, since a single optimizer is in charge of determining all of the decision variables, driving all of the existing objectives and constraints alike. It is assumed, of course, that some representation of the system can supply the output values (either exact or approximate), given a set of inputs. Many options of simulation software exist for the increasingly preferred computational representation of such input to output relationships (References Altiok and Melamed [2010], Sturrock and Pegden [2011], Waller [2012], Ucar et al. [2017]). The AAO construct is depicted in Figure 5.

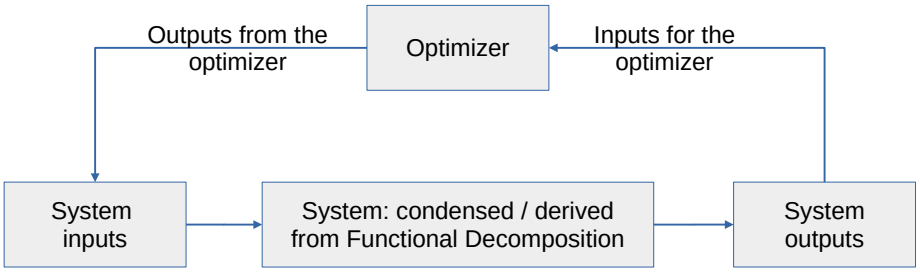


Figure 5: All-at-Once (AAO) approach for Multidisciplinary Systems Design Optimization (MSDO).

4.2 Individual Discipline Feasibility (IDF)

Contrary to All-at-Once, there is no overarching system representation invoked every time the optimizer iterates. Each sub-system is optimized in isolation (although the local mechanism is the same as that applied to the entire system within the AAO approach) and, at the sub-system level, each optimum is, by definition, feasible (or locally feasible, to be strict). The system-level optimizer has then to tune the decision variables if coupling constraints across the multiple-subsystems are violated in any form. These adjustments will penalize the local optima at the sub-systems in exchange of overarching feasibility. The notion of giving-up some optimality is intrinsic to partitioning the system in levels so that, in contrast with the All-at-Once approach, optimality is sacrificed for performance, since sub-systems can be optimized independently, and certainly in parallel. A potential drawback is the possibly unlimited complexity in how to define boundaries between sub-systems. Figure 6 captures the whole IDF idea.

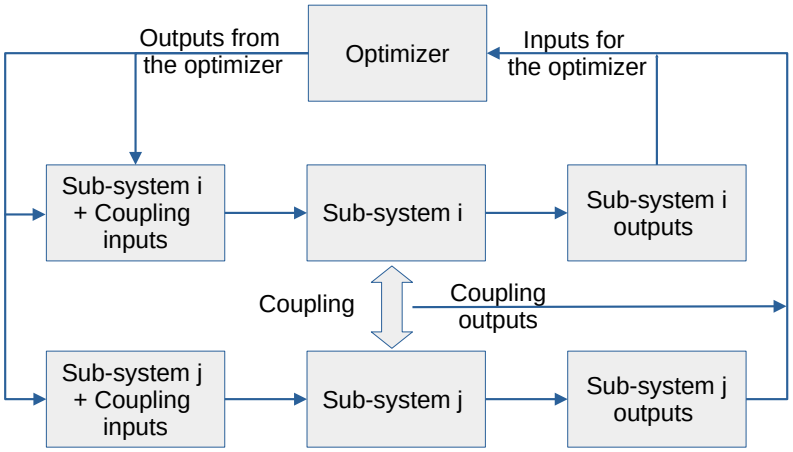


Figure 6: Individual Discipline Feasibility (IDF) approach for Multidisciplinary Systems Design Optimization (MSDO).

4.3 Collaborative Optimization (CO)

With Collaborative Optimization, the notion of a hierarchical approach is again leveraged, with decision variables at 3 levels: the system level, the coupling (inter) level and the sub-system (intra) level. It is hence similar to the IDF approach, but with the additional sophistication of decision variables preserved at the system level. The larger this system-level subset is, the stronger is the effort to preserve optimality despite the system partitions necessary to address local versus global aspects of the optimization process, as indicated in Figure 7.

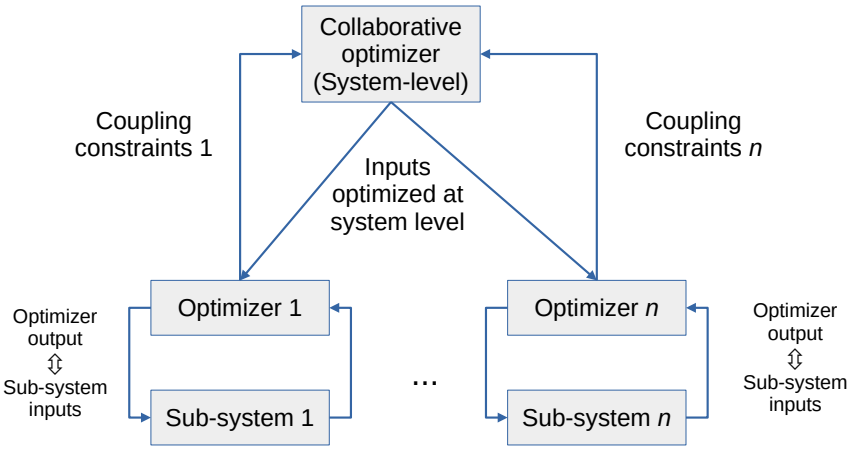


Figure 7: Collaborative Optimization (CO) approach for Multidisciplinary Systems Design Optimization (MSDO).

4.4 Bi-level Integrated System Synthesis (BLISS)

Due to leveraging features/strengths of the many the previous approaches, the BLISS method is somewhat popular and works as follows:

1. Define a starting configuration
2. Perform an entire system analysis, as in each loop of the AAO approach, to check both optimality and feasibility
3. If overall feasibility exists (no violated constraints at any sub-system) and optimality is satisfactory, exit
4. If step 3 above is not satisfied, perform sensitivity analysis at each sub-system in preparation to optimize them individually, recalling that sensitivities are objective measurements of rates of change caused in outputs, given variations on the inputs, in a small enough range or neighborhood (i.e., in the local sense, as opposed to global sensitivity analysis such as discussed in Rashedi et al. [2018])
5. Repeat step 4, but at the system level (i.e., consider sensitivity of the couplings relative to inputs)
6. Optimization at the sub-system level (leveraging step 4)
7. Optimization at the system level (leveraging step 5)
8. Returns to step 3 to either terminate the process or launch the next iteration, until convergence criteria are met.

It should be noted that in very complex systems, where it is justified to unfold a hierarchical tree into many levels, nested BLISS configurations give rise to multi-level system optimization. One could devise something along these lines to encompass from supply chain elements (at the higher end of the system spectrum), all the way

down to structural design of individual parts. Contemporary computer tools and system simulation software make it realistic to address such a use case at the time of writing (References Altiok and Melamed [2010], Sturrock and Pegden [2011], Waller [2012], Nardin et al. [2009]). Figure 8 depicts BLISS graphically.

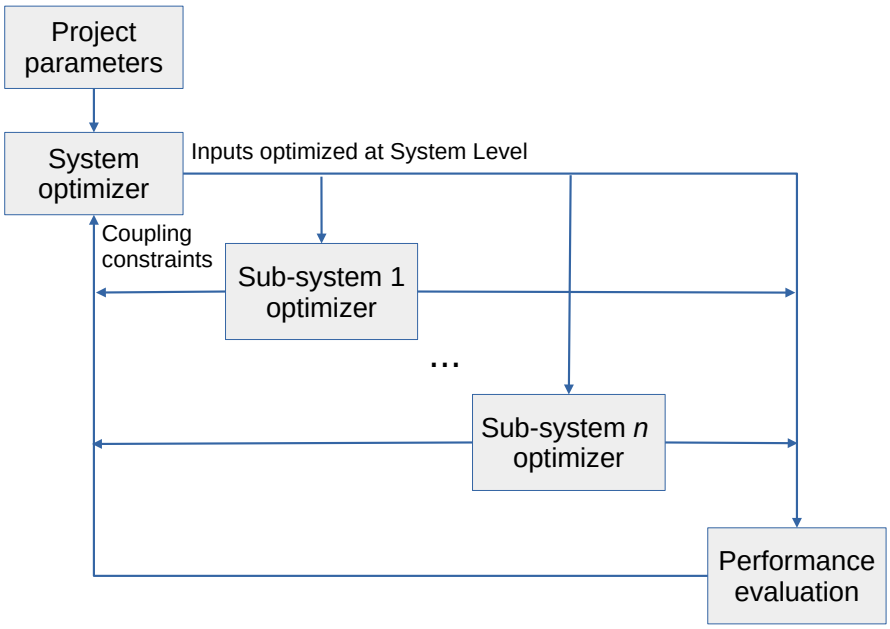


Figure 8: Bi-level Integrated Systems Synthesis (BLISS) approach for Multidisciplinary Systems Design Optimization (MSDO).

5 Solution Techniques

This section describes and discusses approaches to solve the optimization problem stated in its various forms, as per the preceding topics within this chapter. A number of computational implementations is offered to illustrate the main solution techniques, all of them presented as "Code" crafted in the R statistical computing platform R Core Team [2020].

For this purpose, the command `install.packages` is the main tool to add new packages to the R software. It takes a vector of names and a destination library, downloads the packages from the repositories (from a local folder or remote web server) and installs them in the R environment. To install the packages needed to run the codes in this chapter, for example, enter the following `install.packages('Rglpk', 'nloptr')`

Help is available at the command prompt via `?install.packages` command.

5.1 Classical Optimization Techniques

Classical, calculus-based or gradient-based optimization techniques will employ a collection of methods which, for the most part, operate by the use of derivatives to map a decision space between starting point(s) and the optimum, navigating from

the former to the latter in an iterative fashion. Regardless of how specifically these iterations are undertaken by each specific method, the process is deemed convergent into an optimal configuration $\{X^*\}$ upon satisfaction of the Karush-Kuhn-Tucker conditions, as in Equations (10)-(12):

$$G_j(\{X^*\}) \leq 0 \tag{10}$$

$$\lambda_j \cdot G_j(\{X^*\}) = 0 \tag{11}$$

$$\nabla F(\{X^*\}) + \sum_j \lambda_j \cdot \nabla G_j(\{X^*\}) + \sum_k \lambda_k \cdot \nabla H_k(\{X^*\}) = 0 \tag{12}$$

The first of the three conditions states that a solution must be feasible (i.e., satisfies all the constraints) to be considered optimal. Second and third conditions introduce the Lagrange Multipliers λ, which scale the gradient vectors (∇) for both equality and inequality constraints such that, when they are added to the gradient vector of the objective function itself, the resultant is zero at the optimum point $\{X^*\}$. This reflects an equilibrium condition, mathematically expressed by the solution of a homogeneous system of linear equations in $\{X\}$, tackled by the methods described in the remainder of this chapter.

6 Linear Programming

Linear programming consists in a class of problems from mathematical programming which the objective index and constraints are described by linear equations. Hence, they are particularly kin to structural optimization problems that can be linearized and, more specifically, when the decision space is discrete, such as when selecting structural members from a pre-defined catalog. Despite the great variety of problems and mathematical formulations encompassing the category, the linear program can be written in the standard form (Noble and Daniel [1988])

$$\max c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \tag{13}$$

subject to

$$a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n = b_1 \tag{14}$$

$$a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n = b_2 \tag{15}$$

$$\cdots \tag{16}$$

$$a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = b_m \tag{17}$$

$$x_i \geq 0, \ \forall i = 1, \cdots, n \tag{18}$$

where $c \in \mathbb{R}^{n \times 1}, a \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^{m \times 1}$ are fixed real values and $x \in \mathbb{R}^{n \times 1}$ are real numbers. The variables $x_i, \ i = 1, \cdots, n$ are the design variables.

In a compact vector notation the standard problem becomes:

$$\max \{C\}^T \{X\} \tag{19}$$

subject to

$$A\{X\} = \{B\} \tag{20}$$

$$\{X\} \geq 0 \tag{21}$$

With Equations (19) and (20) reinforcing the linear nature of the entire formulation through the dot product (linear combination) and system of simultaneous linear equations, respectively. Given $B \in \mathbb{R}^{m\times m}, m < n$ a set of m linearly independent columns of matrix A in Equation (20), the matrix is non-singular and therefore admits a unique solution for the expression:

$$B\{X_B\} = \{B\}. \tag{22}$$

If all $n-m$ components of $\{X\}$ not associated with columns of a matrix B are zero, the corresponding solution is a basic solution of Equation (22) with respect to B, which is mathematically equivalent to the second condition in Equation (11). The components of the design variable $\{X\}$ associated with columns of B are the basic variables. A vector $\{X\}$ is feasible if it satisfies equations (20) and (21). If a feasible solution is also a basic solution, it is said to be a basic feasible solution.

A fundamental theoretical result states that if there is a feasible solution to the linear program then there is a basic feasible solution. Furthermore, if there is an optimal feasible solution, there is an optimal basic feasible solution. This result is important because it converts the task of solving a linear program to a task of computing basic feasible solutions. For a problem represented by n variables and m constraints there are

$$\frac{n!}{m!(n-m)!} \tag{23}$$

or less basic solutions. This is an upper limit on the number of solution candidates for the linear program. There are several algorithms in the literature dealing with the task of finding an optimal design. The main idea behind a popular method "The Simplex Method" is outlined in the following.

6.1 The SIMPLEX method

Given the general linear programming problem expressed in Equations (19)-(21), where $\{X\} \in \mathbb{R}^{n\times 1}, \{C\} \in \mathbb{R}^{n\times 1}, A \in \mathbb{R}^{m\times n}$ and $\{B\} \in \mathbb{R}^{m\times 1}$, the optimal solution is also a basic feasible solution. Solving such problem requires the addition of slack variables $x_{n+1}, \cdots, x_{n+m}$ to convert the inequality constraint given by Equation (20) in equality constraint. As a result, the extended version of the general linear programming problem is given by

$$\max M = \{C_e\}^T\{X_e\} + u \tag{24}$$

subject to

$$A_e\{X_e\} = \{B\} \tag{25}$$

$$\{X_e\} \geq 0 \tag{26}$$

where $u \in \mathbb{R}, \{X_e\} \in \mathbb{R}^{q\times 1}, \{C_e\} \in \mathbb{R}^{q\times 1}, A_e \in \mathbb{R}^{m\times q}$ and $\{B\} \in \mathbb{R}^{m\times 1}$. Adopting this representation as a general linear programming formulation, it follows that $A_e = [A, I_m]\,, \{C_e\}^T = \left[\{C\}^T, \{0\}^T\right], q = m + n$ and $u = 0$ express the equivalence between Equations (19)-(21) and Equations (24)-(26).

The first step of the Simplex Method consists in the definition of the matrix

$$T = \begin{bmatrix} A_e & \{B\} \\ -\{C_e\}^T & u \end{bmatrix} \tag{27}$$

and then a number of elementary operations are performed over the lines of the matrix. The only allowed operation over the last line of T is the addition of linear combination of the lines above. As a result, this new matrix is obtained

$$T' = \begin{bmatrix} A'_e & \{B'\} \\ -\{C'_e\}^T & u' \end{bmatrix} \tag{28}$$

and the following linear programming problem is equivalent to the problem given by Equations (24)-(26). Along the Simplex Method a sequence of matrix operations is performed, according to the matrix representation of T, Equation (28), which elements where updated by transformations over the lines. For each T, a basic vector of viable solution exists. Suppose that at the step r, a feasible basic solution is found and the basic variables are $x_1, x_2, \cdots, x_m$. Furthermore, if the columns of T_r corresponding to the basic variables are unitary vectors, then T_r is expressed as

$$T_r = \begin{bmatrix} 1 & \cdots & 0 & a_{1,m+1} & \cdots & a_{1,q} & b_1 \\ 0 & 1 & 0 & a_{2,m+1} & \cdots & a_{2,q} & b_2 \\ 0 & \cdots & 1 & a_{m,m+1} & \cdots & a_{m,q} & b_m \\ 0 & \cdots & 0 & -c_{m+1} & \cdots & -c_q & u_r \end{bmatrix}. \tag{29}$$

In addition, the objective now is written as

$$M = u_r + c_{m+1}x_{m+1} + \cdots + c_q x_q \tag{30}$$

Equation (30) holds regardless of the values of $x_1, x_2, \cdots, x_q$. The basic solution available at this stage is found by using $x_{m+1} = x_{m+2} = \cdots = x_q = 0$ and the maximum value is $M = u_r$. At this stage, if all $c_j < 0$, for all $j = m+1, \cdots, q$, any increase in the values of $x_{m+1}, \cdots, x_q$ will lead to a decrease in the objective index M. This test is an indicative of the optimality of the solution.

On the other hand, if there is at least one $c_j > 0$ for $j = s$, the value of the objective index M will be increased by updating the value of $x_s \geq 0$. In this case, a new step is performed and the matrix T is updated. As a result, the numerical procedure is summarized as follows:

1. Find the column index $s \in \{m+1, m+2, \cdots, q\}$ at the last line of the matrix T whose element $-c_s$ has the lowest value.

2. Define $k = \frac{b_i}{a_{i,s}}$ with the smallest value of $\frac{b_i}{a_{i,s}}$ considering cases in which $a_{i,s} > 0$ holds.

3. Make a matrix pivoting at elements $a_{i,s}$ using the k value calculated above.

The succession of steps described above will lead to a sequence of feasible results that will maximize the objective index M. Since the number of basic solutions is finite, this procedure may lead to the optimal design.

Degeneracy occurs when a basic feasible solution contains a smaller number of non-zero variables than the number of independent constraints. In this case the values of some basic variables are zero and the replacement ratio is the same. Cycling in the algorithm will occur when the next basic solution to be investigated is one that was already investigated before. More advanced schemes are proposed to deal with the effects of degeneracy and cycling in linear programming. A comprehensive discussion about the Simplex Method and typical variants can be found in Luenberger and Ye [2008] and Vanderplaats [1998].

6.2 Duality and Sensitivity

Once again alluding to Equations (19)-(21) for a linear program, and considering Y^* to be its optimal solution, the maximum objective value of this formulation is also the minimal objective value of the following, rewritten linear program:

$$\min\{B\}^T\{Y\} \tag{31}$$

$$A^T\{Y\} \geq \{C\} \tag{32}$$

$$\{Y\} \geq 0 \tag{33}$$

and Y^* is also a solution for this problem. The linear program (13)-(18) is the so called primal problem and the new linear program (31)-(33) is the dual problem. There are a number of theoretical and computational properties shared by the formulations:

- The dual problem of a given dual problem is the primal.
- If $\{X\}$ satisfies the primal constraints (20)-(22) and {Y} satisfies the dual constraints (30)-(31) then $\{C\}^T\{X\} \leq \{B\}^T\{Y\}$.
- If $\{X\}$ satisfies the primal constraints (20)-(21), $\{Y\}$ satisfies the dual constraints (32)-(33) and $\{C\}^T\{X\} = \{B\}^T\{Y\}$ then $\{X\}$ is an optimal solution of the primal problem and $\{Y\}$ is an optimal solution of the dual problem.
- If exists non degenerated primal and dual feasible vectors then both primal and dual programs have optimal design solutions and the optimal objective values are identical.
- If the optimal program or the dual program has no feasible vector then both programs have no optimal solution.

These results allow a direct comparison of primal and dual feasible vectors and can be useful in evaluating the optimality of a solution. The results can also be used to convert a formulation with a large number of constraints into variables, that is, if $\{A \in \mathbb{R}^{m\times n} | m \gg n\}$ then the dual problem will be computed through the constraint matrix $\{A^T \in \mathbb{R}^{n\times m} | m \ll n\}$ and the number of matrix operations over the constraints is largely reduced. This scheme is sometimes required due to the concern with computational efficiency in problems with large number of constraints.

Code 1: Linear Programming method.

```
# maximize: 4 x_1 + 3 x_2 + 2 x_3
# subject to: 3 x_1 + x_2 + 2 x_3 <= 80
# 2 x_1 + 4 x_2 + 2 x_3 <= 70
# x_1 + 3 x_2 + 2 x_3 <= 70
# x_1, x_2, x_3 >= 0

library(Rglpk)

obj = c(4, 3, 2)
mat = matrix(c(3,2,1,1,4,3,2,2,2), nrow=3, ncol=3)
sig = c("<=", "<=", "<=")
rhs = c(80, 70, 70)
Rglpk_solve_LP(obj, mat, sig, rhs, max = TRUE)

$optimum
[1] 115

$solution
[1] 25 5 0
```

Another important concept refers to the sensitivity analysis of an optimal solution. Given a basic optimal solution $(\{X_B\}, 0)$ of the linear program (24)-(26) associated to an optimal basis B, where $\{X_B\} = B^{-1}B$, the solution of the corresponding dual formulation is $\{\lambda\}^T = \{C_B\}^T B^{-1}$.

If there is no degeneracy in the solution, the basis B is also optimal when small changes in the bounds $\{B\}$ of the constraint (Equation (25)) occur. Thus, for $\{B\} + \{\Delta B\}$ the optimal solution is

$$\{X\} = (\{X_B\} + \{\Delta X_B\}, 0) \tag{34}$$

where $\{\Delta X_B\} = B^{-1}\{DeltaB\}$. The corresponding change in the objective function value (Equation (30)) is

$$\Delta M = \{C_B\}^T \{\Delta X_B\} \tag{35}$$

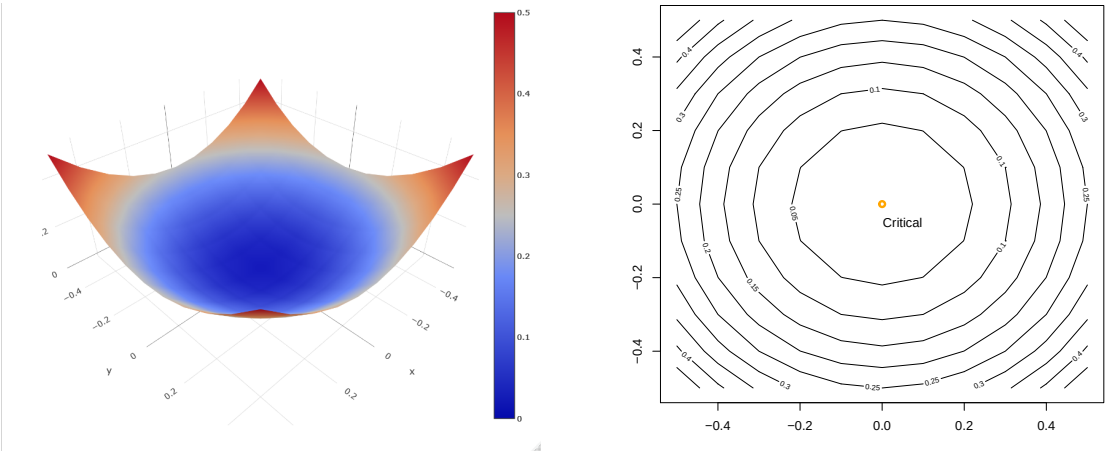
This expression addresses the sensitivity of the optimal value when small changes in $\{B\}$ are found. It can be understood as a marginal price of the component $\{B\}$, since when b_j changes to $b_j + \Delta b_j$, the value of the objective changes accordingly.

To summarize the Linear Programming approach, Code 1 implements it in the R statistical computing platform (Theussl and Hornik [2019]).

7 Non-Linear Programming

In the absence of linearization (either in the objective function and/or the constraints) or convexity within the decision space, the optimality conditions defined in Equations (10)-(12) should be resolved via non-linear optimization methods, to be detailed under each topic within this section.

The plot of $f(x, y) = x^2 + y^2$ is shown in Figure 9, along with a contour plot of the search space. The constrained case will be discussed later in section 8.



(a) Image of $f(x, y)$ (b) Contour plot of $f(x, y)$

Figure 9: Plot of a nonlinear function.

7.1 One-dimensional minimization methods

One-dimensional or line search is the mechanism taking place at each iteration of a non-linear optimization procedure, whereby the values of the decision variables in the current iteration (i) are updated from their values in the previous one $(i-1)$ according to a step of magnitude α taken along the direction delimited by $\{S\}$, a line or one-dimensional search along this vector's direction, as in Equation (36):

$$\{X\}^{(i)} = \{X\}^{(i-1)} - \alpha\{S\}^{(i)} \tag{36}$$

Since the previous status of $\{X\}$ is known, the line search boils down to determining both the search direction and the magnitude of the step along it. Many methods exist, and the broad categories with the most important ones are the object of the forthcoming bullet points. It should be noted that the choice of search direction should influence not only the one-dimensional line to be pursued for the current iteration but also, for the higher-order methods, how the successive lines of search are concatenated and, ultimately, how the overall landscape of the decision space is mapped. In regards to the step size, it plays the role of a learning rate, for which it is necessary to balance not too large a number that the procedure will

diverge/overshoot, neither an excessively small one that will unnecessarily penalize the performance.

The determination of the search direction $\{S\}$ is discussed next.

7.2 Zero-order methods

Methods in this category require function values only, but none of their derivatives, which could still be taxing depending upon the size and complexity of the model used for analysis of the objective function (for now, in an unconstrained scenario).

Usually at the expense of a very high number of function evaluations, these methods tend to be robust enough to circumvent issues related with non-convex and discontinuous functions, extensible to the case of discrete decision variables.

The primary representatives of this class are Random Search and Powell's method (described in greater detail in the sequence), while other methods are also documented in the literature (Vanderplaats [1998]).

7.2.1 Random Search

Practical implementations of Random Search depart from limiting the overall scope (and computational burden) by stipulating bounds as in Equation (4), and then modify Equation (36) accordingly:

$$\{X\}^{(i)} = \{X\}^{(i-1)} + r\left[\{X\}^{UB} - \{X\}^{LB}\right] \tag{37}$$

where r is randomly drawn from the $[0, 1]$ interval.

The important balance to be established relates to the width of the interval between the side constraints $\{X\}^{LB}$ and $\{X\}^{UB}$: make it too wide, and the convergence becomes too expensive, whereas excessive narrowing may lead to missing the global optimum for more complex decision spaces. Ultimately, these factors may be balanced by way of the heuristic search methods detailed in the next chapter.

It should be noted that in Equation (37), the search direction $\{S\}$ is replaced by a randomly chosen point within the side constraints, obtained by the subtraction of the bounds directly, which gets multiplied by the aleatory scale factor r. Hence, the direction itself is fixed, as it arises from $\{X\}^{LB}$ and $\{X\}^{UB}$ that remain constant. A more efficient variation affects each component of $\{S\}$ separately and, for an n-dimensional search space, works as:

$$\{S\}^{(i)} = 2 \cdot [r_i - 0.5],\ i = 1, \cdots, n. \tag{38}$$

7.2.2 Powell's method

This approach relies theoretically on the concept of conjugate directions, which underpins most of the more powerful classical optimization methods, and can yield significant success in practice if the problem to be solved is or can be reasonably approximated as a quadratic one, which is a plausible scenario for structural optimization (via Taylor series, for example).

Code 2: The BOBYQA algorithm.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}

x0 = c(-2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = nloptr(x0=x0, eval_f = fobj, lb = xl, ub = xu,
  opts=list(algorithm='NLOPT_LN_BOBYQA', 'maxeval' = 500))
cat('Optimal value: ', opt1$objective,
  '\nOptimal design: ', opt1$solution)

Optimal value: 4.334152e-05
Optimal design: 1.001892 1.004418
```

Denoting two directions (p) and (q), not necessarily in sequence but indexed as such for notational simplicity, are conjugate if

$$\left(\{S\}^{(p)}\right)^T \cdot [H] \cdot \{S\}^{(q)} = 0. \tag{39}$$

The first direction is transposed to ensure dimensional compatibility with respect to the Hessian square matrix of order n, denoted as H, and holding the second order derivatives of the objective function.

According to Powell's method, a first set of searches following Equation (36) is performed across the n orthogonal directions that constitute the coordinates of the search space, finding each minimum associated with a corresponding α. While neither of these directions are necessarily conjugate, they provide the starting point for building subsequent directions that satisfy this condition. Having completed all n uni-dimensional searches, each of them becomes a column of the following approximation to the Hessian matrix:

$$[\hat{H}] = \left[\alpha_1 \cdot S^{(1)}, \alpha_2 \cdot S^{(2)}, \cdots, \alpha_n \cdot S^{(n)}\right] \tag{40}$$

which is then used as a generator for conjugate directions by summation of the columns of $[H]$, as follows:

$$S^{(i+1)} = \sum_n \alpha_i \cdot S^{(i)} \tag{41}$$

The method is repeated thereafter updating the iteration index, disposing of the leftmost column of $[H]$, shifting all remaining columns leftwards and appending the most recent conjugate direction into the (now empty) rightmost column. The flowchart in Figure (10) displays the entire method.

An advanced method using this concept is the BOBYQA (Powell [2009]), which usage is shown in Code 2 (Johnson [2020]).

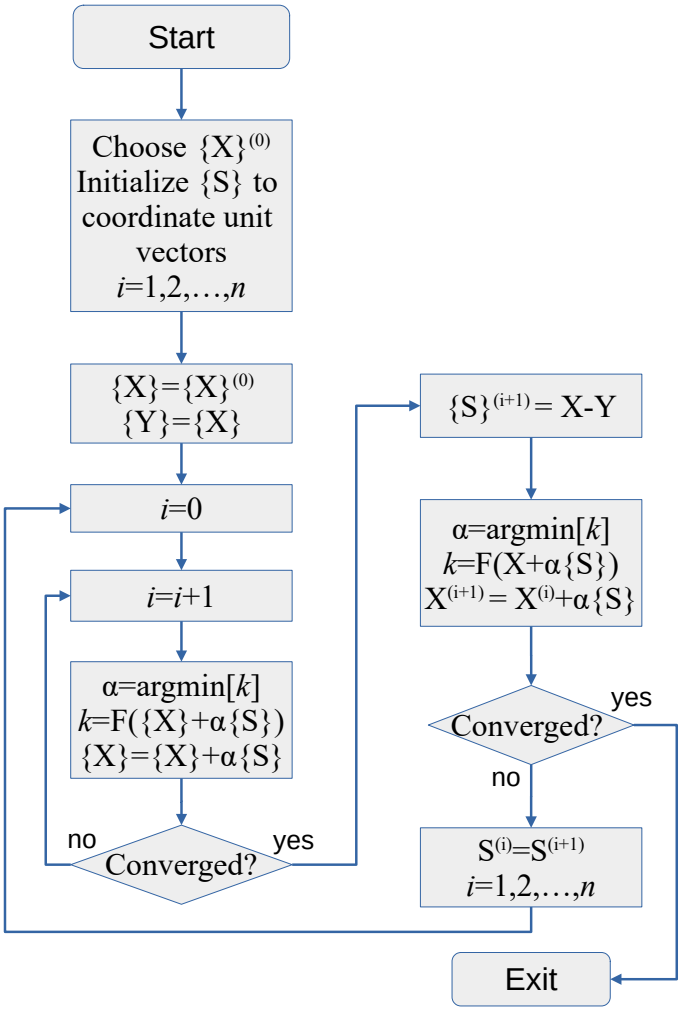


Figure 10: Flowchart with step-by-step description of Powell's zero-order optimization method.

7.3 First-order methods

The foremost first-order method is the Steepest Descent algorithm, which leverages the property of directional derivatives whereby the path of most intense variation of a function $F\{X\}$ at point $\{X^0\}$ is that of the gradient of the function at the same point. In essence, it is as simple as replacing $\{S\}$ by $\nabla F(\{X\})$ at each iteration, until Equations (10)-(12) are reasonably satisfied.

It should be noted however that while the instantaneous variation is the steepest by following the direction of the gradient to the function at each point, the overall performance of the method is severely penalized by the fact that the gradient is always perpendicular to the function surface. Along the successive iterations, this results in a series of search directions in which the current one is perpendicular to its prior. In effect, this means that each search direction retains no information whatsoever about the previous one (by orthogonality), and hence the decision landscape is poorly mapped.

To counter this loss of information that reduces the efficiency of the search, the

Conjugate Direction method (also known as the Fletcher-Reeves method) requires only a simple modification of the Steepest Descent approach, yet significantly improves the convergence rate. In effect, its initial search direction is the gradient of $F\{X\}$, to be updated according to Equations (42) and (43):

$$S^{(i)} = \nabla F(\{X\})^{(i-1)} + \beta^{(i)} \cdot S^{(i-1)} \tag{42}$$

$$\beta^{(i)} = \frac{\left|\nabla F(\{X\})^{(i-1)}\right|^2}{\left|\nabla F(\{X\})^{(i-2)}\right|^2} \tag{43}$$

This approach is conceptually similar to Powell's method, in the sense that the initial search directions are meant only to initialize the procedure, but different from the perspective that all of the search directions are conjugate from the beginning. The flowchart is in Figure 11.

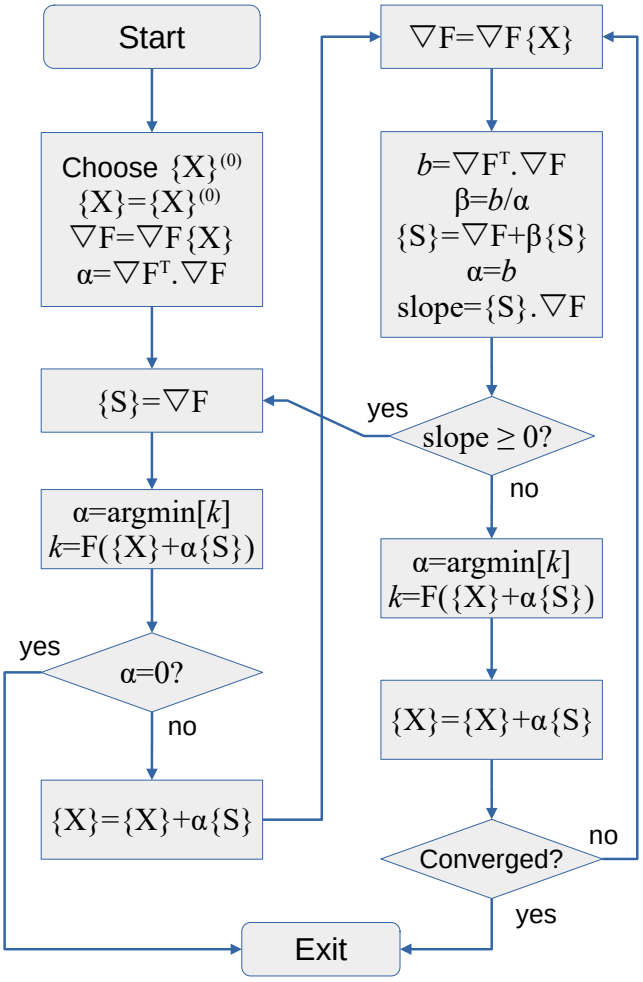


Figure 11: Flowchart with step-by-step description of Fletcher-Reeves Conjugate Direction method.

7.4 Second-order methods and approximations thereof

At this point, it becomes noticeable that adequate mapping of the decision landscape (i.e., topology of the objective function $\{F\}$ and how it varies in n-dimensional space) bears beneficial influence in terms of how efficient the optimization process occurs. On this vein, mapping the second derivatives will not only entail the information about the instantaneous reward (as in each isolated iteration of the Steepest Descent method), but also provide some anticipation on how the decision space varies thereafter, connecting a more plausible succession of search directions other than the sequentially perpendicular ones. As seen in Equation (41), the Hessian matrix $[H]$ is the container to store all information relative to the second-order derivatives, as in Equation (44):

$$[H(F(\{X\}))] = \begin{bmatrix} \frac{\partial^2 F(\{X\})}{(\partial X_1)^2} & \frac{\partial^2 F(\{X\})}{(\partial X_1)\cdot(\partial X_2)} & \cdots & \frac{\partial^2 F(\{X\})}{(\partial X_1)\cdot(\partial X_n)} \\ \frac{\partial^2 F(\{X\})}{(\partial X_2)\cdot(\partial X_1)} & \frac{\partial^2 F(\{X\})}{(\partial X_2)^2} & \cdots & \frac{\partial^2 F(\{X\})}{(\partial X_2)\cdot(\partial X_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\{X\})}{(\partial X_n)\cdot(\partial X_1)} & \frac{\partial^2 F(\{X\})}{(\partial X_n)\cdot(\partial X_2)} & \cdots & \frac{\partial^2 F(\{X\})}{(\partial X_n)^2} \end{bmatrix} \tag{44}$$

which is square of order n and symmetric with respect to the main diagonal, where the second-order derivatives are with respect to each individual decision variable in $\{X\}$, with cross terms in the off-diagonal cells.

The paramount approach for utilizing $[H]$ as part of the optimization procedure is Newton's method, which starts with the Taylor series expansion in Equation (45), whose notation differs from that in Equation (5) to embed the notion of search iterations indexed by (i):

$$\begin{aligned} F(\{X\}) &= F(\{X\}^{(i)}) + \nabla F(\{X\}^{(i)})^T \cdot \delta(\{X\}) \\ &\quad + \frac{1}{2}\delta(\{X\})^T \cdot \left[H(F(\{X\})^{(i)})\right] \cdot \delta(\{X\}) \end{aligned} \tag{45}$$

where

$$\delta(\{X\}) = \{X\}^{(i+1)} - \{X\}^{(i)}. \tag{46}$$

Solving Equation (45) for the stationary condition gives:

$$\delta(\{X\}) = -\left[H(F(\{X\})^{(i)})\right]^{-1} \cdot \nabla F(\{X\}^{(i)}). \tag{47}$$

Next, Equation (48) can be obtained by combining a re-arranged form of Equation (46) with Equation (47), and further simplified into Equation (45)

$$\{X\}^{(i+1)} = \{X\}^{(i)} + \delta(\{X\}) = \{X\}^{(i)} - \left[H(F(\{X\})^{(i)})\right]^{-1} \cdot \nabla F(\{X\}^{(i)}) \tag{48}$$

$$\{S\}^{(i)} = -\left[H(F(\{X\})^{(i)})\right]^{-1} \cdot \nabla F(\{X\}^{(i)}) \tag{49}$$

Complementary, Equation (49) can be derived from Newton's (also known as Newton-Raphson) method for determining roots of equations. In the present case, the goal

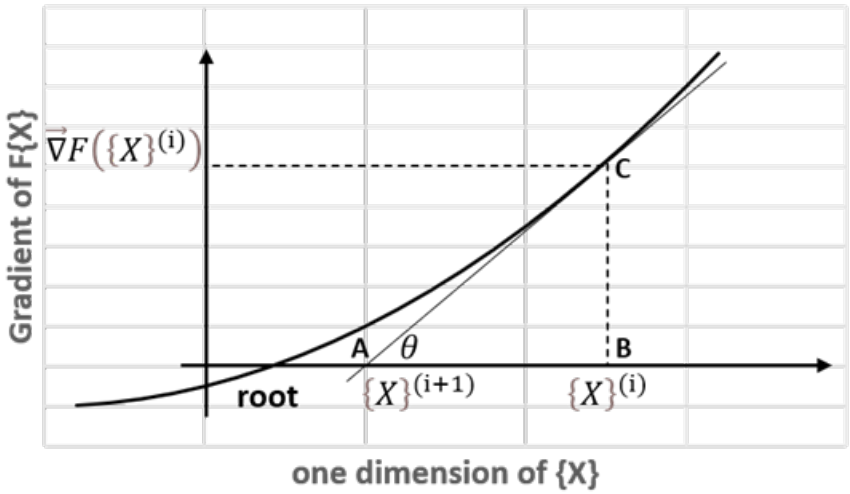


Figure 12: Newton-Raphson root finding approach to derive Newton's second-order search method.

is determining the roots of the equation corresponding to $\nabla F(\{X\}) = 0$, which indicates the stationary point of a function of n variables contained in $\{X\}$. Figure 12 describes the root finding procedure, projected to a single dimension, and how it circles back to Equations (47)-(49).

Starting at point $\{X\}^(i)$, a tangent (derivative or gradient component) to the function is drawn until it intercepts the horizontal axis, much closer to the root at $\{X\}^{(i+1)}$. It is possible to see that iterating a few more times from this new point (corresponding to the vertex A of the triangle ABC), convergence to the root will eventually occur. However, freezing into the current view and back to the triangle ABC, it is possible to see that the motion described in Equation (50) is:

$$\tan(\theta) = \left.\frac{d(\nabla F)}{d\{X\}}\right|_{\{X\}^(i)} = \frac{\nabla F(\{X\}^{(i)})}{\{X\}^{(i+1)} - \{X\}^{(i)}} \tag{50}$$

and given that

$$\{X\}^{(i+1)} - \{X\}^{(i)} = \frac{\nabla F(\{X\}^{(i)})}{\left.\frac{d(\nabla F)}{d\{X\}}\right|_{\{X\}^{(i)}}} \tag{51}$$

and

$$\left.\frac{d(\nabla F)}{d\{X\}}\right|_{\{X\}^{(i)}} = \frac{d^2F(\{X\})}{dx^2} \tag{52}$$

Equations (51) and (52) can be combined as in Equation (53) and, when generalizing the second derivative back into its matrix form, confirm Equation (47):

$$\{X\}^{(i+1)} - \{X\}^{(i)} = \frac{\nabla F(\{X\}^{(i)})}{\frac{d^2F(\{X\})}{dx^2}} \tag{53}$$

The computational counterpart of Equations (44)-(53) in R format is provided within Code 3 (Dembo and Steihaug [1983], Johnson [2020]):

Code 3: The Newton algorithm.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 +
  (1 - x[1])^2 )
}
gfobj <- function(x) {
  return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 *
  (1 - x[1]), 200 * (x[2] - x[1] * x[1])) )
}

x0 = c(2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = nloptr(x0=x0, eval_f = fobj, eval_grad_f = gfobj,
  lb = xl, ub = xu,
  opts=list('algorithm'='NLOPT_LD_TNEWTON',
  'xtol_rel' = 1.0e-6))
cat('Optimal value: ', opt1$objective,
  '\nOptimal design: ', opt1$solution)

Optimal value: 6.252339e-28
Optimal design: 1 1
```

Let's now make some considerations regarding the overhead for computing the second-order derivatives to fill the n-dimensional, symmetrical square matrix $[H]$. A total of $\frac{n(n-1)}{2}$ derivatives is to be calculated, resulting in computational complexity $O(N^2)$. Therefore, while the detailed mapping of the decision space to be navigated is useful, it has to be pondered against the calculation burden, and the trade-off is approximating $[H]$ instead of calculating it explicitly. Davidon-Fletcher-Power (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS), the paramount methods to implement this philosophy, are detailed in the sequence. Both of them share Fletcher and Reeves philosophy about storing information pertinent to previous iterations, but do that using a vector instead of the single scalar β within Equations (42) and (43), as well as Figure (11).

Along these lines, to approximate the Hessian matrix, these methods start by making it equal to the Identity Matrix, which is equivalent to start the search through the direction of Steepest descent (constant gradient across the main diagonal). Then, interactively, this approximate Hessian is updated as follows:

$$[H]^{(i+1)} = [H]^{(i)} + [D]^{(i)} \tag{54}$$

where

$$\begin{aligned}[D]^{(i)} &= \frac{\sigma+\theta}{\sigma^2}\cdot\delta(\{X\})\cdot\delta(\{X\})^T + \frac{\theta-1}{\tau}\cdot[H]^{(i)}\cdot y\cdot([H]^{(i)}\cdot y)^T \\ &\quad -\frac{\theta}{\sigma}\cdot\left[[H]^{(i)}\cdot y\cdot\delta(\{X\})^T + \delta(\{X\})\cdot([H]^{(i)}\cdot y)^T\right]\end{aligned} \tag{55}$$

Code 4: The BFGS method.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
gfobj <- function(x) {
  return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 *
  (1 - x[1]), 200 * (x[2] - x[1] * x[1])) )
}

x0 = c(-2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = nloptr(x0=x0, eval_f = fobj, eval_grad_f = gfobj,
  lb = xl, ub = xu,
  opts=list('algorithm'='NLOPT_LD_LBFGS',
  'xtol_rel' = 1.0e-6))
cat('Optimal value: ', opt1$objective,
'\nOptimal design: ', opt1$solution)

Optimal value: 1.243396e-18
Optimal design: 1 1
```

with the vector y being the difference between the gradients at two successive iterations, as in Equation (55), and the scalars σ and τ defined as in Equations (57) and (58), respectively:

$$y = \nabla F(\{X\}^{(i)}) + \nabla F(\{X\}^{\{}(i-1)\}) \tag{56}$$

$$\sigma = \delta(\{X\})^T \cdot y \tag{57}$$

$$\tau = y^T \cdot [H]^{(i)} \cdot y \tag{58}$$

DFP assumes $\theta = 0$ and BFGS uses $\theta = 1$, while other methods within this family, also known as quasi-Newton or variable metric (because of the nature of the function expressed in Equation (55)) have their own operating strategies. Code 4 provide an illustrative R implementation of the BFGS method (Liu and Nocedal [1989], Johnson [2020]).

While the emergence of high performance computers reduces the performance related difficulties of tackling full-fledged Hessian matrices, methods such as DFP and BFGS retain their relevance with respect to numeric stability, since they avoid inversion of matrices that often grow to very large sizes.

The determination of the step size/learning rate α is discussed next.

Once a search direction $\{S\}$ is determined, as discussed in the preceding item, it is then time to decide upon the magnitude of the motion along $\{S\}$, for which the following methods are mainstream.

7.4.1 Golden-section method

Consider the following logic to create a Fibonacci series which, apart from its two initial terms at 0 and 1, consists of progressively adding up the two latest terms to determine the next one:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2},\ i > 1 \end{aligned} \tag{59}$$

After stabilizing at its recursive portion for $i > 1$, the ratio between any two consecutive terms is equal to the Golden Section ratio ϕ, a constant calculated as in Equation (54):

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.61803 \tag{60}$$

The constant defined as such is useful for finding the optimum of a one-dimensional function (which is the essence of one-dimensional or line searches) when, similarly to the bisection method for root finding, one could interactively determine the minimum of a function bounded within a known interval. Specifically, if in the neighborhood of the minimum we can find three points $x_1 < x_2 < x_3$ with functional values $f(x_1) > f(x_2) > f(x_3)$, then the minimum is located between x_1 and x_2 (or exactly at x_2, at the limit).

Finding this minimum starts by picking a brand-new point x_4 located between x_2 and x_3, such that either the minimum is located between x_4 and x_3 (upwards) or between x_1 and x_4 (downwards). Regardless, a new interval containing 3 points, similar to but narrower than the previous one, is delimited. For efficiency, we prefer that the remaining interval to be interactively shrunk from the previous one is as small as possible, which is obtained by intervals of same width irrespective to which side the search pivots to. Equality of interval lengths using the above notation is enforced as in Equation (55):

$$(x_2 - x_1) + (x_4 - x_2) = (x_3 - x_2) \tag{61}$$

which gets rearranged into

$$(x_2 - x1)^2 + (x_2 - x_1) \cdot (x_3 - x_2) - (x_3 - x_2)^2 = 0 \tag{62}$$

because $(x_3 - x_2) = (x_3 - x_4) + (x_4 - x_2)$. Solving Equation (61) and choosing the positive of the two roots yields a ratio between $(x_2 - x_1)$ and $(x_3 - x_2)$ which is exactly the golden section ϕ of Equation (60). This leads to the following update law for the iterative process, to be deemed converged upon reaching a suitable, typically problem dependent tolerance:

$$x_1^{(i+1)} = x_2^{(i)} \tag{63}$$

$$x_2^{(i+1)} = x_3^{(i)} \tag{64}$$

$$x_3^{(i+1)} = x_3^{(i)} + \varphi \cdot \left(x_3^{(i)} - x_2^{(i)} \right) \tag{65}$$

7.4.2 Root finding/Interpolation methods (quadratic and cubic cases)

A unique parabola may be fitted through 3 points x_1, x_2 and x_3 discussed in the golden section method above, resulting in the functional form and optimum expressed in Equations (66) and (67):

$$a \cdot \alpha^2 + b \cdot \alpha + c \tag{66}$$

$$\alpha_{\text{optimum}} = \frac{-b}{2 \cdot a} \tag{67}$$

Given the functional values at x_1, x_2 and x_3 chosen for the golden search approach, it is guaranteed that $a > 0$ in Equation (66) and, therefore, the optimum α does correspond to a minimum. Moreover, computational efficiencies can be gained if one of the three points within the working interval is made to be 0 and/or they are evenly spaced.

In some cases, even narrowly bounded intervals will contain functional behavior that is highly non-linear, requiring higher order interpolation to be performed. Equation (68) shows the cubic case. Optimality is guaranteed by finding the roots of Equation (69), which is the quadratic form of obtained by deriving the cubic interpolate. Moreover, minimization is obtained when the second derivative, which is linear, is strictly positive as shown in Equation (70). Calculation of the coefficients a and b that satisfy this condition is possible by having four gradient/function evaluations or, alternatively, three gradient evaluations.

$$a \cdot \alpha^3 + b \cdot \alpha 2 + c \cdot \alpha + d \tag{68}$$

$$3 \cdot a \cdot \alpha^2 + 2 \cdot b \cdot \alpha + c = 0 \tag{69}$$

$$6 \cdot a \cdot \alpha + 2 \cdot b > 0 \tag{70}$$

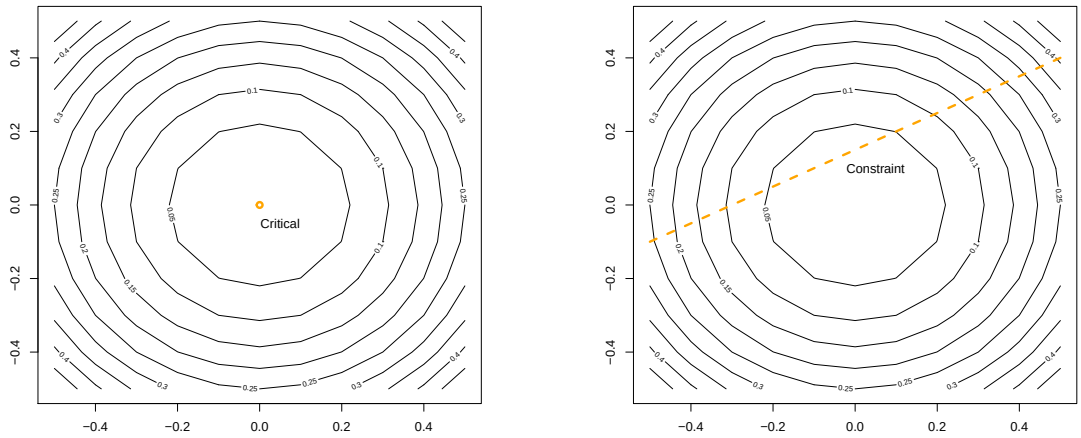
8 Constrained optimization techniques

Insofar, motion along the search direction $\{S\}$ at steps with length α did not account for encountering any constraint. While this is an adequate simplification for methodological purposes, not incorporating constraints prevents actual applicability of these methods, which will then be amended to represent the full extent of the optimization problem statement introduced in Equations (1)-(4). Hence, constrained optimization problems are solved by extensions of the well-established methods for the unconstrained scenarios.

A geometric representation of a constraint is shown in Figure 13. In the case of an equality constraint, the search space is reduced to the dashed line. In the case of an inequality constraint, the search space is a subset above or below the dashed line.

8.1 Sequential Methods

One of the strategies for leveraging unconstrained optimization methods into problems that actually do pose constraints is to penalize the search outcome to limit constraint violation.



(a) Unconstrained case (b) Constrained case

Figure 13: Contour plot of a search space.

While simple, this idea has side effects as it results in numerical ill-conditioning, due to discontinuous nature of the penalties, which behave like "jumps" during the search process. This obstacle may be circumvented by modulating the penalties with a weight factor r_p as in Equation (71), which is moderate at the earlier search stages and increases as the solution approaches the optimum, operating in a sequentially increasing penalization mode, yielding the name for this class of methods:

$$\phi(X, r_p) = F(\{X\}) + r_p \cdot P(\{X\}) \tag{71}$$

The pseudo-objective $\phi(X, r_p)$ results from penalizing the original $F(\{X\})$ with the variable penalty $P(\{X\})$. Once again, the explicit dependency of the penalty on the current value of the decision variables stresses its adaptive nature.

8.1.1 Exterior Penalty Method

A penalty function (Equation (72)) is built in a fashion that it is innocuous if no constraint is violated:

$$P(X) = \sum_{i=1}^{m} \left[\max(0, G_j(\{X\}))\right]^2 + \sum_{i=1}^{k} \left[H_k(\{X\})\right]^2 \tag{72}$$

accounting for both inequality and equality constraints in $G_j(\{X\})$ and $H_k(\{X\})$, respectively. Squaring the constraints ensures a slope of zero for the constraint function at its violation boundary, improving numerical conditioning of the pseudo-objective defined in Equation (56).

Remaining ill-conditioning issues can be resolved by balancing the choice of r_p. When it is too small, the search process is very stable but lax for allowing constraints violation. Conversely, large r_p values lead to strict observance of the constraints, but induce the "jumps" alluded to before. It is customary to modulate this process

by sequentially increasing the value of r_p by a constant factor γ, which is then fine-tuned for each problem. Figure 14 represents the Exterior Penalty method in graphical form.

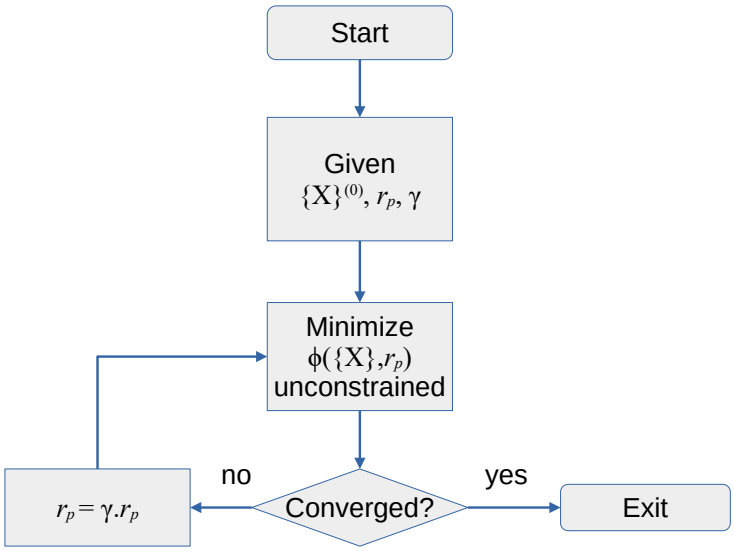


Figure 14: Flowchart for the Exterior Penalty method for constrained optimization.

8.1.2 Interior Penalty Method and Extension

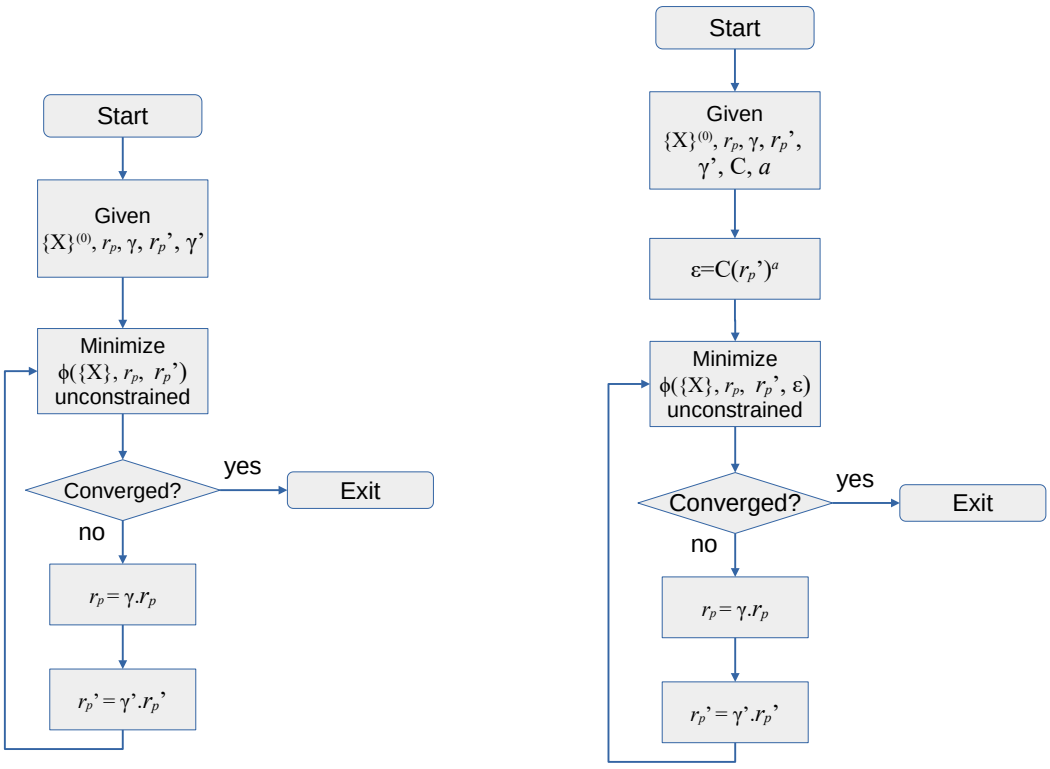
By providing a special treatment for inequality constraints, whereby the penalties are actually reduced instead of decreased over the search process (Equations (73)-(74)), this method aims at simplifying the unconstrained search portion of the solution:

$$\phi(X, r'_p, r_p) = r'_p \cdot \sum_{i=1}^{m} -\frac{1}{G_j(\{X\})} + r_p \cdot \sum_{i=1}^{k} [H_k(\{X\})]^2 \tag{73}$$

$$\phi(X, r'_p, r_p) = r'_p \cdot \sum_{i=1}^{m} -\log(-G_j(\{X\})) + r_p \cdot \sum_{i=1}^{k} [H_k(\{X\})]^2 \tag{74}$$

with both Equations (73) and (74) exchanging the discontinuous max(·) function present in Equation (72) for continuous ones, with variation "b" being reputed for slightly better numerical conditioning. Also, an additional weight factor r'_p is introduced to handle the inequality constraints separately from the equality ones. Over the course of the search, r'_p will decrease at a rate modulated by the constant parameter γ'.

The extended version of the method accounts for the specific degree of constraint violation relative to a tolerance level ε, a small negative quantity dependent upon constants C and a, that marks the transition between the interior penalty method to its extended version. Detailed formulation is described in Equations (75)-(78), and a flowchart of both interior penalty approaches is presented in Figure 15.



(a) Original method (b) Extended method

Figure 15: Interior penalty method for constrained optimization.

$$P(\{X\}) = \sum_{i=1}^{m} \bar{G}_j(\{X\}) \tag{75}$$

where

$$\bar{G}_j(\{X\}) = -\frac{1}{G_j(\{X\})} \quad \text{if } G_j(\{X\}) \leq \varepsilon \tag{76}$$

$$\bar{G}_j(\{X\}) = -\frac{2\varepsilon - G_j(\{X\})}{\varepsilon^2} \quad \text{if } G_j(\{X\}) > \varepsilon \tag{77}$$

and

$$\varepsilon = -C \cdot (r'_p)^a, \ \frac{1}{3} < a < \frac{1}{2}. \tag{78}$$

Closer examination of Equations (75)-(78) reveals that the penalty functions have continuous first derivatives at $G_j(\{X\}) = \varepsilon$, but the continuity of the second derivative is not assured unless the order of the penalty function is increased as in its quadratic form displayed at Equation (79):

$$\bar{G}_j(\{X\}) = -\frac{1}{G_j(\{X\})'} \quad \text{if } G_j(\{X\}) \leq \varepsilon \tag{79}$$

$$\bar{G}_j(\{X\}) = -\frac{1}{\varepsilon} \cdot \left[\left(\frac{G_j(\{X\})}{\varepsilon} \right)^2 - 3 \cdot \frac{G_j(\{X\})}{\varepsilon} + 3 \right] \quad \text{if } G_j(\{X\}) > \varepsilon \tag{80}$$

8.1.3 Additional Penalty Methods for improved numerical conditioning

Relatively recent developments are meant to ensure better balance between feasibility (the optimal solution respects the constraints) and numerical stability. Along these lines, a family of variable penalty functions is proposed as in Equations (81) to (89), and the logarithmic approach introduced in Equation (74) is improved as per Equations (90) to (91).

$$\bar{G}_j(\{X\}) = -\frac{-[G_j(\{X\})]^{1-s}}{s-1} \quad \text{if } G_j(\{X\}) \leq \varepsilon \tag{81}$$

and

$$\begin{aligned} \bar{G}_j(\{X\}) &= (-\varepsilon)^{1-s} \cdot \left[A \cdot \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right)^3 + \frac{1}{2} \cdot \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right)^2 \right. \\ &\quad \left. - \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right) + \frac{1}{s-1} \right] \quad \text{if } G_j(\{X\}) > \varepsilon \end{aligned} \tag{82}$$

which undergo the following modifications when $s = 1$:

$$\bar{G}_j(\{X\}) = -\log\left(-G_j(\{X\})\right) \quad \text{if } G_j(\{X\}) \leq \varepsilon \tag{83}$$

and

$$\begin{aligned} \bar{G}_j(\{X\}) &= A \cdot \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right)^3 + \frac{1}{2} \cdot \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right)^2 \\ &\quad - \left(\frac{G_j(\{X\})}{\varepsilon} - 1 \right) - \log(-\varepsilon) \quad \text{if } G_j(\{X\}) > \varepsilon. \end{aligned} \tag{84}$$

Both Equations (82) and (84) depend on parameters A and s, whose values and fine tuning contemplate literature recommendations as expressed in Equations (85) to (90), connecting with the notion of the constraint violation tolerance ε and the sequential penalty approaches, such as those contained in Equations (71) to (74):

$$\varepsilon = -\beta \cdot (r'_p)^q \tag{85}$$

where β is a positive constant chosen such that ε remains near zero at the start of the search, while q is recommended to be within the range in Equation (86)

$$\frac{1}{2+s} \leq q \leq \frac{1}{s}. \tag{86}$$

On its turn, A should be defined as in Equation (87)

$$A = \frac{1+s}{3} \quad \text{if } G_j(\{X\}) \leq 0 \tag{87}$$

and

$$A = \left[(1-s) \cdot \left(\frac{C^*}{\varepsilon - 1} \right) \right] \left[\frac{1}{3} \left(\frac{C^*}{\varepsilon - 1} \right)^{-2} \right] \quad \text{if } G_j(\{X\}) > 0 \tag{88}$$

indicating violated constraints, at a maximum C^*, and falling back into Equation (89) when all constraints are violated.

$$A = \frac{s}{6 \cdot \left(1 - \frac{C^*}{\varepsilon}\right)} \tag{89}$$

Regarding the logarithmic penalty function, also known as log-barrier function, its enhancement towards better numerical conditioning involves the modification in Equation (90). While it only explicitly addresses inequality constraints, the equality ones can be either handled by the standard exterior penalty method, or treated as two equal but opposite inequality constraints.

$$P(x, r'_p, \lambda^i) = r'_p \cdot \sum_{j=1}^{m} \lambda_j^i \cdot \log\left[1 - \frac{G_j(\{X\})}{r'_p}\right]. \tag{90}$$

Lastly, the Lagrange multipliers λ, recalled from the KKT convergence conditions in Equations (10)-(12), are updated along the i iterations as described in Equations (91)-(92).

$$\lambda_j^{i+1} = \frac{\lambda_j^i}{\left[1 - \frac{G_j(\{X\})}{r'_p}\right]} \quad \text{if } G_j(\{X\}) \leq k \cdot r'_p, \text{ with } k < 1 \tag{91}$$

$$\lambda_j^{i+1} = \frac{\lambda_j^i}{2r'_p(1-k)} \quad \text{if } G_j(\{X\}) > k \cdot r'_p, \text{ with } k < 1. \tag{92}$$

The next topic is dedicated to cover the role of Lagrange Multipliers in greater detail, including additional methods in which they have a more prominent role in solving constrained optimization problems.

8.1.4 Augmented Lagrange Multiplier Method

Adding a penalty term to the original objective function to account for the constraints continues to be the governing idea, but with direct influence from the KKT convergence condition of Equations (10)-(12). Historically, this approach was first derived for equality constraints (Equation (93)) and then generalized to include the inequality ones (Equation (99)). Both approaches are similar, as highlighted by the flowcharts in Figure 16.

$$A(X, \lambda, r_p) = F(\{X\}) + \sum_k 2\lambda_k H_k(\{X\}) + r_p[H_k(\{X\})]^2 \tag{93}$$

Observe that Equation (93) reduces to the exterior penalty method of Equation (72) if all λ_k are equal to 0. More interestingly, if the values of the Lagrange multipliers are chosen to be optimal, it is possible to prove that the true optimum of $F(\{X\})$ can be determining (for positive, but finite values of r_p), skipping the constrained portion of the method altogether and requiring a single unconstrained search. Since these optimum values of the Lagrange multipliers are actually unlikely to be available, it is possible to update them from an initial guess by using Equation (94):

$$\lambda_k^{i+1} = \lambda_k^i + 2r_p H_k(\{X\}^i) \tag{94}$$

Now, Equation (96) will provide the inequality constraints version of Equation (93), after the inequalities from Equation (2) are evened out into equalities, by adding the term Z^2 as in Equation (95). The value Z is squared due to the numerical conditioning ease of parabolic functions.

$$G_j(\{X\}) + Z^2 = 0 \tag{95}$$

$$A(X, \lambda, Z, r_p) = F(\{X\}) + \sum_j \left[\lambda_j \cdot \left(G_j(\{X\}) + Z^2\right) + r_p \cdot \left(G_j(\{X\}) + Z^2\right)^2\right] \tag{96}$$

By way of mathematical equivalence, Equation (79) brings an advantage over Equation (96) for it achieves the same conversion of inequality into equality constraints, but dispensing with the potentially many slack variables Z^2. Connecting with the optimization theory presented in the linear programming section, it should be noted that there will be as many Z^2 as there are inequality constraints, and then their number can be also contained by using the duality principle.

$$A(X, \lambda, r_p) = F(\{X\}) + \sum_j \left[\lambda_j \cdot \psi_j + r_p \cdot \psi_j^2\right] \tag{97}$$

$$\psi_j = \max\left(G_j(\{X\}), \frac{-\lambda_j}{2r_p}\right). \tag{98}$$

Finally, the all-encompassing case, with both equality and inequality constraints, can be represented in Equation (99) by combining Equations (93) and (97):

$$\begin{aligned} A(X, \lambda, r_p) \;\; = \;\; & F(\{X\}) + \sum_j \left[\lambda_j \cdot \psi_j + r_p \cdot \psi_j^2\right] \\ & + \sum_k \lambda_{k+m} H_k(\{X\}) + r_p [H_k(\{X\})]^2. \end{aligned} \tag{99}$$

Among the many advantages of this formulation, the following should be highlighted:

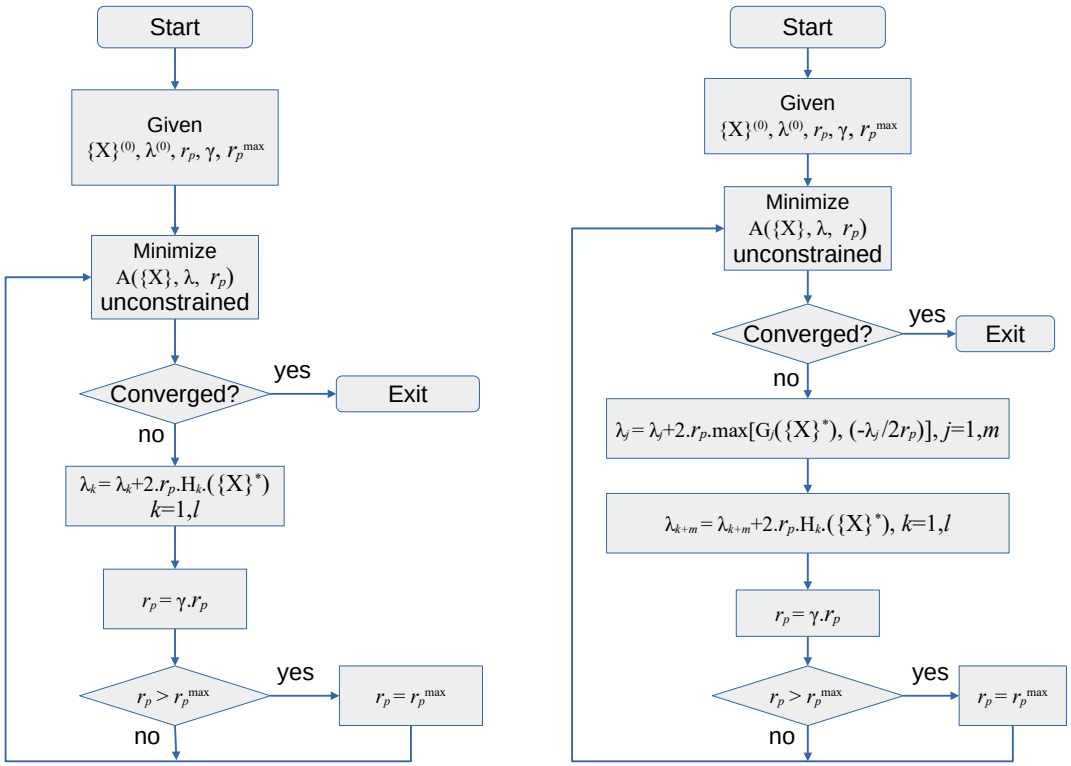
- Relative insensitivity with respect to the value of r_p;
- Precise zero values of both equality and inequality constraints are handled in a numerically stable way;
- Acceleration can be achieved by updating the values of the Lagrange multipliers, as in Equations (100) and (101):

$$\lambda_j^{(i+1)} = \lambda_j^{(i)} + 2r_p \cdot \max\left(G_j(\{X\}), \frac{-\lambda_j^{(i)}}{2r_p}\right), \; j = 1, \ldots, m \tag{100}$$

$$\lambda_{k+m}^{(i+1)} = \lambda_{k+m}^{(i)} + 2r_p \cdot H_k(\{X\}^{(i)}) \tag{101}$$

where k is the number of equality constraints.

Code 5 provides an illustrative implementation of the Augmented Lagrangian Multiplier method (Birgin and Martínez [2008]) in the R statistical computing platform (Johnson [2020]).



(a) Only equality constraints (b) Equality and inequality constraints

Figure 16: Augmented Lagrangian method for constrained optimization.

9 Direct Methods

The naming for this class of methods reflects their strategy of dealing with the constraints directly, instead of using modified versions of methods initially intended to solve unconstrained problems to tackle constrained ones.

9.1 Sequential Linear Programming (SLP)

This method consists of using a truncated Taylor series around a point X^0 as a linearized version of the general non-linear optimization problem, to be updated at every iteration. Equations (102) to (105) reflect the resulting problem statement, as it stems from the original Equations (1) to (4):

$$\min, \max F(\{X\}) \approx F(\{X^0\}) + \{\nabla F(\{X^0\})\}^T \cdot \delta\{X\} \tag{102}$$

subject to

$$G_j(\{X\}) \approx G_j(\{X^0\}) + \{\nabla G_j(\{X^0\})\}^T \cdot \delta\{X\} \leq 0 \tag{103}$$

$$H_k(\{X\}) \approx H_k(\{X^0\}) + \{\nabla H_k(\{X^0\})\}^T \cdot \delta\{X\} = 0 \tag{104}$$

and side constraints

$$\{X\}^{LB} \leq \{X\} + \delta\{X\} \leq \{X\}^{UB}. \tag{105}$$

Code 5: Augmented Lagrange Multiplier Method.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
gcon <- function(x) {
  return(x[1] + x[2] - 2.5) # x1 + x2 >= 2.5
}

x0 = c(-2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = auglag(x0=x0, fn = fobj, hin = gcon,
  lower = xl, upper = xu)
cat('Optimal value: ', opt1$value,
  '\nOptimal design: ', opt1$par)

Optimal value:  0.02504044
Optimal design:  1.158152 1.341849
```

It should be noted that, although this theoretical construct is such that any linear programming method as described in section 6 could in principle be applied to its solution, practitioners have devised more specific approaches, which rely on several tunning parameters to enhance the overall performance. As a result, SLP is usually a very robust alternative when solving complex engineering problems, even if the attained optimality would vary in a problem dependent fashion.

An example of the SLP method (Nelder and Mead [1965]) implemented in R is provided by Code 6 (Johnson [2020]).

9.2 Method of Feasible Directions and Modified Method of Feasible Directions (MFD and MMFD)

In trying to deal with the actual non-linearity of the optimization problem in an explicit manner, MFD attempts to follow the constraint boundaries, but not in a precise tangential fashion. It does so by leveraging the concepts of usable and feasible search directions $\{S\}$, as defined in Equations (106) and (107), respectively:

$$\{\nabla F(\{X^0\})\}^T \cdot \{S\} \leq 0 \tag{106}$$

$$\{\nabla G(\{X^0\})\}^T \cdot \{S\} \leq 0 \tag{107}$$

Equation (106) means that a small move from $\{X^0\}$ along $\{S\}$ will improve the value of the objective function $F(\{X\})$. Equation (107) means that a small move from $\{X^0\}$ along $\{S\}$ will not violate any constraint $G(\{X\})$, represented here without the sub-indices for brevity and generality.

As in Equation (108), a positive parameter θ is added to Equation (107) to ensure feasibility, that is, the constraint will not be violated despite movements along $\{S\}$

Code 6: Sequential Linear Programming Method.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
x0 = c(-2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = nloptr(x0=x0, eval_f = fobj, lb = xl, ub = xu,
  opts=list('algorithm'='NLOPT_LN_NELDERMEAD',
  'xtol_rel' = 1.0e-6))
cat('Optimal value: ', opt1$objective,
  '\nOptimal design: ', opt1$solution)

Optimal value:  0.003021801
Optimal design:  0.9569033 0.9122515
```

while trying to remain strictly tangent to a function with curvature:

$$\{\nabla G(\{X^0\})\}^T \cdot \{S\} + \theta \leq 0 \tag{108}$$

From vector dot product theory, the geometric interpretation of Equation (108) is that the cosine of the angle between the search direction and the gradient of the constraint function has a strictly negative value. Hence, the search direction is separated from the constraint boundary by an angle always greater than 90 deg, meaning that they are being moved away from each other, fulfilling the feasibility goal (no constraint violation). On the other hand, we can balance this separation motion (between the search direction and the constraint boundary) by prescribing the usability from Equation (106):

$$\{\nabla G(\{X^0\})\}^T \cdot \{S\} - \left(\{\nabla F(\{X^0\})\}^T \cdot \{S\}\right) \cdot \theta \leq 0 \tag{109}$$

Furthermore, the usability condition itself can be simplified, rendering Equation (96), in which the maximization of β will minimize the dot product involving the gradient of the objective function and the search direction, making the search direction further usable, that is, capable of improving the objective (note that the search direction no longer improves the objective when the dot product in Equation (106) crosses into positive territory, and therefore optimality is enhanced by having it minimized further into the negative one).

Combining Equations (109) and (110),

$$\{\nabla F(\{X^0\})\}^T \cdot \{S\} + \beta \leq 0 \tag{110}$$

the feasibility requirement becomes:

$$\{\nabla G(\{X^0\})\}^T \cdot \{S\} + \theta\beta \leq 0. \tag{111}$$

Besides rewriting the usability and feasibility conditions initially expressed in Equations (106) and (107), Equations (110) and (111) also hold as constraints applicable to the following transformation of the optimization problem statement, with Equation (112) subject to (113):

$$\max \beta \tag{112}$$

subject to

$$\{S\}^T \cdot \{S\} \leq 1. \tag{113}$$

The ensemble of Equations (110) to (113) is the method of feasible directions, based on maximizing a parameter β as long as the solution remains feasible, usable, the search direction does not degenerate into a vector whose magnitude explodes into infinity (Equation (113)).

The MMFD method, on its hand, once more rewrites the statement of the optimization problem as follows:

$$\max \left(\{-\nabla F(\{X^0\})\}^T \cdot \{S\} \right) \tag{114}$$

which is still equivalent to minimize Equation (106) in the spirit of usability, subject to Equation (113) and also

$$\{\nabla G_j(\{X^0\})\}^T \cdot \{S\} + \theta \leq 0 \tag{115}$$

which slightly but importantly rewrites Equation (107) to highlight individual constraints, allowing for a special treatment for those that are initially infeasible. Moreover, the equality constraints are also addressed in a specific manner, resulting in the following two additional constraints appended to the objective stated in Equation (114) and the original constraint in Equation (113) to complete the formulation:

$$[A] \cdot \{S\} \leq 0 \tag{116}$$

$$[B] \cdot \{S\} = 0 \tag{117}$$

where $[A]$ contains the gradients of inequality constraints only if they are active (with the safety margin term θ from Equation (109) dropped), and $[B]$ contains the gradients of the equality constraints.

9.3 Sequential Quadratic Programming (SQP)

As an extension of the linear approximation introduced via Equations (102) to (105), the SQP method will define a search direction creating a quadratic approximation for the augmented objective function and a linear approximation to the problem constraints, such that:

$$\min \left(F(\{X\}) + \{\nabla F(\{X\})\}^T \cdot \{S\} + \frac{1}{2} \cdot \{S\}^T \cdot [H_L] \cdot \{S\} \right) \tag{118}$$

subject to

$$\{\nabla G_j(\{X\})\}^T \cdot \{S\} + \delta_j \cdot G_j(\{X\}) \leq 0 \tag{119}$$

$$\{\nabla H_k(\{X\})\}^T \cdot \{S\} + \tilde{\delta} \cdot H_k(\{X\}) = 0 \tag{120}$$

Code 7: Sequential Quadratic Programming Method.

```
library(nloptr)

fobj <- function(x) {
  return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
}
gfobj <- function(x) {
  return( c( -400 * x[1] * (x[2] - x[1] * x[1])
    - 2 * (1 - x[1]), 200 * (x[2] - x[1] * x[1])) )
}

x0 = c(-2, 2); xl = c(-10, -10); xu = c(10, 10)
opt1 = nloptr(x0=x0, eval_f = fobj, eval_grad_f = gfobj,
  lb = xl, ub = xu,
  opts=list('algorithm'='NLOPT_LD_SLSQP',
  'xtol_rel' = 1.0e-6))
cat('Optimal value: ', opt1$objective,
  '\nOptimal design: ', opt1$solution)

Optimal value: 1.174518e-19
Optimal design: 1 1
```

where H_L starts as an identity matrix and gradually converges to the Hessian of the Lagrangian, which is directly derived from the KKT conditions of Equations (10)-(12) (expressed below in Equation (121)), with parameters δ_j and $\tilde{\delta}$ providing the other angle of the update process throughout the method iterations, as per the rules in Equations (122)-(124):

$$\nabla F(\{X\}) + \sum_j \lambda_j \cdot \nabla G_j(\{X\}) + \sum_k \lambda_k \cdot \nabla H_k(\{X\}) \tag{121}$$

$$\delta_j = 1, \quad \text{if } G_j(\{X\}) < 0 \tag{122}$$

$$\delta_j = \tilde{\delta}, \quad \text{if } G_j(\{X\}) \geq 0 \tag{123}$$

$$0 \leq \tilde{\delta} \leq 1, \quad \text{(typically, towards the upper bound).} \tag{124}$$

An example of the SQP method (Kraft [1988]) implemented in R is provided by Code 7 (Johnson [2020]).

This method will provide a compromise about efficiency in numerical computation (typical of linear methods) and precision in evaluation (typical of nonlinear methods). A number of nonlinear programming methods will rely on SQP as internal engine to solve nonlinear problems with reduced computational cost.

References

H. Agarwal and J. Renaud. Reliability based design optimization using response surfaces in application to multidisciplinary systems. *Engineering Optimization*, 36(3):291–311, 2004.

N. Alexandrov and S. Kodiyalam. Initial results of an MDO method evaluation study. In *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, September 1998.

T. Altiok and B. Melamed. *Simulation modeling and analysis with Arena*. Elsevier, 2010.

E. G. Birgin and J. M. Martínez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008.

S. Butenko and P. M. Pardalos. *Numerical Methods and Optimization: An Introduction*. Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series, 2014.

S. Butkewitsch and V. Steffen, Jr. A case study in frequency response optimization. In *DINAME 99 - Applied Mechanics in the Americas*, volume 8, 1999.

R. A. Canfield. Design of frames against buckling using a rayleigh quotient approximation. *AIAA journal*, 31(6):1143–1149, 1993.

R. S. Dembo and T. Steihaug. Truncated newton algorithms for large-scale optimization. *Math. Programming*, 26:190–212, 1983. doi: http://doi.org/10.1007/BF02592055.

X. Gandibleux. Multiple criteria optimization: state of the art annotated bibliographic surveys. 2006.

A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi. *Metaheuristic applications in structures and infrastructures*. Elsevier, 2013.

A. A. Giunta, V. Balabanov, D. Haim, B. Grossman, W. H. Mason, L. T. Watson, and R. T. Haftka. Aircraft multidisciplinary design optimisation using design of experiments theory and response surface modelling. *Aeronautical Journal*, 101 (1008):347–356, 1997.

X. Gu, J. E. Renaud, S. M. Batill, R. M. Brach, and A. S. Budhiraja. Worst case propagated uncertainty of multidisciplinary systems in robust design optimization. *Structural and Multidisciplinary Optimization*, 20(3):190–213, 2000.

A. Haldar and S. Mahadevan. *Probability, Reliability, and Statistical Methods in Engineering Design*. Wiley, 1999.

S. G. Johnson. The NLopt nonlinear-optimization package. 2020. URL `https://cran.r-project.org/package=nloptr`.

D. Kraft. A software package for sequential quadratic programming. Technical report, Technical Report DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, July 1988.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45:503–528, 1989.

D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer, 3rd edition, 2008.

L. Nardin, K. Sørensen, S. Hitzel, and U. Tremel. modeFRONTIER, a framework for the optimization of military aircraft configurations. In *MEGADESIGN and MegaOpt-German Initiatives for Aerodynamic Simulation and Optimization in Aircraft Design*, pages 191–205. Springer, 2009.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

R. D. Neufville, O. D. Weck, D. Frey, D. Hastings, R. Larson, D. Simchi-Levi, K. Oye, A. Weigel, and R. Welsch. Uncertainty management for engineering systems planning and design. In *Proceedings of the Engineering Systems Symposium, MIT*, March 2004.

B. Noble and J. W. Daniel. *Applied Linear Algebra*. Prentice-Hall, 1988.

G. O. Odu and O. E. Charles-Owaba. Review of multi-criteria optimization methods–theory and applications. *IOSR Journal of Engineering (IOSRJEN)*, 3 (10):1–14, 2013.

M. J. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org/`.

S. A. Ragon, Z. Gürdal, R. T. Haftka, and T. J. Tzong. Bilevel design of a wing structure using response surfaces. *Journal of Aircraft*, 40(5):985–992, 2003.

E. Rashedi, E. Rashedi, and H. Nezamabadi-pour. A comprehensive survey on gravitational search algorithm. *Swarm and evolutionary computation*, 41:141–158, 2018.

D. T. Sturrock and C. D. Pegden. Recent innovations in Simio. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 52–62. IEEE, 2011.

S. Theussl and K. Hornik. *Rglpk: R/GNU Linear Programming Kit Interface*, 2019. URL `https://CRAN.R-project.org/package=Rglpk`. R package version 0.6-4.

I. Ucar, B. Smeets, and A. Azcorra. Simmer: Discrete-event simulation for R. *arXiv preprint arXiv:1705.09746*, 2017.

G. N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design*. Vanderplaats Research and Development, 1998.

G. Venter and R. T. Haftka. Using response surface approximations in fuzzy set based design optimization. *Structural Optimization*, 18(4):218–227, 1999.

A. Waller. Witness simulation software. In *Proceedings of the Winter Simulation Conference*, pages 1–12, 2012.

B. Wilson, D. Cappelleri, T. W. Simpson, and M. Frecker. Efficient pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, 2(1): 31–50, 2001.