



Politecnico di Torino

Master's Degree Course in Automotive Engineering

Master's Degree Thesis

Tools For Optimisation Strategies in Mechanical Applications

Speaker:
Prof. Elvio Bonisoli

Correlator:
Eng. Venturi Simone

Candidate:
Mariam Sadek

A.a. 2021/2022

Acknowledgements

I am deeply grateful for Professor Bonisoli for taking a gamble on me and having me undertake his thesis proposal. I wish to have at least fulfilled his expectations of me. I am also grateful to him for letting me work with Eng. Simone Venturi, who is a well-deep in knowledge and understanding. I am thankful for his patience and mentorship along this journey.

I also owe everything to my mom Mona for her sleepless nights and my dad Hussein for working overtime to provide my education. I thank my sisters for their encouragement and supporting my decisions emotionally and financially when possible. I warmly thank my great friend Mirna as well for her kindness, patience, and intelligence. I hope to meet her soon in Italy. My last thanks go to my four cats that made this tough experience full of laughs and cuddles.

Index

Index.....	3
Abstract	4
Introduction.....	5
1. Preliminary optimisation introduction	6
1.1 Types of optimisation problems.....	6
1.1.1 Linear programming	6
1.1.2 Quadratic programming	6
1.1.3 Quadratic constraints and conic optimisation	7
1.1.4 Mixed integer programming	7
1.1.5 Constraint programming	7
1.1.6 Smooth nonlinear optimisation problems	7
1.1.7 Non-smooth optimisation problems	7
1.2 Local vs global optimisation	8
1.2.1 Local optimisation.....	8
1.2.2 Global optimisation.....	8
1.3 Constraints	10
1.3.1 A project management triangle example	10
1.4 Objective Function.....	11
1.4.1. Fitness function.....	11
1.4.2 Deterministic vs probabilistic models.....	11
1.4.3 Optimal configuration	11
2. Optimisation strategies.....	12
2.1 Simplex method	12
2.2 Genetic algorithm.....	15
2.3 Trust region method	17
2.3.1 Taylor expansion.....	17
2.3.2 Methodology	19
2.4 Interior point method	21
2.5 Sequential quadratic programming	23
Convergence of the error.....	25
3 Optimisation Panel.....	27
3.1 Fmincon function	27
3.2 Graphical user interphase application	29
3.3 Optimisation panel guide	31
3.4 External code examples	38
3.4.1 Rosenbrock function	38
3.4.2 Handle function.....	39
3.4.3 External function.....	41
3.4.4 External application	42
4. Case studies.....	45
Buckingham pi theorem:	50
5. Uniqueness of optimisation.....	62
5.1 Initial point.....	62
5.2 Redundant constraints	63
5.3 Introducing new variables.....	63
Final remarks.....	64
Conclusions.....	65
Reference	66

Abstract

Many aspects of engineering nowadays focus on improving what has already been invented, to better existing technology. The idea behind this paper is just that. This paper will discuss what is optimisation and the need for optimisation in different applications. In what follows, an optimisation tool will be developed to help students and academics properly construct their problems and objectives and find suitable and feasible arrangements to solve or optimise them. The tool is a graphical user interphase constructed using MATLAB functions. It has been applied to a case study called: “Uncertainty Effects on Bike Spoke Wheel Modal Behaviour”. In addition, an original approach to problem dimensionality reduction has been introduced and applied on the provided case study with varying results and future possibilities.

Introduction

Since the start of time, creatures evolved in ways that would increase their chances of survival. That could be seen as a form of optimization. Optimisation can be defined as the betterment of a situation or the use of a resource to fulfil certain requirements. In a more direct approach, people use optimisation methods instinctively and sometimes unconsciously every day, such as choosing a product that offers the best compromise between quality, quantity, and price. Optimisation problems are not always so simple. During a traffic or an accident, the shortest route isn't always the fastest one to a destination. Many variables and unknowns are involved that can change the shape and outcome of what was previously perceived as the best solution for the problem. Optimisation is an act that people perform daily, from which they find solutions rather quickly because they are veterans in the fields of their expertise. However, when the challenges become too complicated with parameters that are highly dependent and faced with multiple constraints and new risks, it becomes necessary to revise previous solutions and admit that current knowledge or techniques are not enough to optimize the problem. Optimisation in its simplest form is a mathematical equation or formula with certain variables. The task is to find the best relationship between those variables within a set of constraints that allow to achieve the desired outcome (such as maximizing profits while reducing sales). In the above example, under normal circumstances, companies would rely on economies of scale. Unfortunately, that would not benefit this business since much of the produced goods will be redundant and hence further increase the cost. So, here one would need to tackle this issue from a different perspective. The company can increase the prices, but the outcome depends on the level of correlation between price and the number of sales. So, ultimately it may result in decreasing the profit; similarly with lowering the quality to cut costs. An extensive study is needed to reach a favourable decision. Optimisation tools have been created to assist and facilitate the decision-making process. One such tool is the aim of this thesis paper. The Optimisation Panel is a friendly graphical user interphase that can solve most optimisation problems.

1. Preliminary optimisation introduction

Every optimisation problem consists of decision variables, an objective function, and constraints. The decision variables are the main interest of the optimisation procedure. They are the values that can be reproduced and modified in a real-world setting. The objective function is the mathematical translation of a real-world problem. The solution of the objective function can be remodelled into a set of actions and decisions in which the end purpose is to replicate the desired simulated outcome. The functions can be simplified or tackled head on. However, “Simplicity is a principle, not a rule” [15]. Simplification can expedite the optimisation procedure and efficiently offer viable solutions to extremely complex problems. Nevertheless, it is not always crucial to invest in such methods, or it may not even be possible. There is an infinite number of variables that can be introduced to a single study. Normally, most of these variables can be omitted due to their lack of importance, inability to control, or to their low probability of occurrence. Moreover, instead of maximizing or minimizing a certain objective, it becomes necessary to work towards achieving conflicting needs. Hence, it is important to introduce a list of constraints that trap the solutions within a range of favourable and/or feasible arrangements.

1.1 Types of optimisation problems

With multidimensional equations, the type of the problem can dictate which optimisation procedure to follow, as some can be quite complex and unnecessarily lengthy while others can be applied only to special formats.

1.1.1 Linear programming

In a Linear Programming (LP) problem, the objective function $f(\mathbf{x})$ and constraints $C(\mathbf{x})$ are all linear functions of the decision variables. Each have the following form respectively:

$$f(\mathbf{x}) = a_0 + \sum_{i=1}^n a_i x_i \quad (1.1.1.1)$$

$$C(\mathbf{x}) : \sum_{i=1}^n c_i x_i = c_0 \quad (1.1.1.2)$$

Where a_i and c_i are constant coefficients, and \mathbf{x} is the $(n \times 1)$ vector of decision variables x_i .

1.1.2 Quadratic programming

In a Quadratic Programming (QP) problem, the objective function is a quadratic function while the constraints are linear (refer to equation 1.1.1.2). A quadric is a surface defined by a second-degree algebraic equation. A quadratic objective function has the following generic form:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{A} \mathbf{x} + b \quad (1.1.2.1)$$

Where \mathbf{x} is the $(n \times 1)$ vector of decision variables and \mathbf{x}^T is its transpose. \mathbf{H} and \mathbf{A} are $(n \times n)$ and $(1 \times n)$ matrices respectively with constant elements and b is a constant coefficient. The construction of the \mathbf{H} and \mathbf{A} matrices are explained in pages 33 and 34.

1.1.3 Quadratic constraints and conic optimisation

The constraints are quadratic functions (refer to equation 1.1.2.1). If the quadric surface resembles that of a cone, the problem becomes a Second Order Cone Programming (SOCP) [1], in which the vector solution of decision variables is constrained to lie within a cone. Equation 1.1.3.1 represents the canonical constraint $C(\mathbf{x})$ of a n dimensional problem:

$$C(\mathbf{x}) : \sum_{i=2}^n x_i^2 \leq x_1^2 \quad (1.1.3.1)$$

Where \mathbf{x} is the $(n \times 1)$ vector of decision variables x_i .

1.1.4 Mixed integer programming

A Mixed-Integer Programming (MIP) problem is one where some or all the decision variables are constrained to be of integer value. This approach closes in on real world optimisation problems. It forces a discrete nature on the decision variables. Multiple iterations are performed until an integer valued vector solution is found that is closest to the optimum solution.

1.1.5 Constraint programming

Constraint Programming (CP) does not just refer to having constraints, it refers to the type of constraint. It originated from artificial intelligence research where the problems require assigning symbolic values which are finite in number and are usually presented by integer values. The most common of these constraints is the all-different constraint [4] where a n set of decision variables needs to be arranged in a non-repeating order \mathbf{P} . If a position is repeated, then the solution is invalid.

1.1.6 Smooth nonlinear optimisation problems

A nonlinear problem is one in which the objective or at least one of the constraints is a nonlinear function. Smooth indicates that the functions used, and their derivatives are continuous. Examples of nonlinear functions include polynomials of second-degree or higher, variables divided by each other, radicals, and transcendental functions such as log, exp, sine and cosine. Quadratic programming is a special case of Smooth Nonlinear Problems (SNLP).

1.1.7 Non-smooth optimisation problems

Non-Smooth Problems (NSP) are the most difficult optimisation problems to solve. They refer to problems that are nondifferentiable at their maximisers and minimisers. Hence, gradients and derivatives generally cannot be used to determine the direction of the function. During iterative

procedures, a current solution may not be able to lead to a better one. Therefore, in most NSPs, a sample of possible solutions is gathered and the best one is chosen. They rely on random sampling of possible solutions which can yield different results on each run depending on the chosen points. The type of optimisation tool to use depends on the nature of the optimisation problem. The Simplex method is most suitable for linear programming problems, while the genetic algorithm is preferred to be used for more complex and complicated problems. Different optimisation tools will be discussed further on.

1.2 Local vs global optimisation

The object of any optimisation procedures is to find the optimum of the objective function that satisfies the constraints of the problem. However, in an unconstrained problem, there might not always exist an optimum.

1.2.1 Local optimisation

A local optimum is the minimum or maximum of the objective function in a limited region of the solution (bounded solution). If f is the objective function defined over a set X and S is a subset of X , then the point $\mathbf{x}^* \in S$ is a local maximiser of $f(\mathbf{x})$ if $\exists \varepsilon > 0$ such that, $distance(\mathbf{x}, \mathbf{x}^*) < \varepsilon \rightarrow f(\mathbf{x}) \leq f(\mathbf{x}^*) \forall \mathbf{x} \in S$, whereas \mathbf{x}^{*s} is the local minimiser if $f(\mathbf{x}) > f(\mathbf{x}^{*s}) \forall \mathbf{x} \in S$.

Hence, an objective function can have multiple local optima. If the function has a single optimum, then the local optimum is also the global one. Local optimisation methods can be used to find the global optimum if the region where the global optimum resides is known, or if the local optimum is the same as the global optimum. This is an efficient way to reduce computational time and effort.

1.2.2 Global optimisation

A global optimum is the minimum or maximum of the objective function over the entire search space. Every objective function has a global optimum or else it could not be optimised.

The global minimum can be found as the minimum of all the local minima. Similarly, the global maximum is the maximum of all the local maxima. The definition of the global optimiser is the same as that of the local optimiser, but instead of S being a subset of the search input X , it is the search input X .

If the objective function has more than one global optimum, then the value of the function is the same at these respective inputs. In this case, the problem becomes multimodal. The global optimisation algorithm is used when little is known about the shape or behaviour of the objective function, as well as, when there are interfering local optima.

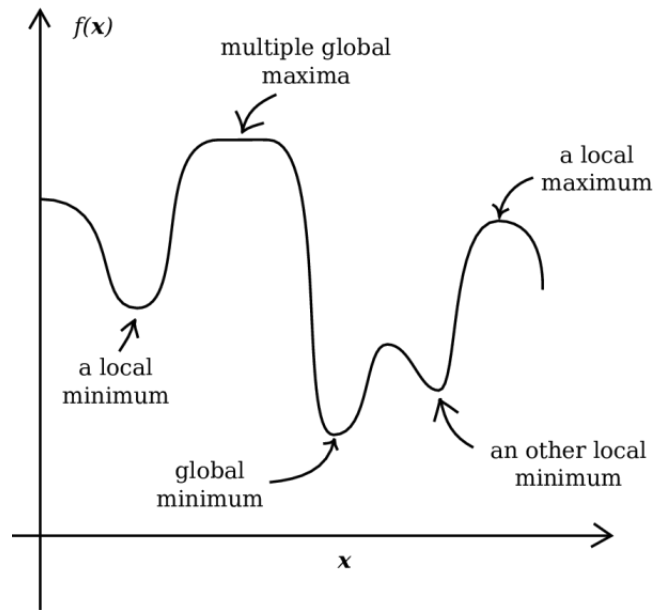


Figure 1.2.1 – Example of local and global optima of a one-dimensional problem [16]

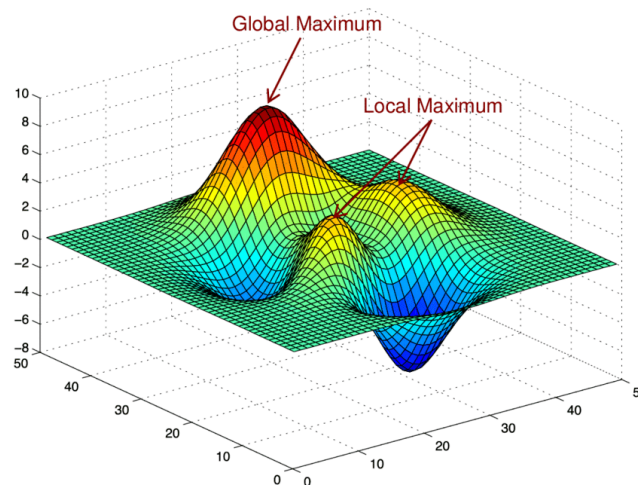


Figure 1.2.2 – Example of local and global maxima of a two-dimensional problem [17]

Unimodal problems can be solved through direct approaches. However, most problems are multimodal in nature since the desired target often depends on variable parameters and conflicting interests. It is noteworthy to say that the local/ global optima can be themselves functions. The objective function can have variable or time-dependent coefficients and thus the optima may depend on these changeable parameters.

1.3 Constraints

The main aim of constraints is to limit the range of variability of the system or problem. They generally refer to the limitations or restrictions imposed on the decision variables. A constraint does not change the shape of the function but defines the region of acceptable solutions. The most common constraint is the boundary constraint in which the decision variables are trapped between an upper bound (UB) and a lower bound (LB). If the variables are not left constrained, then the lower bound is denoted with $-\infty$. If the variables are not right constrained, then the upper bound is denoted with $+\infty$. The distance between the two bounds represents the range of the system. The system can also have multiple boundary conditions. Constraints include the equality, and the inequality constraint functions. The variables are either permitted to or excluded from certain values within the range of the system. These conditions can also prohibit two decision variables from having the same value. The constraint functions are not necessarily linear functions. They can introduce new relations between the decision variables or a level of correlation among the parameters. The problem description and system requirements should be analysed very thoroughly to determine which relations should be written as equalities and which as inequalities.

1.3.1 A project management triangle example

In project management, the three major constraints are time, cost, and scope, also known as the project management triangle [5]. These unique constraints are dependent on each other and thus altering one will affect the other two. Each constraint is linked to the following objectives. The time constraint refers to the time division of tasks and the duration of the project. The scope constraint refers to the features and goals of the project. The cost constraint refers to the budget of the project, the necessary expenses to finish the project on time and within its scope. One way to maximise the profit (objective function) could be by reducing costs. However, this might affect the quality of the product which might result in losing the competitive advantage in the market. Another way could be by expanding the scope and thus the customer base (to increase sales). Nevertheless, there is a risk of prolonging the product release date into the market and losing initiative, and consequently a need to increase costs to accommodate for the new product design. Looking at this example, it is impossible to reach a perfect result (a fast releasing, high quality, and low-cost product). Constraints are integrated into the problem, not to find the perfect solution, but a better feasible one. It allows a range of variability that offers a compromise between the different decision variables. Someone might look at the problem from a different perspective and note that, from the first glance, the profit equation:

$$P(s, c) = price \cdot s - c \quad (1.3.1.1)$$

does not include a time variable, where s and c are the number of sales and costs respectively. A solution that optimizes both the scope and limits the costs is reached at the expense of time. However, it would be soon realised that it is a false solution when sales do not match expectations. When writing constraints, they can be either implicit or explicit. Explicit constraints are explicitly defined, where the decision variables are restricted to a given set of values. Implicit constraints, on the other hand, depend on design parameters that cannot be explicitly defined or measured.

1.4 Objective Function

The objective function is the target of optimisation that needs to be minimized or maximized to find its corresponding decision variables that satisfy a set of constraints.

1.4.1. Fitness function

If the objective function is the target of optimisation, then the fitness function guides the optimisation process. A fitness function is a special form of an objective function. It can be the same as the objective function if the only goal is to either maximise or minimise the given objective function. However, in multi-objective and constraint problems, it takes a different form. It is mainly used in genetic programming and genetic algorithms. It defines the goal of the genetic algorithm. It compares how good or fit two solutions are to the problem's set of aims. Hence, it defines the relative importance of a design, where higher values correspond to a better design. It is a function that maps the solution representation into a real scalar value. So, it is a function of the form: $f: \Gamma \rightarrow \mathbb{R}$, where Γ is the space of vectorial solutions.

The fitness function is a way to incorporate constraints on the shape of the fitness landscape. Each fitness function corresponds to an objective. In a multi-objective optimisation problem, a final fitness function is established as a combination of these multiple fitness functions.

Once the objective functions are optimised and a set of solutions is presented, a selection process performed by the fitness function takes place that evaluates the fit of the set.

1.4.2 Deterministic vs probabilistic models

A deterministic model does not have random elements. The results of each iteration do not change if the initial conditions are the same. On the contrary, the solution of a probabilistic model changes with each iteration even if the initial conditions are preserved. The occurrence of events in a deterministic system is known with accuracy while that of a probabilistic system is hard to predict. If a data is missing or its contribution is unknown, then probabilistic data are used to enrich deterministic datasets and to scale deterministic models.

To direct the flow of a probabilistic problem, brute force is used to control certain aspects of the model and thus turning it into a deterministic one by eliminating the randomness in each iteration. It is noteworthy to mention that the brute force approach is not an optimisation technique. It is merely a control strategy.

1.4.3 Optimal configuration

It is important to differentiate between optimum and optimal. Optimum is the best while optimal refers to the best possible. Due to many reasons or constraints, it is not always possible to consider the optimum of the objective function as a feasible solution to the optimisation problem. Performance optimization is an iterative method of creating and monitoring modifications to an application or problem. Hence, by continuously changing the problem description, it is likely to further optimize the previously perceived solution. These modifications can include design changes or new constraint introduction.

2. Optimisation strategies

The optimisation strategy used depends on the complexity of the problem and the level of risk associated with it. For each strategy, the outcomes need to be properly defined, the database arranged, and the precision, performance, and implementation need to be continuously monitored. In what follows, are some demonstrations of well-known and commonly used optimisation strategies. Each is an iterative procedure which models the NLP for a given iterate x_k , where $k \in \mathbb{N}$. The solution of the subproblem is used to construct the new iterate x_{k+1} which should converge to the optimiser of the NLP as k tends to ∞ .

2.1 Simplex method

In 1947, George Bernard Dantzig [8] created the simplex algorithm. The simplex method is an approach for solving linear models. The optimal solution exists on the vertices. It is a systematic procedure that relies on finding pivot points to reduce row elements above and below the unitary pivot point to zero through row operations. It was designed to be able to solve linear programming problems by hand instead of relying on software.

To apply the simplex method, the problem must be transformed into the standard form. A standard form satisfies three requirements:

- The problem is a maximisation problem. It is sufficient to multiply both sides of the objective function by a -1 to change it from a minimisation problem to a maximisation problem.
- All constraints are in a less than or equal to inequality. The greater than or equal to inequality can be inverted by multiplying both sides by -1.
- All variables are positive.

The following linear programming problem is obtained:

$$\max z = \sum_{i=1}^n a_i x_i \quad (2.1.1)$$

$$C_j(x) : \sum_{i=1}^n c_{ji} x_i \leq c_j \quad (2.1.2)$$
$$j = 1 : m$$

Where n and m are the numbers of decision variables and constraint conditions respectively, a_i is the constant coefficient of the i^{th} variable x_i in the objective function, and c_{ji} is the constant coefficient of the i^{th} variable x_i in the j^{th} constraint equation.

Afterwards, slack variables are added to change the inequality constraints to equality ones.

For the objective function, all right terms are moved to the left and the equation becomes equal to zero. The equations are then rewritten in a table or matrix form with the top row representing all the variables in the system, the last row and the rows in-between containing the coefficients of the objective function and the constraints respectively corresponding to each variable, and the last column holding the equality values for each row.

$$\max z - \sum_{i=1}^n a_i x_i = 0 \quad (2.1.3)$$

$$C_j(x): \sum_{i=1}^n c_{ji}x_i + s_j = c_j \quad (2.1.4)$$

$$j = 1:m$$

x_1	x_2	\dots	x_n	s_1	s_2	\dots	s_m	z	b
c_{11}	c_{12}	\dots	c_{1n}	1	0	\dots	0	0	c_1
c_{21}	c_{22}	\dots	c_{2n}	0	1	\dots	0	0	c_2
\cdot	\cdot	\cdot	\cdot	\cdot	0	\cdot	\cdot	\cdot	\cdot
\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot
\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot	\cdot
c_{m1}	c_{m2}	\dots	c_{mn}	0	0	\dots	1	0	c_m
$-a_1$	$-a_2$	\dots	$-a_n$	0	0	\dots	0	1	0

(2.1.5)

Where s_j is the slack variable of the j^{th} constraint equation and b represents the right-hand term of each equation.

Optimality is reached when all the values in the last row are greater than or equal to zero. The method revolves around finding the pivot point which is the intersection between the pivot column and the pivot row. The pivot column is the one containing the smallest number (greatest negative number) in the last row. The pivot row corresponds to the one with the lowest indicator. The indicator is the value of the equality (last column) divided by its respective coefficient in the pivot column.

The pivot row is then divided by the value of the pivot point to have a unitary pivot value. After that, row operations are performed with the pivot row to change the values of all the other coefficients in the pivot column to zero.

Optimality needs to be checked for each new table. If optimality is not reached, another iteration is performed by finding the new pivot point and redoing the previous steps.

After the last iteration, the optimal values can be identified. A basic variable has a single 1 in its column and all other values are zero, else, it is non-basic. The optimal solution of a non-basic variable is zero. For a basic variable, the optimal solution is the intersection of the last column with the row containing the 1 value.

For a two/three-variable system, the solutions can be found graphically by plotting the graphs of the constraints and the objective function.

Consider the following 2D problem:

$$\begin{aligned} \max z &= 40x_1 + 30x_2 \\ x_1 + 2x_2 &\leq 16 \\ x_1 + x_2 &\leq 9 \\ 3x_1 + 2x_2 &\leq 24 \\ x_1, x_2 &\geq 0 \end{aligned} \quad (2.1.6)$$

There are two ways to find z_{\max} graphically. The first is by transforming the objective function into a 2D line by assuming that z is a constant. Hence, the following equation is formed:

$$x_2 = -\frac{40}{30}x_1 + \frac{1}{30}z \quad (2.1.7)$$

By taking $z = 0$, equation (2.1.7) can also be plotted. Since z is a positive value, as z increases, the line of equation (2.1.7) would be shifted upward. Hence by moving the straight line with slope $-4/3$ along the feasible region, the optimal solution is found when the line is tangent to the feasible region, except at point A (the origin) which is a impractical solution.

The second graphical method depends on the extreme points. Since the objective function can only be tangent to the feasible region at its extremities, plotting the objective function can be omitted and instead, the value of z is calculated at each of the extreme points. The highest value of z is then at the optimal solution.

The shaded region in figure (2.1.1) is the feasible region which is the intersection of all the constraint equations (in red). The yellow line is the translation of the objective function (equation 2.1.7). Both figures below yield the same solution which is $x_1 = 6$ and $x_2 = 3$.

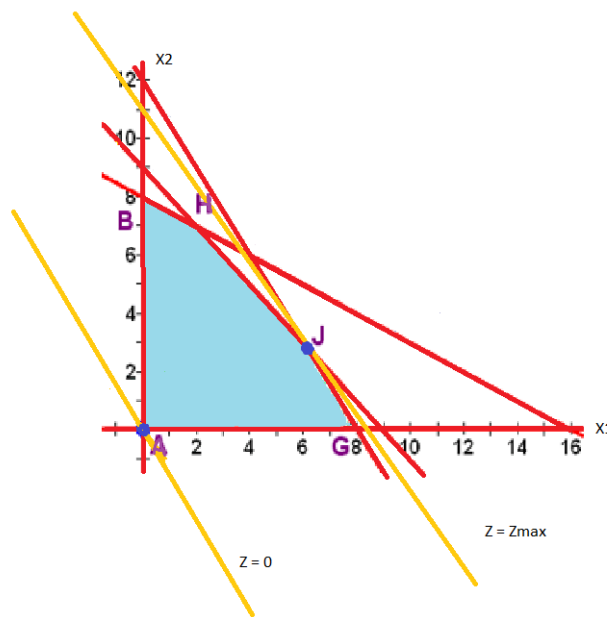


Figure 2.1.1 – Representation of a graphical solution

Table 2.1.1 – Evaluation of extremities.

Extreme Points	x_1	x_2	z
A	0	0	0
B	0	8	240
H	2	7	290
J	6	3	330
G	8	0	320

2.2 Genetic algorithm

Genetic Algorithm (GA) is a process based on natural selection. It drives the biological evolution of species.

In a population (optimisation problem), there are many individuals with different characteristics, in addition to unforeseen events or mutations. In every population, there exists dominant individuals with dominant features and non-dominant individuals with recessive features. Naturally, most of the offspring of the population will resemble their dominant parent but still inherit some characteristics of their less dominant parent, and in the case of a mutation, these recessive characteristics may become more apparent. It is a probabilistic system. Hence, more dominated solutions have a higher probability of occurring. However, using the pareto-based fitness assignment which gives equal probability of reproduction to all non-dominated members of the population, non-dominated solutions have a higher fitness than dominated solutions. The probability of being selected to the population depends on the fitness value. Genetic operators are used to explore the research space more effectively. The crossover operator, as in figure 2.2.1, uses a random cut-point crossover method. Two parents are chosen, and two child solutions are created, where the first child takes after the first part of the first parent and the second part of the second parent, while the second child takes after the second part of the first parent and the first part of the second parent. On the other hand, the mutation operator, as in figure 2.2.2, picks at random one or more operations to substitute into the child's genetic sequence.

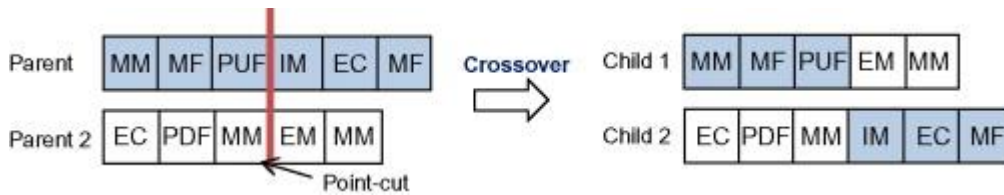


Figure 2.2.1 – Example of crossover operator [6]



Figure 2.2.2 – Example of mutation operator [6]

The last operator used in genetic algorithms is the selection operator. Many selection criteria exist, but they differ with the way they either evaluate or compute the fitness of the individuals. A well-known selection operator is the Fitness Proportionate Selection (FPS), also known as the Roulette Wheel Selection (RWS) where any individual can be selected. However, the probability of that happening depends on the individual's fitness. Individuals with higher fitness values have a higher chance of getting selected according to equation (2.2.1).

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j} \quad (2.2.1)$$

Where p_i is the probability of the individual with fitness f_i to be selected and n is the number of individuals.

Figure 2.2.3 represents the optimisation procedure regarding genetic algorithms. The initial population is chosen randomly from the research space. Operators are applied to the population and solutions are presented. A selection process takes place that discards infeasible solutions. If the termination criteria are not met, another iteration of the solution is performed.

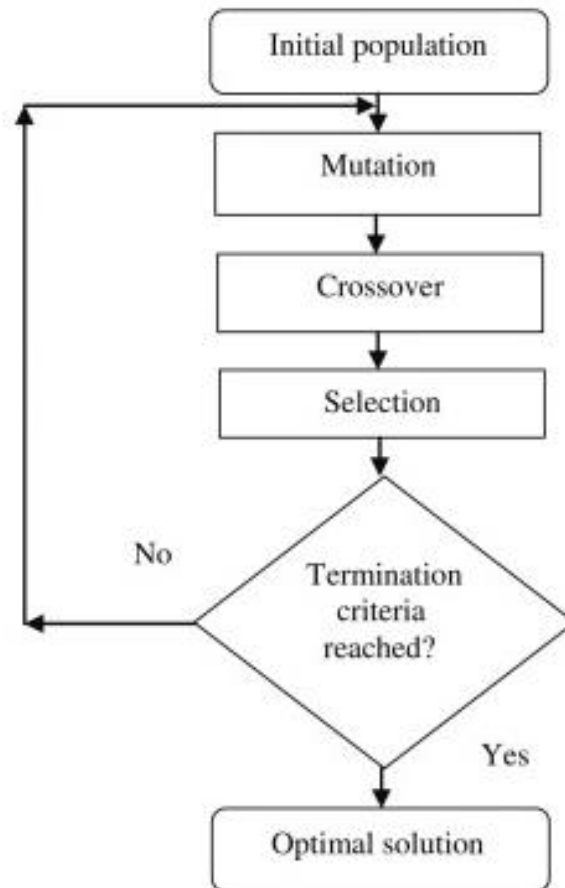


Figure 2.2.3 – Differential evolution process [7]

2.3 Trust region method

A trust region is a subset of the solution space where an appropriate model is trusted to be an acceptable representation of the objective function. A quadratic approximation of the objective function is often chosen as the model which is the second-order Taylor polynomial of the function. The trust region is often chosen as a hypersphere.

2.3.1 Taylor expansion

The Taylor expansion to the order k approximates a k times differential function at a given point.

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k \quad (2.3.1.1)$$

Such that

$$\lim_{x \rightarrow a} h_k(x) = 0 \quad (2.3.1.2)$$

Where:

$$x \in \mathbb{R} \quad (2.3.1.3)$$

Hence the k^{th} order Taylor polynomial is defined as:

$$P_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots + \frac{f^{(k)}(a)}{k!}(x-a)^k \quad (2.3.1.4)$$

A linear approximation is a 1st order Taylor polynomial which is also the tangent to the function at that point.

$$P_1(x) = f(a) + f'(a)(x-a) \quad (2.3.1.5)$$

The approximate error of Taylor polynomial is:

$$R_k(x) = f(x) - P_k(x) = h_k(x)(x-a)^k \quad (2.3.1.6)$$

Which tends to zero as x tends towards a .

The approximation becomes more accurate in the region surrounding the point a as the order of the polynomial increases. Hence a quadratic polynomial is a better approximation than a linear one.

$$P_2(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 \quad (2.3.1.7)$$

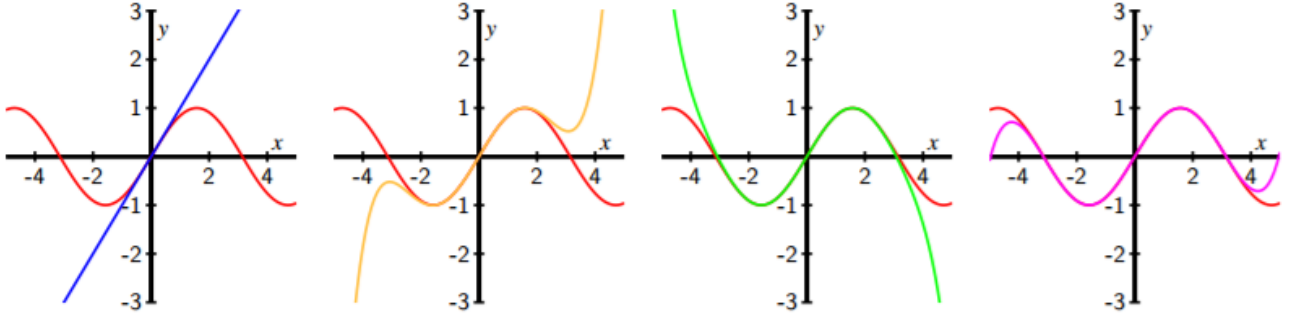


Figure 2.3.1.1 – The 1st, 5th, 7th, and 9th order Taylor polynomial at $x = 0$ for $f(x) = \sin(x)$ [13]

In the case of a multivariable function, the single partial derivative is substituted with a matrix of partial derivatives.

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \nabla f(\mathbf{x})(\mathbf{x} - \mathbf{a}) + \sum_{|\alpha|=2}^k (\mathbf{x} - \mathbf{a})^T \frac{D^\alpha f(\mathbf{x})}{\alpha!} (\mathbf{x} - \mathbf{a})^{\alpha-1} + \sum_{|\alpha|=k} h_\alpha(\mathbf{x})(\mathbf{x} - \mathbf{a})^\alpha \quad (2.3.1.8)$$

Where $D^\alpha f(\mathbf{x})$ is a squared ($n \times n$) matrix composed of the α order partial derivatives of the function $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$ is the gradient of $f(\mathbf{x})$ where:

$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T \in \mathbb{R}^n \quad (2.3.1.9)$$

$$\nabla f = \left(\frac{\partial f}{\partial x_1} \ \dots \ \frac{\partial f}{\partial x_n} \right) \quad (2.3.1.10)$$

The Hessian (\mathbf{H}) is a second order partial derivative matrix which corresponds to $D^2 f(\mathbf{x})$:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (2.3.1.11)$$

Hence, the quadratic approximation $q(\mathbf{x})$ in a region N around \mathbf{x}_k can be written as:

$$q_k(\mathbf{x}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \mathbf{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \quad (2.3.1.12)$$

The hypersphere trust region is defined as:

$$\|\mathbf{x} - \mathbf{x}_k\| \leq \Delta_k \quad (2.3.1.13)$$

Where \mathbf{x}_k is the centre and Δ_k is the radius of the trust region at the k iteration.

2.3.2 Methodology

Starting from an initial point \mathbf{x} , the algorithm aims to find another point with a higher optimality value by approximating a complex objective function f with a simpler quadratic function that resembles the behaviour of the original function inside a neighbourhood around \mathbf{x} , known as the trust region. The method revolves around computing \mathbf{s} by solving the trust region subproblem:

$$\underset{\mathbf{s}}{\text{Optimum}}\{q(\mathbf{s}), \mathbf{s} \in N\} \quad (2.3.2.1)$$

Where N is the trust region and q is the quadratic approximation of the objective function. For a minimisation problem, if $f(\mathbf{x} + \mathbf{s}) \leq f(\mathbf{x})$, then the trust region is moved to become the neighbourhood of point $(\mathbf{x} + \mathbf{s})$. Else, the trust region is reduced in size. The same occurs in a maximisation problem, but the condition becomes: $f(\mathbf{x} + \mathbf{s}) \geq f(\mathbf{x})$. By substituting $q(\mathbf{s} + \mathbf{x})$, the subproblem becomes:

$$\underset{\mathbf{s}}{\text{Optimum}}\left\{\frac{1}{2}\mathbf{s}^T \mathbf{H}(\mathbf{x})\mathbf{s} + \nabla f(\mathbf{x})\mathbf{s} + f(\mathbf{x}), \|\mathbf{s}\| \leq \Delta\right\} \quad (2.3.2.2)$$

The solution to the subproblem satisfies the following equation:

$$(\mathbf{H} + \lambda \mathbf{I})\mathbf{s} = -\nabla f \quad (2.3.2.3)$$

Where \mathbf{I} is the identity matrix and λ is the Lagrange multiplier of the subproblem. λ is found by solving:

$$\nabla_{\mathbf{s}, \lambda} \left(\frac{1}{2}\mathbf{s}^T \mathbf{H}\mathbf{s} + \nabla f\mathbf{s} + f - \lambda (\|\mathbf{s}\| - \Delta) \right) = 0 \quad (2.3.2.4)$$

There exists different literature on how much to expand or contract the trust region. However, this paper will not be discussing them.

In conclusion, the trust region method consists of four steps:

- Formulating the trust region subproblem.
- Solving the subproblem for \mathbf{s} .
- If $f(\mathbf{x} + \mathbf{s}) < f(\mathbf{x})$ (for minimisation) or $f(\mathbf{x} + \mathbf{s}) \geq f(\mathbf{x})$ (for maximisation), then $\mathbf{x} = \mathbf{x} + \mathbf{s}$.
- Adjusting Δ according to preferred literature.

After completing the above four steps, a new iteration is conducted starting from the new acquired point and trust region radius. The iterations can be halted once convergence is reached.

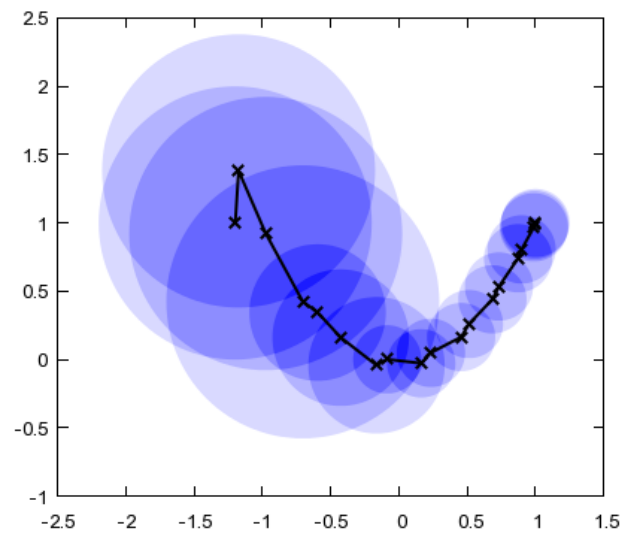


Figure 2.3.2.1 – Visual representation of an optimisation performed with trust region algorithm [14]

2.4 Interior point method

Interior point method, also known as barrier method, is used to solve linear and nonlinear convex optimisation problems. The algorithm is a minimisation solver. If the problem is a maximisation one, it is sufficient to multiply the objective function by -1. Starting from a general problem description as equation (2.4.1), it is necessary to transform the problem into the standard format.

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{such that} \quad & \begin{cases} g(\mathbf{x}) \geq 0 \\ h(\mathbf{x}) = 0 \end{cases} \end{aligned} \quad (2.4.1)$$

The problem format consists only of the barrier function to minimise and the set of equality constraints. The inequality constraints are converted into equality ones through the subtraction of slack variables which are positive by nature.

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{such that} \quad & \begin{cases} c(\mathbf{x}) = 0 \\ \mathbf{s} \geq 0 \end{cases} \end{aligned} \quad (2.4.2)$$

Where \mathbf{s} is the vector of slack variables and $c(\mathbf{x})$ is:

$$c(\mathbf{x}) = \{g(\mathbf{x}) - \mathbf{s}, h(\mathbf{x})\} \quad (2.4.3)$$

Finally, the inequality constraints are integrated into the objective function to make up the barrier function as such:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & f(\mathbf{x}) - \mu \sum_{i=1}^m \ln(s_i) \\ \text{such that} \quad & c(\mathbf{x}) = 0 \end{aligned} \quad (2.4.4)$$

Where μ is a small positive scalar called the barrier parameter, and m is the size of \mathbf{s} and the number of inequality constraints in $g(\mathbf{x})$.

Another form of the problem neglects the use of slack variables and integrates $g(\mathbf{x})$ directly into the barrier function.

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & f(\mathbf{x}) - \mu \sum_{i=1}^m \ln(g_i(\mathbf{x})) \\ \text{such that} \quad & h(\mathbf{x}) = 0 \end{aligned} \quad (2.4.5)$$

The search points are limited to the interior feasible space. The overpass of the inequality constraints is prevented by modifying the objective function with a barrier term that restricts the unconstrained value within the region of acceptable solutions.

As μ converges to zero, the solution of the barrier function tends to that of the original unconstrained objective function.

The barrier problem should satisfy the Karush-Kuhn-Tucker (KKT) conditions for the solution to be optimal. The theorem states that if $f(\mathbf{x})$, $g(\mathbf{x})$, and $h(\mathbf{x})$ are continuously differentiable at an optimal point \mathbf{p} , then there exists constants $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$, called the KKT multipliers, that satisfy the following:

$$\begin{aligned} \text{For minimising } f(\mathbf{x}) : \nabla f(\mathbf{p}) + \nabla h(\mathbf{p})\boldsymbol{\lambda}_p - \nabla g(\mathbf{p})\boldsymbol{\mu}_p &= 0 \\ \text{For maximising } f(\mathbf{x}) : -\nabla f(\mathbf{p}) + \nabla h(\mathbf{p})\boldsymbol{\lambda}_p - \nabla g(\mathbf{p})\boldsymbol{\mu}_p &= 0 \\ h(\mathbf{p}) &= 0, \quad g(\mathbf{p}) \geq 0 \\ \boldsymbol{\mu}_p &\geq 0 \\ g(\mathbf{p})\boldsymbol{\mu}_p &= 0 \end{aligned} \quad (2.4.6)$$

By applying the KKT conditions to problem (2.4.4), the following is obtained:

$$\begin{aligned} \nabla f(\mathbf{x}) + \nabla c(\mathbf{x})\boldsymbol{\lambda} - \mu \sum_{i=1}^m \frac{1}{s_i} \nabla(s_i) &= \nabla f(\mathbf{x}) + \nabla c(\mathbf{x})\boldsymbol{\lambda} - \mathbf{z} = 0 \text{ such that } \mathbf{x} = [\mathbf{x}, \mathbf{s}] \in \mathbb{R}^{n+m} \\ c(\mathbf{x}) &= 0 \text{ and } \mathbf{X}\mathbf{Z}\mathbf{e} - \mu\mathbf{e} = 0 \\ \text{or} \\ \nabla f(\mathbf{x}) + \nabla h(\mathbf{x})\boldsymbol{\lambda} - \mu \sum_{i=1}^m \frac{1}{g_i(\mathbf{x})} \nabla g_i(\mathbf{x}) &= 0 \\ h(\mathbf{x}) &= 0 \text{ and } g(\mathbf{x}) \geq 0 \end{aligned} \quad (2.4.7)$$

Where n is the size of the problem's initial variables, \mathbf{e} is a unit vector, \mathbf{X} is the diagonal matrix of x_i and \mathbf{Z} is the diagonal matrix of $z_i = \begin{cases} 0 & \text{for } i \leq n \\ \frac{\mu}{x_i} & \text{for } i > n \end{cases}$.

By applying the Newton-Raphson method to the KKT conditions, the following solution is developed:

$$\begin{pmatrix} \mathbf{W}_k + \mathbf{X}_k^{-1}\mathbf{Z}_k & \nabla c(\mathbf{x}_k) \\ \nabla c(\mathbf{x}_k)^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{d}_k^x \\ \mathbf{d}_k^z \end{pmatrix} = - \begin{pmatrix} \nabla f(\mathbf{x}_k) + \nabla c(\mathbf{x}_k)\boldsymbol{\lambda}_k \\ c(\mathbf{x}_k) \end{pmatrix} \quad (2.4.8)$$

$$\mathbf{d}_k^z = \mu_k \mathbf{X}_k^{-1} \mathbf{e} - \mathbf{z}_k - \mathbf{X}_k^{-1} \mathbf{Z}_k \mathbf{d}_k^x$$

Where \mathbf{x}_k is the point at the k^{th} iteration, \mathbf{W} is the Hessian of the barrier function, while the deltas are the search direction of the problem.

A step size α is chosen to determine the values of the following iterations. It can be calculated by decreasing the merit function or through filter methods. However, those two approaches will not be discussed further. It is often acceptable to assign α a value ≤ 1 .

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{d}_k^x \\ \boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \alpha_k \mathbf{d}_k^\lambda \\ \mathbf{z}_{k+1} &= \mathbf{z}_k + \alpha_k \mathbf{d}_k^z \end{aligned} \quad (2.4.9)$$

The iterations can be stopped once convergence is reached.

2.5 Sequential quadratic programming

A major advantage of the sequential quadratic programming (SQP) is that the algorithm is forgiving during the computation phase. The iterates \mathbf{x}_k are not obliged to be inside the feasible region if they converge correctly towards the optimiser of the problem.

The problem (2.4.1) will be reintroduced again. The solution begins by converting the objective function into its quadratic approximation and the constraints into their linear approximations.

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \mathbf{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \\ \text{such that} \quad & \begin{cases} g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \geq 0 \\ h(\mathbf{x}_k) + \nabla h(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0 \end{cases} \end{aligned} \quad (2.5.1)$$

It is recommended to transform all inequality constraints into equality ones. This can be done through the addition/subtraction of positive numbers known as the slack variables.

$$g(\mathbf{x}) - \mathbf{s} = 0 \quad (2.5.2)$$

Where \mathbf{s} is the slack variable.

By applying the linear approximation to (2.5.2), the problem becomes:

$$\begin{aligned} \min_{(x,s) \in \mathbb{R}^n \times \mathbb{R}_+^p} \quad & f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \mathbf{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \\ \text{such that} \quad & \begin{cases} g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) - (\mathbf{s} - \mathbf{s}_k) = 0 \\ h(\mathbf{x}_k) + \nabla h(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0 \end{cases} \end{aligned} \quad (2.5.3)$$

Where p is the number of inequality constraints $g(\mathbf{x})$ and the size of \mathbf{s} .

However, SQP is preferred to be used on problems of the form:

$$\begin{aligned} \min_{(x,s) \in \mathbb{R}^n \times \mathbb{R}_+^p} \quad & f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \mathbf{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \\ \text{such that} \quad & c(\mathbf{x}_k, \mathbf{s}_k) + \nabla c(\mathbf{x}_k, \mathbf{s}_k)(\mathbf{x} - \mathbf{x}_k) + \nabla c(\mathbf{x}_k, \mathbf{s}_k)(\mathbf{s} - \mathbf{s}_k) = 0 \end{aligned} \quad (2.5.4)$$

Where:

$$c(\mathbf{x}) = [h(\mathbf{x}); g(\mathbf{x}) - \mathbf{s}] \quad (2.5.5)$$

The KKT conditions are applied to (2.5.4) to solve the SQP subproblem.

$$\begin{aligned} \mathbf{H}_k \mathbf{d}_x + \nabla c(\mathbf{x}_k) \boldsymbol{\lambda}_{k+1} &= -\nabla f(\mathbf{x}_k) \\ \nabla h(\mathbf{x}_k) \mathbf{d}_x &= -h(\mathbf{x}_k) \\ \nabla g(\mathbf{x}_k) \mathbf{d}_x - \mathbf{d}_s &= -g(\mathbf{x}_k) \end{aligned} \quad (2.5.6)$$

Such that:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{d}_x \\ \mathbf{s}_{k+1} &= \mathbf{s}_k + \mathbf{d}_s\end{aligned}\tag{2.5.7}$$

\mathbf{s}_0 is calculated from \mathbf{x}_0 as follows:

$$\mathbf{s}_0 = g(\mathbf{x}_0)\tag{2.5.8}$$

The iterations continue until convergence is reached.

Convergence of the error

Convergence is best linked to consistency of the solution. A mathematical model is said to be convergent if after each iteration, the solution tends to a constant.

If an actual solution exists, then the error is the absolute difference of the actual solution to the current solution.

$$error = |x_k - x^*| \quad (2.6.1)$$

Where x_k is the solution at the k^{th} iteration and x^* is the actual unknown solution.

The error can be re-written as a geometric sequence of the form:

$$e_{k+1} = \mu e_k^\alpha \quad (2.6.2)$$

Where μ is a constant and α is the order of convergence.

In theory, the number of iterations required to reach an acceptable approximation of the actual solution decreases as the order of convergence increases.

To find the value of α , a logarithmic transformation is applied on the division of two consecutive errors:

$$\ln\left(\frac{e_{k+1}}{e_k}\right) = \ln\left(\frac{\mu e_k^\alpha}{\mu e_{k-1}^\alpha}\right) = \ln\left(\frac{e_k^\alpha}{e_{k-1}^\alpha}\right) = \alpha \ln\left(\frac{e_k}{e_{k-1}}\right) \quad (2.6.3)$$

Hence,

$$\alpha = \frac{\ln\left(\frac{e_{k+1}}{e_k}\right)}{\ln\left(\frac{e_k}{e_{k-1}}\right)} \quad (2.6.4)$$

μ can be found from (2.6.2):

$$\mu = \frac{e_{k+1}}{e_k^\alpha} \quad (2.6.5)$$

α needs at least three iterations to be calculated from which afterwards, μ can be computed. The values of α and μ will differ slightly with each iteration. However, they can be rounded to approximate reasonable values. Usually, the values of α and μ are standardised and depend on the method of iteration used. For example, the Bisection Method has an order of convergence of 1 and

$\mu = 1/2$ $\left(|e_{k+1}| = \frac{1}{2}|e_k|\right)$, since the value of the next iteration is the mean value of the solution interval.

It is possible to estimate the number of iterations needed to reach the desirable solution if and only if the model is converging; meaning, the error is decreasing with each iteration.

Since the error is a geometric sequence, it can be written as a function of the first error:

$$e_k = (\mu e_0^\alpha)^k \quad (2.6.6)$$

Where the first error depends strictly on the initial value. Which is why, choosing an initial value in the region of the solution can greatly decrease the computation time.

The iterations stop when the error is less than or equal to a predefined tolerance ε .

$$(\mu e_0^\alpha)^n = \varepsilon \quad (2.6.7)$$

Where n is the final iteration.

By applying the logarithmic transformation, like (2.6.3), the expected number of iterations can be found.

$$n = \frac{\log \varepsilon}{\log \mu e_0^\alpha} \quad (2.6.8)$$

3 Optimisation Panel

The Optimisation Panel (OP) is a GUI application exploiting the MATLAB function `fmincon` to solve optimisation problems. The GUI is aimed towards the simple user with mostly educational-oriented problems, and the more sophisticated user riddled with complex experimental problems.

3.1 Fmincon function

`Fmincon` is a nonlinear constrained problem optimisation solver in MATLAB. It finds the minimum of a multivariable constrained objective function. It has the following general syntax:

$$\mathbf{x} = \text{fmincon} (f, \mathbf{x0}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}, \mathbf{lb}, \mathbf{ub}, \text{nonlcon}, \text{options}) \quad (3.1.1)$$

Each term should be represented as in the above order. If any term is unspecified, it cannot be removed if a following term is used. It should be set to `[]`.

Some terms can be omitted from the general expression and more simplified versions can be used depending on the nature of the constraints. The most simplified syntax of the `fmincon` function is:

$$\mathbf{x} = \text{fmincon} (f, \mathbf{x0}, \mathbf{A}, \mathbf{b}) \quad (3.1.2)$$

Where f is the objective function to be optimised, \mathbf{x} is the vectorial solution, and $\mathbf{x0}$ is the vector of initial conditions/values.

The solution should satisfy the following linear inequality:

$$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \quad (3.1.3)$$

Where \mathbf{A} is the linear inequality matrix, and \mathbf{b} is the inequality vector.

The above syntax can be expanded by adding equality restraints:

$$\mathbf{x} = \text{fmincon} (f, \mathbf{x0}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}) \quad (3.1.4)$$

Such that:

$$\mathbf{Aeq} \cdot \mathbf{x} = \mathbf{beq} \quad (3.1.5)$$

Where \mathbf{Aeq} is the linear equality matrix and \mathbf{beq} is the equality vector.

A more expanded version includes the upper and lower bound of each variable in \mathbf{x} .

$$\mathbf{x} = \text{fmincon} (f, \mathbf{x0}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}, \mathbf{lb}, \mathbf{ub}) \quad (3.1.6)$$

Where \mathbf{lb} and \mathbf{ub} are respectively the lower bound and upper bound vectors.

However, the boundary constraints can be integrated into the linear inequality matrix \mathbf{A} and hence, syntax (2.2.4) can still be used.

Non-linear conditions can be applied by using `nonlcon`.

$$\mathbf{x} = \text{fmincon} (f, \mathbf{x0}, \mathbf{A}, \mathbf{b}, \mathbf{Aeq}, \mathbf{beq}, \mathbf{lb}, \mathbf{ub}, \text{nonlcon}) \quad (3.1.7)$$

Through a function:

$$\begin{aligned} &\text{function } [c, \text{ceq}] = g(\mathbf{x}) \\ &\quad c = \\ &\quad \text{ceq} = \\ &\text{end} \\ &\text{nonlcon} = @g; \end{aligned} \quad (3.1.8)$$

Such that:

$$c(\mathbf{x}) \leq 0 \quad (3.1.9)$$

$$\text{ceq}(\mathbf{x}) = 0 \quad (3.1.10)$$

Where $c(\mathbf{x})$ and $\text{ceq}(\mathbf{x})$ are respectively the nonlinear inequality function and nonlinear equality function to be satisfied for \mathbf{x} .

A vectorial form can be used to return other relevant values.

$$[\mathbf{x}, \text{fval}] = \text{fmincon}(_) \quad (3.1.11)$$

Where fval is the value of the objective function at the solution \mathbf{x} for any used syntax.

The options term is set using the `optimoptions` function and it is the last possible term to add to the `fmincon` function syntax:

$$\text{options} = \text{optimoptions}('fmincon', 'Display', '___', 'Algorithm', '___'); \quad (3.1.11)$$

More attributes can be added to the above function. Each has a default value.

In summary, below are all the components of a `fmincon` optimisation problem:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \\ &\mathbf{Aeq} \cdot \mathbf{x} = \mathbf{beq} \\ &c(\mathbf{x}) \leq 0 \\ &\text{ceq}(\mathbf{x}) = 0 \\ &\mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{aligned} \quad (3.1.12)$$

3.2 Graphical user interphase application

A Graphical User Interphase (GUI) is an interactive visual system for computer software. A GUI displays a screen that conveys information and requests certain inputs from or actions to be taken by the user by interacting with the objects on the screen. These objects are plentiful with a specific purpose. They can be in the form of a pushbutton that initiates a command if pressed, an edit box that evaluates the data written inside it, an option list, or etc. Users can interact with the GUI through pointing devices such as a mouse, keyboard, or touch screen by moving over the GUI component and pressing it.

MATLAB offers three ways to create an app:

- Live Editor converts a script into a simple app where users are given the choice to modify variables.
- App Designer or the GUIDE tool uses the drag and drop mechanism to customise the appearance of the GUI while the code is developed in the MATLAB editor.
- The programmer can manually write the code by using functions to define the layout and behaviour of the GUI.

For more added control, the GUI is created programmatically. The only functions used are the ones compatible with the 2016b version of MATLAB. Hence, uifigure-based [10] functions are omitted. Most of the visuals on the screen can be made by means of the uicontrol [11] function which has the following syntax:

$$uicontrol(parent, Name, Value) \quad (3.2.1)$$

Its parent is a figure made from the figure function. If the parent is not specified, by default, a new figure is constructed to contain the uicontrol component. It can be written as is or it can be assigned as an input to a variable. This allows to manipulate the element after further coding.

Name refers to the attribute given to the element and Value is the output of the defined attribute. Some Name-Value pairs can also be layout features to control the appearance of the UI element such as font size and colour. The position attribute (Name)'] of the UI element can be assigned through the position vector (Value).

$$c = uicontrol('Position', [left bottom width height]) \quad (3.2.2)$$

Left and bottom indicate the directional distance from the figure frame. While width and height indicate the size of the UI component.

The following list represents all the possible Values for the style attribute:

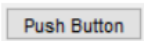
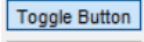
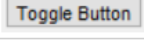
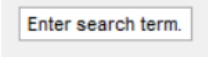
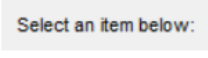

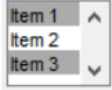
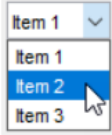
Style Property Value	Example
'pushbutton'	
'togglebutton'	 
'checkbox'	<input checked="" type="checkbox"/> Check Box <input type="checkbox"/> Check Box
'radiobutton'	<input checked="" type="radio"/> Radio Button <input type="radio"/> Radio Button
'edit'	
'text'	
'slider'	
'listbox'	
'popupmenu'	

Figure 3.2.1 – Style – Value pairs [11]

A table-styled element was constructed independently through the `uitable` [12] function because this value is not supported by `uicontrol`. It has the same syntax as `uicontrol` but with additional row and column attributes.

$$\text{uitable}(\text{parent}, \text{Name}, \text{Value}) \quad (3.2.3)$$

The value of the data attribute is either a numeric, logical, string, or cell array. The array dimensions define the number of columns and rows of the table. Each table cell is occupied by its respective array value. The rows and columns of the UI table can be assigned names/values as well by using the `RowName` and `ColumnName` attributes respectively.

3.3 Optimisation panel guide

The screenshot shows a window titled "Optimisation Panel" with a version number "ver. 2022-05-15". The interface is divided into several sections:

- Decision Variables:** Includes a "Load" button, a table with columns "Variable", "Description", "Unit", and "Lo", and buttons "Edit", "Delete", and "Add". Below the table are input fields for "Lower Bound", "Initial Value", and "Upper Bound".
- Objective Function:** Features a dropdown menu set to "Linear", checkboxes for "Minimise" and "Maximise", and a "Load File" button.
- Constraint Relations:** Includes input fields for "Linear Equality", "NonLinear", and "Linear Inequality", along with a "Load File" button.
- Bottom Section:** Contains a "Load Problem" button, checkboxes for "Enable Plot" and "Enable Save", and a "Find Optimum" button.

Figure 3.3.1 – Optimisation Panel

Four parts need to be specified to solve an optimisation problem: the decision variables, the constraints, the objective function, and the objective of the optimisation.

By referring to figure (3.3.2), each variable has six components and should be uniquely defined. The Add button is enabled after the variable has been named. The description of the variable can be left empty. However, if the variable is dimensionless, the unit should be assigned as '-'. The lower and upper bounds represent the range of the variable which are numeric values. If the variable is unbounded, the value should be set to $-\infty$ or $+\infty$. The initial value should lay within the range as it is the value taken for the first iteration.

Press the *Add* button to save the variable. If it is correctly defined, it will be added to the table, else, the user will receive an error message on the wrongly inputted field.

Decision Variables

Type name

Variable

Description

Type description

Unit

Type unit

Lower Bound

Initial Value

Upper Bound

Input value

Add

Press

Figure 3.3.2 – Decision Variable_Add

	Variable	Description	Unit	Lower Bound	Initial Value	Upper Bound
1	k	Elasticity	N/mm	-3	2	20
2	Fp	Force	N	-Inf	123	400
3	x1		mm	-10	1	10
4	x2		mm	5	8	20
5	r	ratio	-	0.9	1.05	1.3
< >						

Figure 3.3.3 – Decision Variable Table Example

The user is given the choice to upload the completed list of the decision variables from an independent .m file by pressing the *Load* button (figure 3.3.4) adjacent to the table and selecting a file.

Decision Variables

Load

Press

Figure 3.3.4 – Decision Variable_Load

The user is also given the ability to edit a variable already inserted into the table by selecting the cell whose contents they wish to modify, typing the new input in its corresponding text box, and pressing the *Edit* button (figure 3.3.5).

If the user wishes to remove a variable from the table, they can select any cell from the row containing the variable and pressing the *Delete* button (figure 3.3.6).

	Variable	Description	Unit	Lower Bound	Initial Value	Upper Bound
1	x		-	0	7	23

Decision Variables

Variable Description Unit Lower Bound Initial Value Upper Bound

Lower Bound Initial Value Upper Bound

Press

Edit

Figure 3.3.5 – Decision Variable_Edit

Press

Delete

	Variable	Description	Unit	Lower Bound	Initial Value	Upper Bound
1	x		cm	0	7	23
2	k		-	1.01	2	5.34
3	F	force	kN	20	200	400

< >

Figure 3.3.6 – Decision Variable_Delete

The objective of the optimisation can be either a minimisation or a maximisation, which can be chosen by checking either the *Minimise* or *Maximise* checkbox (figure 3.3.7). The objective function is constructed by selecting one of the four options in the popup menu (figure 3.3.7).

Objective Function

☐ Minimise

☐ Maximise

Linear

Linear

Quadratic

nth Polynomial

Other

Figure 3.3.7 – Objective Function Options

The Linear option requires the user to input a numeric vector whose components are the coefficients of the decision variables in the order they are introduced in the table. The first coefficient, however, should be the solitary term of the equation. The objective function $f(\mathbf{x})$ has the form of equation (1.1.1.1).

Figure 3.3.8 – Option 1_Linear

Such that,

$$f(\mathbf{x}) = a_0 + \sum_{i=1}^n a_i x_i \quad (3.3.1)$$

$$\text{Coefficients Vector} = [a_0, a_1, \dots, a_p, \dots, a_n]$$

Where n is the number of decision variables and a_i is the coefficient of the i^{th} variable.

The Quadratic option requires the user to input a \mathbf{H} matrix and an \mathbf{A} vector in addition to the solitary term.

Figure 3.3.9 – Option 2_Quadratic

The objective function has the form of equation (1.1.2.1). \mathbf{A} is a horizontal vector containing the coefficients of each first-degree variable. \mathbf{H} is a square matrix where the diagonal elements are the coefficients of the squared decision variables. For representative purposes, the following equation will be considered as the expanded form of (1.1.2.1).

$$f(\mathbf{x}) = a_0 + \sum_{i=1}^n \left(a_{ii} x_i^2 + \sum_{j=2}^{n-1} a_{ij} x_i x_j \right) + \sum_{i=1}^n a_i x_i = a_0 + \mathbf{X}^T \mathbf{H} \mathbf{X} + \mathbf{A} \mathbf{X} \quad (3.3.2)$$

Where n is the number of decision variables, a_i is the coefficient of the i^{th} variable, a_{ii} is the coefficient of the squared i^{th} variable, and a_{ij} is the coefficient of the product of the i^{th} and j^{th} variables.

The \mathbf{H} matrix can be constructed in different ways as illustrated below since all forms produce the same result.

$$H = \begin{bmatrix} a_{11} & 0.5a_{12} & 0.5a_{13} \\ 0.5a_{12} & a_{22} & 0.5a_{23} \\ 0.5a_{13} & 0.5a_{23} & a_{33} \end{bmatrix} \Leftrightarrow \begin{bmatrix} a_{11} & 0 & 0 \\ a_{12} & a_{22} & 0 \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Leftrightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} \quad (3.3.3)$$

$$H = [a_{11}, a_{12}, a_{13}; 0, a_{22}, a_{23}; 0, 0, a_{33}]$$

The nth Polynomial requires the user to input the solitary term and a matrix whose dimension represents the highest degree in the equation

Figure 3.3.10 – Option 3_nth Polynomial

The objective function is composed only of the summation of powered variables. Thus, it has the following form:

$$f(\mathbf{x}) = a_0 + \sum_{i=1}^p \sum_{j=1}^n a_{ij} x_i^j \quad (3.3.4)$$

Where p is the number of decision variables, n is the degree of the equation, and a_{ij} is the coefficient of the i^{th} variable to the power j .

The array is a $(p \times n)$ matrix constructed such that each row contains all the coefficients of a variable starting from the lowest degree to the highest one.

The Other enables the Load File to work, and the user can share an independent .m file.

Figure 3.3.11 – Option 4_Other

The user can input a linear equality matrix and a linear inequality matrix. Each matrix is $(p \times n + 1)$, where p is the number of equalities or inequalities, and n is the number of decision variables. The first element of each row is the solitary term of the constraint while the rest are the coefficients of each variable. Each constraint equation is either equal to the solitary term in case of equality or less than the solitary term in case of inequality.

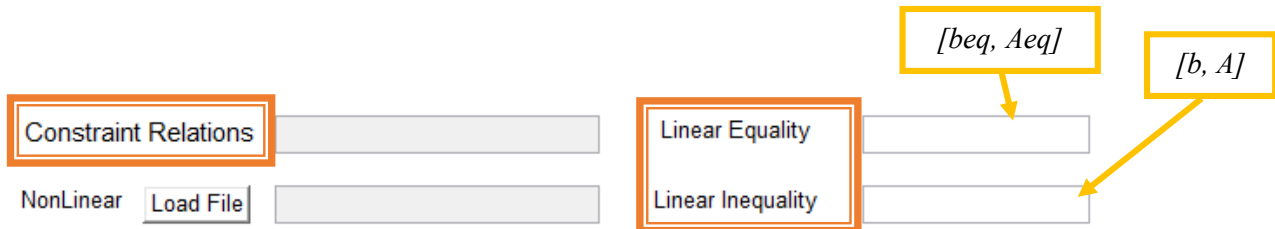


Figure 3.3.12 – Constraints _ Linear Equality and Inequality

Such that,

$$\begin{aligned} \mathbf{Aeq} \cdot \mathbf{x} &= \mathbf{beq} \\ \mathbf{A} \cdot \mathbf{x} &\leq \mathbf{b} \end{aligned} \quad (3.3.5)$$

Where \mathbf{Aeq} and \mathbf{A} are the equality and inequality linear matrices and \mathbf{beq} and \mathbf{b} are the equality and inequality vectors respectively.

$$\begin{aligned} size(\mathbf{A}, 1) &= size(\mathbf{b}, 1) \\ size(\mathbf{Aeq}, 1) &= size(\mathbf{beq}, 1) \\ size(\mathbf{Aeq}, 2) &= size(\mathbf{A}, 2) = n \\ size(\mathbf{b}, 2) &= size(\mathbf{beq}, 2) = 1 \end{aligned} \quad (3.3.6)$$

Where n is the number of decision variables.

The non-linear constraints, if they exist, can be loaded (figure 3.3.13) onto the GUI as well from an independent .m file. If the functions are written correctly, they will be displayed on the adjacent text box.

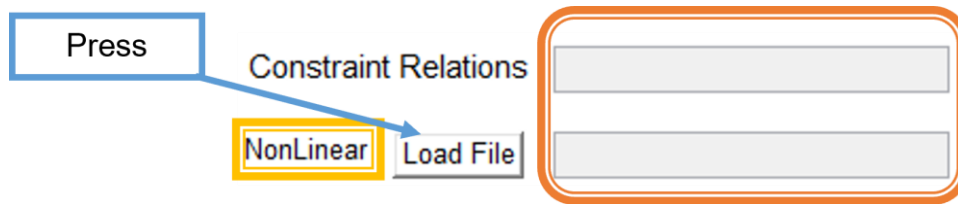


Figure 3.3.13 – Constraints _ Non-linear

Once all terms of the problem have been specified, the user can choose to enable the *plot* and/or *save* options at the bottom of the GUI (figure 3.3.14) before pressing the *Find Optimum* button (figure 3.3.15). If enabled, the program will display three or four figures with graphs of the values of the objective function, the variables, and the non-linear constraints at each iteration. The fourth figure will only be plotted if a normalisation procedure is performed, which depends on the boundaries of the decision variables introduced. The values of the objective function and the variables will be saved in a .txt file after each iteration as well.

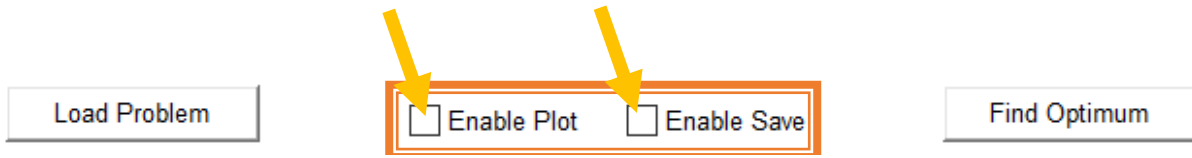


Figure 3.3.14 – Plot and Save



Figure 3.3.15 – Final Step

The user can load/input the decision variables, objective function, and constraints independently or all at the same time from a single .m file by pressing the *Load Problem* button (figure 3.3.16).



Figure 3.3.16 – Load Problem

However, for any of the *Load* buttons to work, the user should follow the standard format for the .m file.

The decision variables are assigned to a $(n \times 6)$ cell called `Optim.x` in which n is the number of decision variables. The six components of each decision variable are arranged as follows:

$$\text{Optim.x} = \{\text{'String'}, \text{'String'}, \text{'String'}, \text{Value}, \text{Value}, \text{Value}; \dots\} \quad (3.3.7)$$

Where the first three components are strings defining the decision variable's name, description, and unit in this order. The last three components are numerical and represent the lower bound, initial value, and upper bound of the decision variable respectively.

Each decision variable follows the restrictions mentioned previously. If the cell is written correctly, the decision variables will be displayed inside the table. Otherwise, an error message will be displayed instead.

The objective function is assigned to a variable called `Optim.obj_func`. The constraints are assigned to variables as follows:

- `Optim.A` and `Optim.b` are numeric and are the linear inequality matrix and the linear inequality vector respectively.
- `Optim.Aeq` and `Optim.beq` are numeric and are the linear equality matrix and the linear equality vector respectively.
- `Optim.c` is a $(1 \times n_c)$ cell of function handles where n_c is the number of non-linear inequality constraints.
- `Optim.ceq` is a $(1 \times n_{ceq})$ cell of function handles where n_{ceq} is the number of non-linear equality constraints.

The linear constraint equations can be omitted from their respective matrices and vectors and added to the non-linear ones. Any of the above constraint variables can be left empty or unassigned. If any variable is missing, it will be considered empty.

The objective of the problem is defined through the variable `Optim.Opt_case`. It can be assigned only two values: 'Min' or 'Max'.

The user can also define the option setting of `fmincon` through the variable `Optim.Options`. the syntax is defined in (3.1.11). If it is not introduced, `fmincon` will work based on its default settings.

3.4 External code examples

A variant of the Rosenbrock function will be used to test the functionality of the GUI *Optimisation Panel*. Three different formats of the same function will be applied as .m files to help the user understand the versatility of the `Optim.obj_func` options.

3.4.1 Rosenbrock function

The Rosenbrock function is a valley function whose minimum is known. It is used as a test to study the performance of optimisation algorithms. It has the following general form:

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} \left[b(x_{i+1} - x_i^2)^2 + (a - x_i)^2 \right] \quad (3.4.1.1)$$

Where N is a natural number, $\mathbf{x} \in \mathbb{R}^N$ and $a, b \in \mathbb{R}$.

The function is made up of the summation of $N-1$ subfunctions $f(x_i, x_{i+1})$, whose minimum is easily and directly calculated by setting the gradient of the subfunction to zero.

$$\nabla f(x_i, x_{i+1}) = \left[\frac{\partial f}{\partial x_i}, \frac{\partial f}{\partial x_{i+1}} \right] = \left[2(x_i - a) - 4bx_i(x_{i+1} - x_i^2), 2b(x_{i+1} - x_i^2) \right] = 0 \quad (3.4.1.2)$$

The minimum of the subfunction is given by:

$$\min f(x_i, x_{i+1}) = f(a, a^2) = 0 \quad (3.4.1.3)$$

Since the derivative of the sum is the sum of the derivative, then the partial derivatives of each subfunction are added together to form the gradient of the function. Each subfunction has only two partial derivatives that are not null, which correspond to their variables. Except for the first and last variables (x_1 and x_N), each two consecutive subfunctions have exactly one variable in common. Hence, the syntax of the gradient element at any partial derivative is given as:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_p}, \dots, \frac{\partial f}{\partial x_N} \right] = \left[2(x_1 - a) - 4bx_1(x_2 - x_1^2) \dots 2b(x_p - x_{p-1}^2) + 2(x_p - a) - 4bx_p(x_{p+1} - x_p^2) \dots 2b(x_N - x_{N-1}^2) \right] \quad (3.4.1.4)$$

Where $1 < p < N$.

A solution to (3.4.4) is $\mathbf{x} = [a \cdots a \cdots a^2]$. However, it is not unique. Multiple optima may surface as more variables are introduced to the equation. Nevertheless, $\mathbf{x} = [a \cdots a \cdots a^2]$ is always a global minimiser of the Rosenbrock function. Only for $a = 1$, the value of the global minimum is zero for any N . While for $a \neq 1$, the minimum moves further away from zero as N increases.

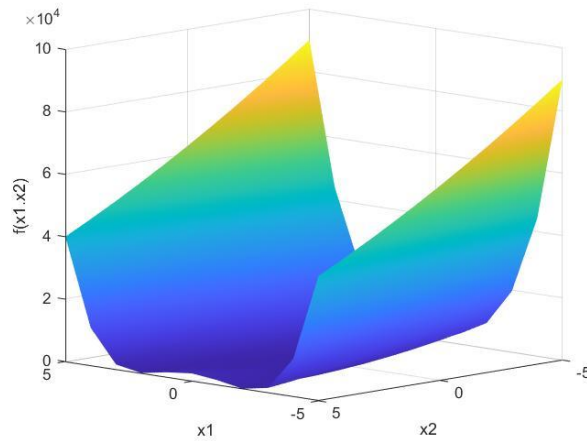


Figure 3.4.1.1 – Rosenbrock function for $n = 2$, $a = 1$ and $b = 100$

3.4.2 Handle function

A handle function is created, in MATLAB, by assigning $@(\mathbf{x})$ to the beginning of the anonymous function. \mathbf{x} is the vector of variables used inside the equation. Taking the objective function to optimise, the decision variables introduced must be written as $\mathbf{x}(i)$ s.

```

1 - Optim.x = {
2 -     'x1', 'x_1', '-', -10, 2, 10
3 -     'x2', 'x_2', '-', -20, -4, 20
4 - };
5 - Optim.obj_func = @(x) (1-x(1))^2+100*(x(2)-x(1)^2)^2;
6 - Optim.c = { @(x) (x(1)^2 + x(2)^2 - 1) };
7 - Optim.ceq = [];
8 - Optim.Opt_case = 'Min';

```

Figure 3.4.2.1 – Handle function .m file example

The above figure displays the problem and the anonymous function to optimise. The objective function is a 2-dimensional Rosenbrock function.

The minimum value of all the iterations yielded the following results:

Table 3.4.2.1 – Results of optimisation.

x	x1	x2
	0.78641	0.6177
f_{\min}	0.045675	

The following figures represent the convergence of the problem towards the optimal solution.

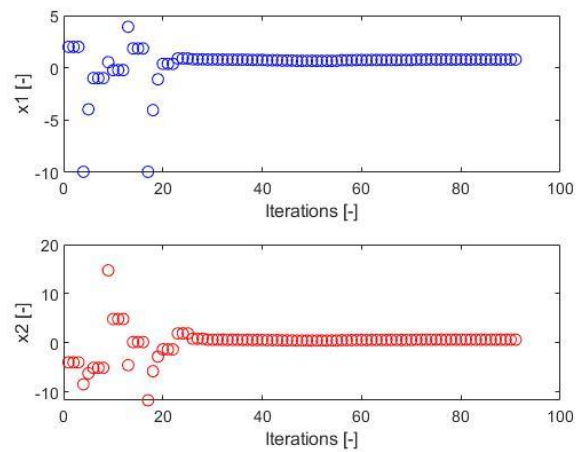


Figure 3.4.2.2 – Decision variables

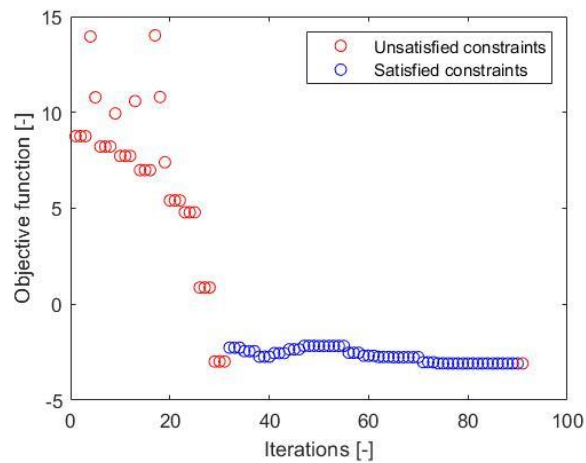


Figure 3.4.2.3 – Objective function in log scale

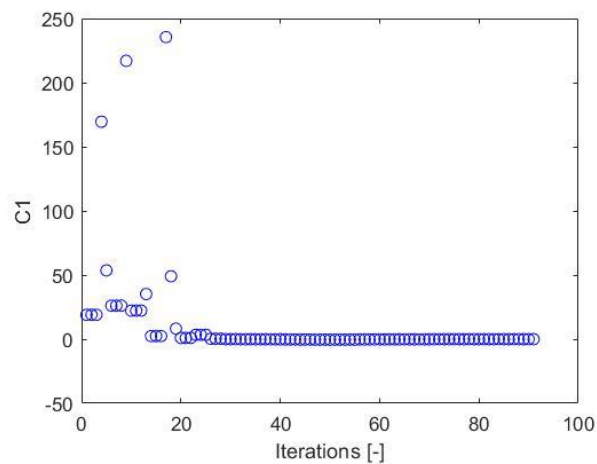


Figure 3.4.2.4 – Constraint function

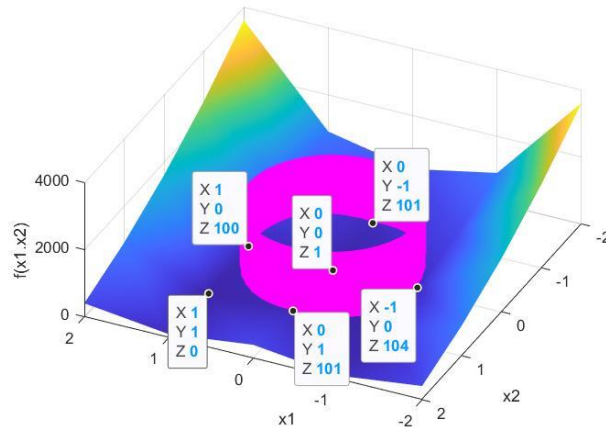


Figure 3.4.2.5 – Optimisation problem close-up

As observed in figure 3.4.2.5, the function is mostly decreasing in the quadrant connecting the centre of the feasible region and the unconstrained minimum of the objective function $[1, 1]$. Hence, the optimal solution to the problem is along a contour heading to the optimum at the boundary of the feasible region, as illustrated in figure 3.4.2.6.

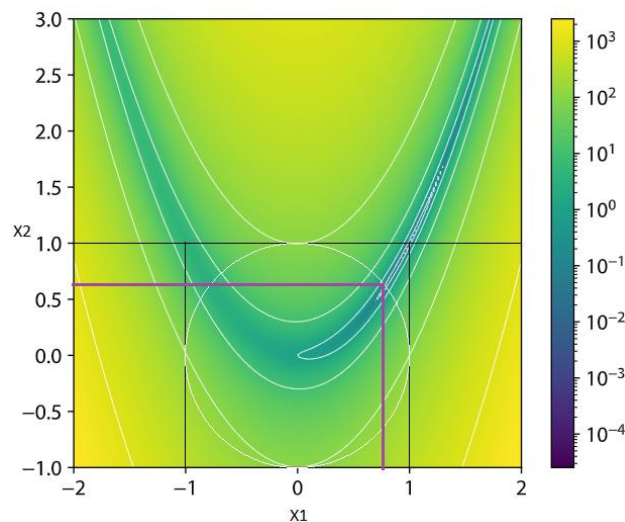


Figure 3.4.2.6 – Graphical representation of the optimisation problem

The algorithm is an iterative procedure, and the results are approximants of the actual solution.

3.4.3 External function

MATLAB allows the user to write and execute their own functions by using the `function` command. Each function is assigned a function name and a set of inputs enclosed in parenthesis. In this example, a N -dimensional Rosenbrock function is constructed as the objective function and loaded into the GUI as an external function.

```

1 - Optim.x = {
2 -     'x1', 'x_1', '-', -10, 2, 10
3 -     'x2', 'x_2', '-', -20, -4, 20
4 - };
5 - Optim.obj_func = @(x)Rosenbrock(x);
6 - Optim.c = {@(x)(x(1)^2 +x(2)^2 - 1)};
7 - Optim.ceq = [];
8 - Optim.Opt_case = 'Min';

```

```

1 - function [result] = Rosenbrock(x)
2 -     b = 100;
3 -     a = 1;
4 -     f = 0;
5 -     for p = 1:size(x,1)-1
6 -         f = f + b*(x(p+1)-x(p)^2)^2+(a-x(p))^2;
7 -     end
8 -     result = f;

```

Figure 3.4.3.1 – External function .m file example

The previous results were replicated, and the solution in table 3.4.2.1 was computed again as were the graphs and the number of iterations.

3.4.4 External application

The code in figure 3.4.4.1 was converted into an application using the application compiler feature in MATLAB. The application was then loaded into the GUI from a .m file as in figure 3.4.4.2.

```

1 - function status = Rosenbrock2(inputFile,outputFile)
2 -     status = -1;
3 -     x = dlmread(inputFile);
4 -     x = x(:);
5 -
6 -     a = 1;
7 -     b = 100;
8 -
9 -     f = 0;
10 -    for count=1:size(x,1)-1
11 -        f = f + b*(x(count+1)-x(count)^2)^2+(a-x(count))^2;
12 -    end
13 -
14 -    try
15 -        fout = fopen(outputFile,'w');
16 -        status = 0;
17 -    end
18 -
19 -    try
20 -        fprintf(fout,'%f',f);
21 -        status = 0;
22 -    end
23 -    fclose(fout);

```

Figure 3.4.4.1 – Rosenbrock external application

```

1 - Optim.x = {
2 -     'x1', 'x_1', '-', -10, 2, 10
3 -     'x2', 'x_2', '-', -20, -4, 20
4 - };
5 - Optim.obj_func = 'C:\Users\USER\Desktop\MATLAB Projects\Rosenbrock\application\Rosenbrock.exe';
6 - Optim.c = {@(x)(x(1)^2 +x(2)^2 -1)};
7 - Optim.ceq = [];
8 - Optim.Opt_case = 'Min';

```

Figure 3.4.4.2 – Rosenbrock external application .m file example

The minimum value of all the iterations yielded the following results:

Table 3.4.4.1 – Results of optimisation.

x	x1	x2
	-0.03082	-0.010556
f_{\min}	1.0758	

The following figures represent the convergence of the problem towards the optimal solution.

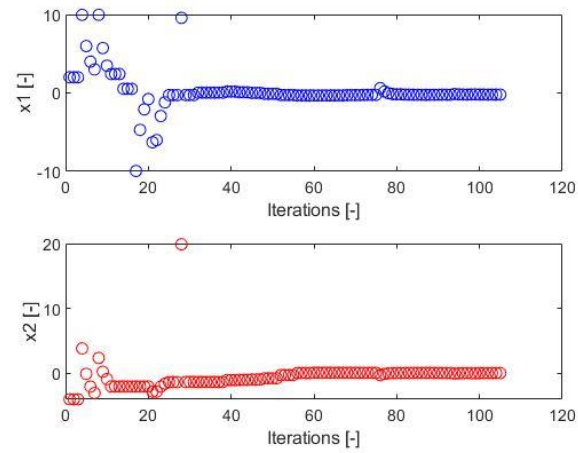


Figure 3.4.2.2 – Decision variables

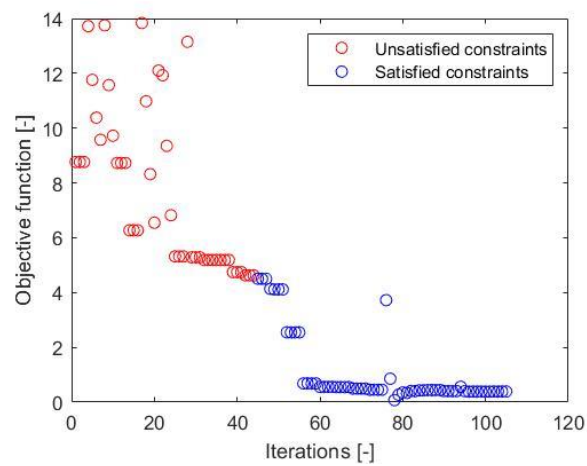


Figure 3.4.2.3 – Objective function in log scale

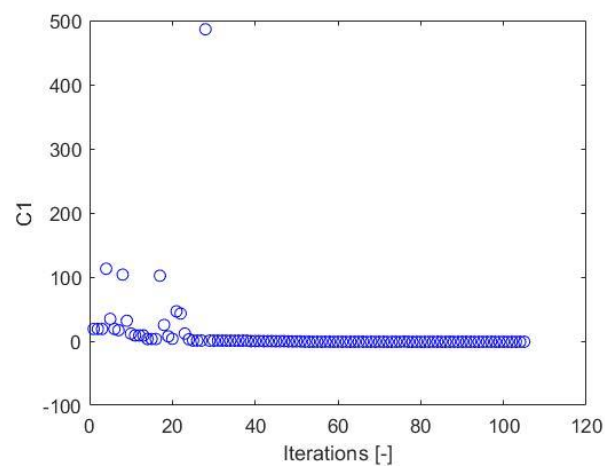


Figure 3.4.2.4 – Constraint function

The same problem was introduced in all three cases. So, theoretically, all the solutions should be identical, given that the same procedure was performed in all the examples. This applies to the first two cases which confirms the repeatability of `fmincon`. However, the results slightly differ in the third. The difference is caused from loading the values of the decision variables introduced into the `Input.dat` file before each iteration. The application receives the values from the `Input.dat` file and not directly from the `fmincon` function evaluation. This slight difference caused the number of iterations and the approximation error to increase.

4. Case studies

The case study will revolve around a study that was previously performed by E. Bonisoli, A.D. Vella, and S. Venturini titled: “Uncertainty Effects on Bike Spoke Wheel Modal Behaviour”. It discusses how to achieve a proper modal behaviour of the bike spoke wheel under the effect of structural and material uncertainties and ununiform mass and force distribution. An optimisation problem was attempted to minimise the relative error between the experimental and analytical modes of a spoke bike wheel under the effect of uncertain parameters.

Modal analysis revolves around determining the natural frequencies and their respective mode shapes of a structure under free vibration. The natural frequencies are calculated from the overall mass of the structure and its stiffness. Afterwards, the Modal Assurance Criteria (MAC) is used to determine the similarity of two mode shapes. If the mode shapes are identical, then the value of the MAC is 1. The MAC is mostly sensitive to large differences in the mode shapes unlike small differences, which makes it a good indicator to study the correlation between mode shapes.

$$MAC = \frac{[\Phi_j^T \Phi_k]}{[\Phi_j^T \Phi_j][\Phi_k^T \Phi_k]} \quad (4.1)$$

Where Φ_j and Φ_k are two eigenvalue vectors.

The eigenvalue analysis provides dynamic properties of the structure by solving the characteristic equation of the system:

$$(\mathbf{K} - w^2 \mathbf{M}) \Phi = 0 \quad (4.2)$$

Where \mathbf{K} and \mathbf{M} are the stiffness and mass matrices of the system respectively and w^2 is the eigenvalue also referred to as λ .

The eigenvalues are found by solving the determinant of equation (4.2).

$$\det[\mathbf{K} - \lambda \mathbf{M}] = 0 \quad (4.3)$$

Such that:

$$w = \sqrt{\lambda} \quad (4.4)$$

Where w is the natural frequency. The smallest value of w is called the fundamental frequency.

Now, equation (4.2) can be solved for Φ for each eigenvalue. Φ is also known as the mode displacements. After plotting the displacements, the mode shape (the vibratory motion) can be observed by alternating the nodal positions between the initial position (usually at rest) and the mode displacement positions. The size of Φ depends on the degree of freedom of the structure (number of nodes), as it represents the displacement of each node in the structure.

In the case study, the *MACW2* was used instead of *MAC* to include the eigenvalue contribution to the correlation index.

$$MACW2 = \frac{[\Phi_j^T \Phi_k]}{[\Phi_j^T \Phi_j][\Phi_k^T \Phi_k]} e^{-\sqrt{2} \frac{|w_j^2 - w_k^2|}{w_j^2 + w_k^2}} \quad (4.5)$$

Like the *MAC*, the *MACW2* gives a value of 1 for identical modes. Hence, the summation of *n* *MACW2*s of identical modes should yield a value of *n*. Therefore, equation (4.6) has been chosen as an objective function for optimisation, which is the relative error of *MACW2* and the mass.

$$\begin{aligned} \text{Objective Function} &= RE(MACW2) + RE(m) \\ RE(MACW2) &= \frac{n - \sum_{r=1}^n \max(MACW2_{r,k} \mid \forall k)}{n} \\ RE(m) &= \frac{|m_{ref} - m|}{m_{ref}} \end{aligned} \quad (4.6)$$

Where *m* and *m_{ref}* are the modified and reference masses respectively and *n* is the number of experimental studied modes.

A preliminary Finite Element (FE) model with the properties of table 4.1 has been established to determine the mode shapes. However, as evident from figure 4.1, it tends to overestimate the frequencies of the experimental mode shapes, making the model more rigid than it is. The optimisation problem aims to find the best choice of parameters that allow to construct a FE model that can optimally translate analytically the experimental mode shapes.

Table 4.1 – Geometric and material properties of FE model. [19]

Component	Parameter	Value
Rim	<i>E</i> , Young Modulus [GPa]	70
	ρ , density [kg/m ³]	2700
	<i>t</i> , thickness [mm]	0.8 – 2.0
Hub	<i>E</i> , Young Modulus [GPa]	70
	ρ , density [kg/m ³]	2700
	<i>t</i> , thickness [mm]	3 - 8
Spokes	<i>E</i> , Young Modulus [GPa]	210
	ρ , density [kg/m ³]	7800
	\varnothing , diameter [mm]	2
	<i>N</i> , pretension [N]	1200
Gear	<i>E</i> , Young Modulus [GPa]	210
	ρ , density [kg/m ³]	7800

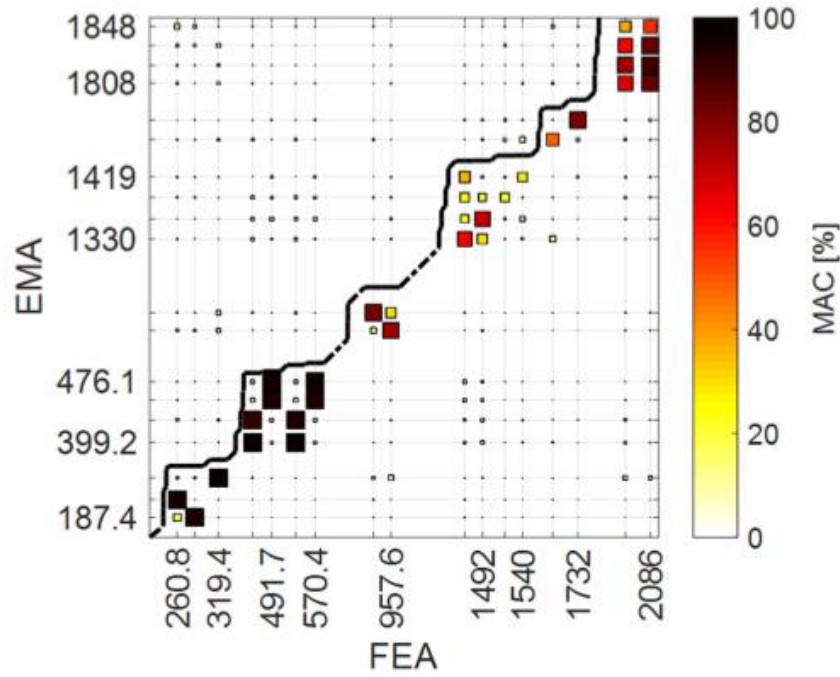


Figure 4.1 – MAC between Experimental Modal Analysis (EMA) and preliminary FE model [19]

The intensity of the colour of the points in figure 4.1 represents the correlation of the FEA and the EMA. Thus, black is the best correlation corresponding to a MAC of 1, while white is least correlated corresponding to a MAC of 0. An ideal MAC figure is diagonal, matching each analytical mode to its experimental mode. However, the above figure exhibits relations between experimental and analytical modes that should not exist. These unwanted relations appear in areas of the graph with concentrated spots and scatter points.

For the optimisation process, the 7 experimental mode shapes from figure 4.2 have been chosen and the starting mass has been set to 1.5774 kg.

Table 4.2 represents the initial and optimal values of the design parameters which were achieved in the optimisation study using *fmincon*.

Table 4.2 – Design parameters with boundaries [19]

Parameter	Lower bound	Starting point	Upper bound	Optimal
E_{rim} , Young Modulus [GPa]	35	52.5	105	68.30
ρ_{rim} , density [kg/m ³]	1350	2025	4050	3589
$t_{mult,rim}$, thickness multiplier [-]	0.1	0.75	10	1.497
E_{spokes} , Young Modulus [GPa]	105	157.5	315	163.3
ρ_{spokes} , density [kg/m ³]	3900	5850	11700	7204
ϕ_{spokes} , diameter [mm]	1	1.5	3	1.617
N , pretension [N]	0	900	3000	6010
ρ_{gear} , density [kg/m ³]	3900	5850	11700	7800
ρ_{hub} , density [kg/m ³]	1350	2025	4050	2302

Figure 4.2 compares the previous MAC calculated using the preliminary FE model with the new MAC calculated using the tuned FE model based on the results of the optimisation problem.

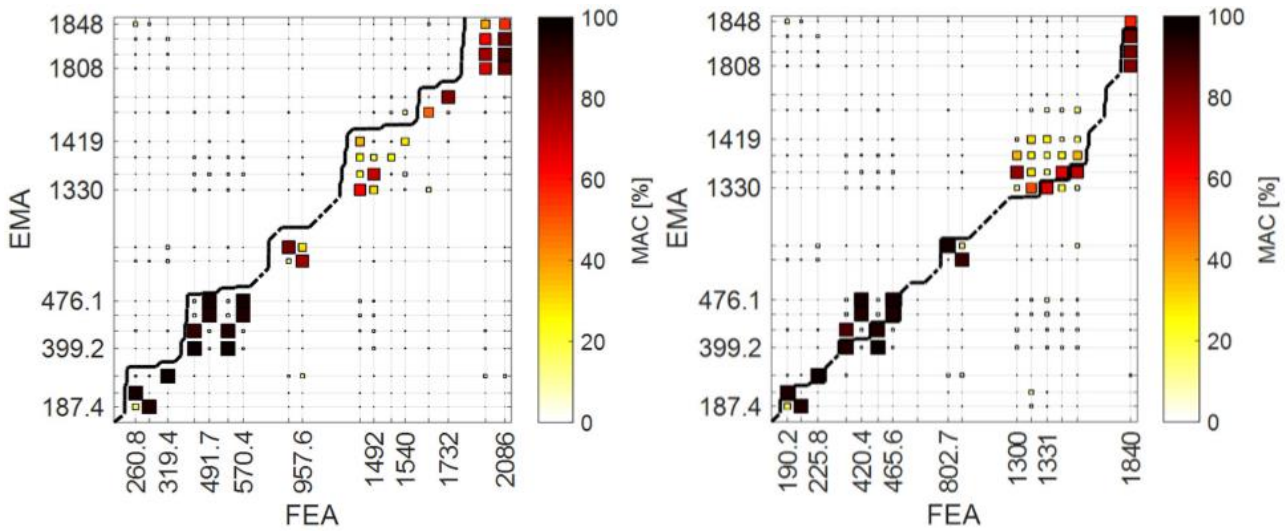


Figure 4.2 – Comparison between initial (left) and optimal (right) MAC values [19]

The result of the study has been proven to be valid, meaning that the new tuned FE model is a good approximator for experimental mode shapes. This is demonstrated using the dashed black line connecting the MAC points in both figures. In the right figure, the black line is almost converging into a diagonal straight line.

The optimisation of the case study has been imitated and solved through the optimisation panel. The problem does not have constraints besides the boundary conditions. The options for the `fmincon` function have been chosen to reduce computation time and might not resemble the choices that were made in the case study. However, the results should still be similar.

The results are displayed below, in addition to the graphs of the iterative procedure.

Table 4.3 – Results of optimisation.

E_{rim}	ρ_{rim}	$t_{mult,rim}$	E_{spokes}	ρ_{spokes}	D_{spokes}	N	ρ_{gear}	ρ_{hub}	Optimum	Iterations
[GPa]	[kg/m ³]	[-]	[GPa]	[kg/m ³]	[mm]	[N]	[kg/m ³]	[kg/m ³]	0.26751	165
50.95	2978	1.984	168.93	8645	1.77	773.83	6162	2312		

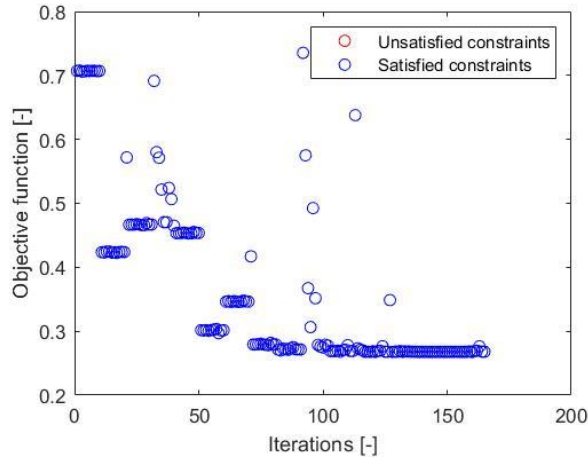


Figure 4.3 – Objective function

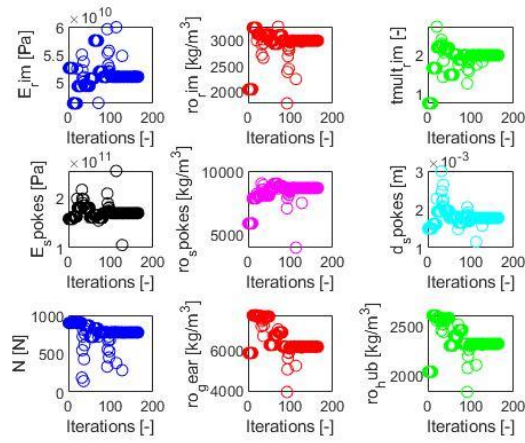


Figure 4.4 – Decision variables

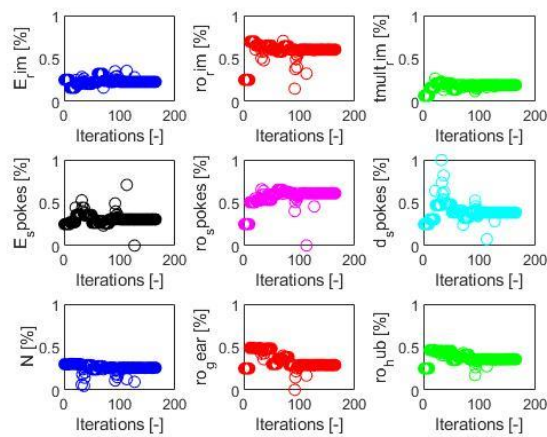


Figure 4.5 – Normalised decision variables

As `fmincon` is a deterministic procedure, the difference in results from the case study and the values provided by the optimisation panel are only due to the chosen option settings. Nevertheless, it can be said with great certainty that the results have been replicated.

In what follows, a reduction approach will be implemented to attempt to solve the problem from a different perspective. The reduction approach will be based on Buckingham's pi theorem.

Buckingham pi theorem:

The Buckingham's pi theorem correlates a set of variables or parameters to each other. It states that a set of n variables with m primary variables can be rewritten as a set of $(n-m)$ dimensionless groups which are referred to as π groups. Any variable in the set can be written as a function of the primary variables. A π group has the following general form:

$$\pi_i = f(x_i, \mathbf{x}_m) = x_i \prod_{j=1}^m x_{mj}^{a_j} \quad \text{for } i = 1 : (n-m) \quad (4.7)$$

$$\mathbf{x}_m = [x_{m1}, x_{m2} \dots x_{mm}]$$

Where \mathbf{x}_m is the vector of primary variables and a_j are rational exponents.

The exponents are computed such that the unit of the function is 1, meaning the result is dimensionless.

Any dimensional physical quantity is represented by a unit of measurement. In the mechanical field, these units are either mass (M), length (L), time (T), or f (M, L, T). In the electrical field, another set of units is used. Nevertheless, in a purely mechanical problem, the maximum number of primary variables that can exist is three. The unit of each of the primary variables chosen should contain at least one of the primary units (M, L or T). In addition, they should not be able to be nondimensionalised, as in, no solution satisfies equation (4.8).

$$\dim f(\mathbf{x}_m) = \dim \prod_{j=1}^m x_{mj}^{a_j} = 0 \quad (4.8)$$

The number of primary variables is the same as the number of primary units that exist in the problem. By applying Buckingham's pi theorem to an optimisation problem, the number of decision variables to compute can be reduced by m . Hence, the case study will be reintroduced as a nondimensionalized problem and the results from the original optimisation procedure will be compared to check if the conversion is viable or not.

Table 4.4 – Parameters and units.

Parameter
E_{rim} , Young Modulus [GPa]
ρ_{rim} , density [kg/m ³]
$t_{mult,rim}$, thickness multiplier [-]
E_{spokes} , Young Modulus [GPa]
ρ_{spokes} , density [kg/m ³]
ϕ_{spokes} , diameter [mm]
N , pretension [N]
ρ_{gear} , density [kg/m ³]
ρ_{hub} , density [kg/m ³]

As seen in table (4.4), all the primary units are used. The diameter, density of the spokes and the Young modulus of the spokes will be chosen to represent the length, mass, and time respectively. Hence, six π groups can be constructed.

Table 4.5 – π groups

Decision Variables	Lower Bound	Starting Point	Upper Bound
$\pi_1 = E_{rim} / E_{spokes}$	0.11111	0.33333	1
$\pi_2 = N / [E_{spokes} D_{spokes}^2]$	0	2.53968e-3	0.02857
$\pi_3 = \rho_{rim} / \rho_{spokes}$	0.11538	0.34615	1.03846
$\pi_4 = \rho_{hub} / \rho_{spokes}$	0.11538	0.34615	1.03846
$\pi_5 = \rho_{gear} / \rho_{spokes}$	0.33333	1	3
$\pi_6 = t_{mult,rim}$	0.1	0.75	10

Due to this conversion, the Optimisation Panel will only compute six variables instead of nine, which will greatly improve the computational time. On the other hand, the objective function needs the values of the original set to compute the relative error. Hence, the function will be slightly modified to calculate the original parameters from the values of the π groups provided by `fmincon`.

The problem becomes an underdetermined system since the number of unknowns exceeds the number of equations. In an underdetermined system, the solution is not unique. The solution consists of a basis of the null space and a particular solution.

$$\mathbf{Ax} = \mathbf{A}(\mathbf{u} + \mathbf{v}) = \mathbf{Au} + \mathbf{Av} = \mathbf{0} + \mathbf{b} = \mathbf{b} \quad (4.9)$$

Where \mathbf{x} , \mathbf{u} and \mathbf{v} are the general, null space and particular solutions respectively. \mathbf{A} is the matrix of equations and \mathbf{b} is the vector of equalities of the equations.

By applying the \ln operator, the system can be translated into the form of equation (4.9).

$$\left(\begin{array}{l} \ln \pi_1 = \ln E_{rim} - \ln E_{spokes} \\ \ln \pi_2 = \ln N - \ln E_{spokes} - 2 \ln D_{spokes} \\ \ln \pi_3 = \ln \rho_{rim} - \ln \rho_{spokes} \\ \ln \pi_4 = \ln \rho_{hub} - \ln \rho_{spokes} \\ \ln \pi_5 = \ln \rho_{gear} - \ln \rho_{spokes} \\ \ln \pi_6 = \ln t_{mult,rim} \end{array} \right) \Rightarrow \left(\begin{array}{cccccccc} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \begin{bmatrix} \ln E_{rim} \\ \ln \rho_{rim} \\ \ln t_{mult,rim} \\ \ln E_{spokes} \\ \ln \rho_{spokes} \\ \ln D_{spokes} \\ \ln N \\ \ln \rho_{gear} \\ \ln \rho_{hub} \end{bmatrix} = \begin{bmatrix} \ln \pi_1 \\ \ln \pi_2 \\ \ln \pi_3 \\ \ln \pi_4 \\ \ln \pi_5 \\ \ln \pi_6 \end{bmatrix} \quad (4.10)$$

Hence the \mathbf{A} matrix and \mathbf{b} vector are obtained.

The null space of \mathbf{A} is computed to find the \mathbf{u} vector (basis solution).

$$\mathbf{A}\mathbf{u} = \mathbf{0} \Rightarrow \mathbf{u} = \begin{bmatrix} \ln E_{spokes} \\ \ln \rho_{spokes} \\ 0 \\ \ln E_{spokes} \\ \ln \rho_{spokes} \\ \ln D_{spokes} \\ \ln E_{spokes} + 2 \ln D_{spokes} \\ \ln \rho_{spokes} \\ \ln \rho_{spokes} \end{bmatrix} = \begin{bmatrix} \ln a \\ \ln b \\ 0 \\ \ln a \\ \ln b \\ \ln c \\ \ln a + 2 \ln c \\ \ln b \\ \ln b \end{bmatrix} \quad (4.11)$$

Where a , b , and c are free values.

Then the particular solution is computed by substituting with the lower bounds of the major variables.

$$\mathbf{A}\mathbf{v} = \mathbf{b} \Rightarrow \mathbf{v} = \begin{bmatrix} \ln \pi_1 + \ln E_{spokes} \\ \ln \pi_3 + \ln \rho_{spokes} \\ \ln \pi_6 \\ \ln E_{spokes} \\ \ln \rho_{spokes} \\ \ln D_{spokes} \\ \ln \pi_2 + \ln E_{spokes} + 2 \ln D_{spokes} \\ \ln \pi_5 + \ln \rho_{spokes} \\ \ln \pi_4 + \ln \rho_{spokes} \end{bmatrix} = \begin{bmatrix} \ln \pi_1 + \ln 1.05e11 \\ \ln \pi_3 + \ln 3900 \\ \ln \pi_6 \\ \ln 1.05e11 \\ \ln 3900 \\ \ln e - 3 \\ \ln \pi_2 + \ln 1.05e11 + 2 \ln e - 3 \\ \ln \pi_5 + \ln 3900 \\ \ln \pi_4 + \ln 3900 \end{bmatrix} \quad (4.12)$$

Equation (4.13) represents the general solution of the original set of parameters, while equation (4.14) represents the constraints which the free variables are subjected to, from which the boundary conditions are deduced.

$$\mathbf{x} = e^{\wedge} \begin{bmatrix} \ln \pi_1 + \ln 1.05e11 + \ln a \\ \ln \pi_3 + \ln 3900 + \ln b \\ \ln \pi_6 \\ \ln 1.05e11 + \ln a \\ \ln 3900 + \ln b \\ \ln e - 3 + \ln c \\ \ln \pi_2 + \ln 1.05e11 + 2 \ln e - 3 + \ln a + 2 \ln c \\ \ln \pi_5 + \ln 3900 + \ln b \\ \ln \pi_4 + \ln 3900 + \ln b \end{bmatrix} = \begin{bmatrix} 1.05e^{11} a \pi_1 \\ 3900 b \pi_3 \\ \pi_6 \\ 1.05e^{11} a \\ 3900 b \\ 0.001 c \\ 105000 a c^2 \pi_2 \\ 3900 b \pi_5 \\ 3900 b \pi_4 \end{bmatrix} \quad (4.13)$$

$$\begin{aligned}
35 &\leq a\pi_1 105 \leq 105 \\
1350 &\leq \pi_3 3900b \leq 4050 \\
- \\
1 &\leq a \leq 3 \\
1 &\leq b \leq 3 \\
1 &\leq c \leq 3 \\
0 &\leq ac^2\pi_2 105 \leq 3 \\
1 &\leq \pi_5 b \leq 3 \\
1350 &\leq \pi_4 3900b \leq 4050
\end{aligned} \tag{4.14}$$

The only constraint that needs to be introduced into the problems is $ac^2\pi_2 105 \leq 3$. The density constraints are irrelevant since any parameter value that does not satisfy them will correspond to a high value of the objective function and will be immediately dismissed. By substituting the π groups with their max and min values accordingly, all the other terms of equation (4.14) result in the boundary conditions of the free variables, as shown in table 4.6, except for the term $ac^2\pi_2 105 \leq 3$.

The initial values of the free variables are calculated in equation (4.15) by reverse calculating equation (4.13) and setting the value of \mathbf{x} to that of the starting values of the parameters given in table 4.2 and the values of the π groups to their starting points in table 4.5.

$$\begin{bmatrix} 1.05e^{11}a_i 0.33333 \\ 3900b_i 0.34615 \\ 0.75 \\ 1.05e^{11}a_i \\ 3900b_i \\ 0.001c_i \\ 105000a_i c_i^2 2.53968e^{-3} \\ 3900b_i 1 \\ 3900b_i 0.34615 \end{bmatrix} = \begin{bmatrix} 52.5e^9 \\ 2025 \\ 0.75 \\ 157.5e^9 \\ 5850 \\ 1.5 \\ 900 \\ 5850 \\ 2025 \end{bmatrix} \Leftrightarrow \begin{cases} a_i = 1.5 \\ b_i = 1.5 \\ c_i = 1.5 \end{cases} \tag{4.15}$$

Where a_i , b_i and c_i are the initial values of the free variables a , b , and c respectively.

Table 4.6 introduces all possible combinations of the free variables that can be constructed to reduce the number of decision variables.

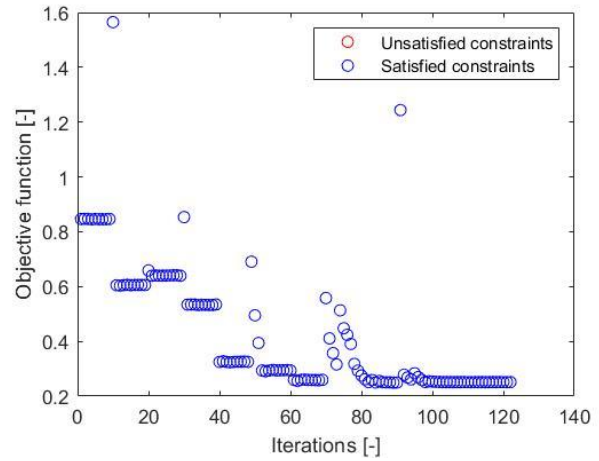
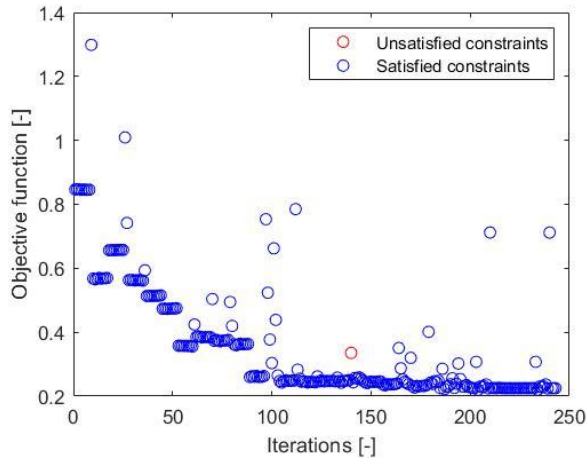
Table 4.6 – Possible combinations of reduction variables.

Combinations		Variables		
		Size	Boundaries	Initial Values
1	$a = b = c$	7	$1 \leq \pi_7 \leq 3$	$\pi_7 = 1.5$
2	$a = b$	8	$1 \leq \pi_7, \pi_8 \leq 3$	$\pi_7 = 1.5, \pi_8 = 1.5$
3	$a = c$	8	$1 \leq \pi_7, \pi_8 \leq 3$	$\pi_7 = 1.5, \pi_8 = 1.5$
4	$b = c$	8	$1 \leq \pi_7, \pi_8 \leq 3$	$\pi_7 = 1.5, \pi_8 = 1.5$

Table 4.7 – Reduction optimisation results

1 st Combination				2 nd Combination				3 rd Combination				4 th Combination			
Variables		Parameters		Variables		Parameters		Variables		Parameters		Variables		Parameters	
π_1	0.4142	E_{rim}	75.82	π_1	0.4039	E_{rim}	76.7	π_1	0.3581	E_{rim}	73.24	π_1	0.3687	E_{rim}	79.78
π_2	0.0042	ρ_{rim}	3380	π_2	0.0032	ρ_{rim}	3430	π_2	0.0018	ρ_{rim}	3189	π_2	0.0031	ρ_{rim}	3607
π_3	0.71	$t_{mult,rim}$	2.715	π_3	0.6946	$t_{mult,rim}$	2.288	π_3	0.5088	$t_{mult,rim}$	1.59	π_3	0.725	$t_{mult,rim}$	2.251
π_4	0.4725	E_{spokes}	183	π_4	0.4541	E_{spokes}	189.9	π_4	0.4212	E_{spokes}	204.5	π_4	0.4707	E_{spokes}	216.4
π_5	1.1944	ρ_{spokes}	4760	π_5	1.2049	ρ_{spokes}	4938	π_5	1.1232	ρ_{spokes}	6267	π_5	1.1379	ρ_{spokes}	4975
π_6	2.7147	D_{spokes}	1.2	π_6	2.2882	D_{spokes}	1.3	π_6	1.5904	D_{spokes}	1.4	π_6	2.2507	D_{spokes}	1.3
α	1.2204	N	795.4	α	1.2662	N	721.8	α	1.3636	N	488.6	α	1.4425	N	770
-	-	ρ_{gear}	5685	b	1.299	ρ_{gear}	5950	b	1.6069	ρ_{gear}	7039	b	1.2756	ρ_{gear}	5661
-	-	ρ_{hub}	2249	-	-	ρ_{hub}	2242	-	-	ρ_{hub}	2640	-	-	ρ_{hub}	2342
Iterations = 243				Iterations = 122				Iterations = 256				Iterations = 188			
Optimum = 0.22007				Optimum = 0.24914				Optimum = 0.26895				Optimum = 0.24646			

The below figures will represent the convergence of each combination to its respective optimum.



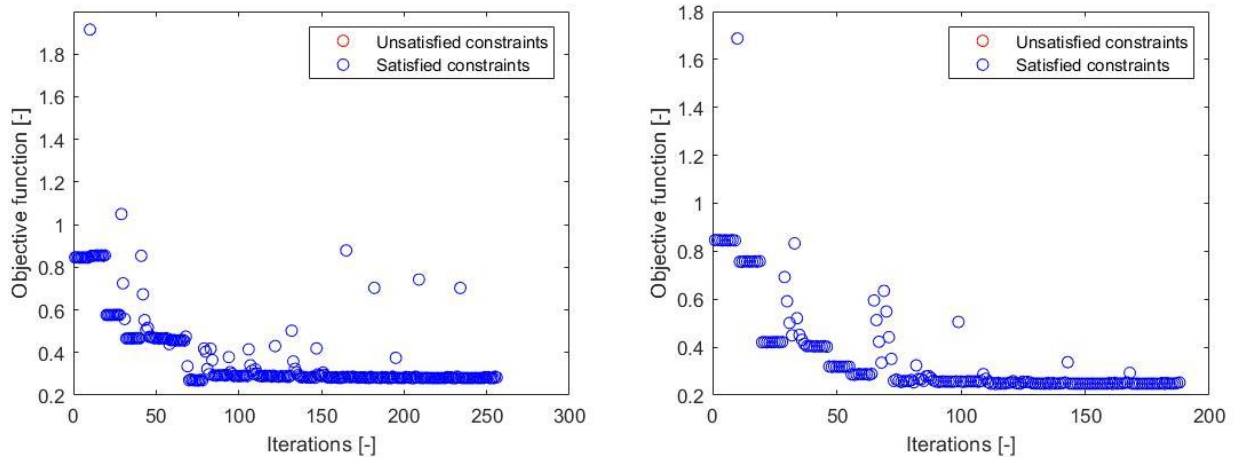


Figure 4.6 – Objective function convergence of the first (top left), second (top right), third (bottom left) and fourth (bottom right) combinations.

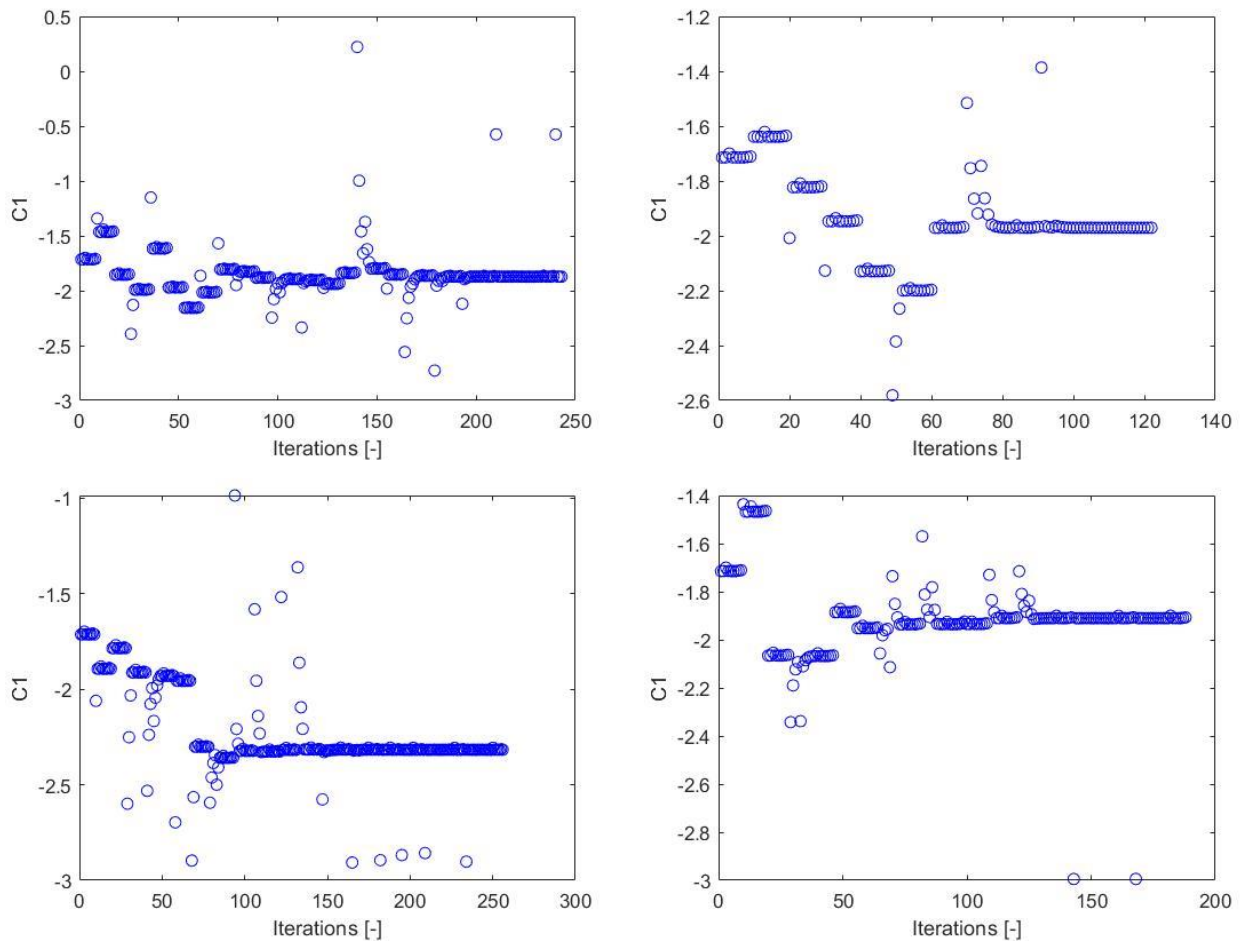


Figure 4.7 – Constraint convergence of the first (top left), second (top right), third (bottom left) and fourth (bottom right) combinations.

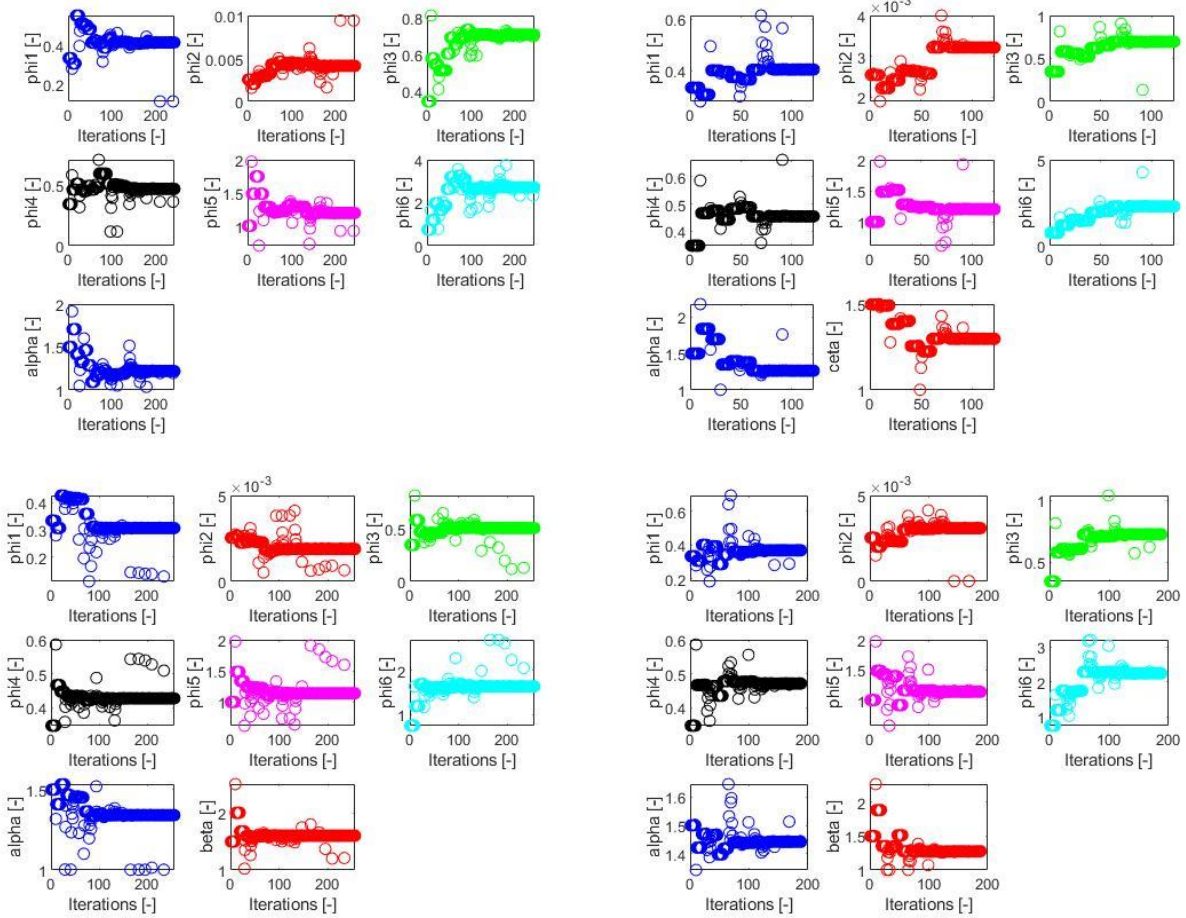


Figure 4.8 – Decision variables convergence of the first (top left), second (top right), third (bottom left) and fourth (bottom right) combinations.

In this case study, the most time is spent on calculating the value of the objective function and not on calculating the values of the decision values for each iteration. Hence, reducing the computation time is strictly proportional to decreasing the number of iterations.

As noticed from the results, reducing the number of variables does not necessarily reduce the computation time. Another similar observation yields that the number of iterations vary according to the correlation among the problem parameters. In the original problem, the decision variables (parameters) were independent. However, the parameters calculated through the reduction variables are correlated to one or the other through at least one decision variable. `fmincon` is not aware of the inner workings of the objective function. The direction and size of the augmentation of each decision variable depend on their effect on the progression of the objective function. The typical procedure that `fmincon` follows relies on testing the individual effect of each decision variable on the value of the objective function. When testing a decision variable inside the `fmincon` algorithm, more than one parameter is being modified during the calculations of the objective function. This change can yield unpredictable results. The objective function may appear falsely sensitive to certain variables, which could trap the iterations around a fixed point or in a loop. This is noticed in the third combination, where the objective function maintains an almost fixed value for more than 100 iterations, and the minimum has already been found at the 75th iteration out of 256 iterations. The influence of the hidden correlation among the parameters can push away the solution from the optimum as in the third combination or draw it closer as in the second and fourth combinations. The highest sensitivity to the objective function is expected from the first

combination, which justifies the high number of iterations. Yet, it managed to avoid getting trapped in a loop and achieved the lowest optimum among the original problem and the other combinations. The proof for the influence of this correlation is even more evident when the parameters in the original problem were swapped with the π groups, α , b and c . This way the problem remains with 9 decision variables, but the solution has not been repeated; even though mathematically it could have. Below are the results of the substitution:

Table 4.8 – Results of the substitution.

π_1	π_2	π_3	π_4	π_5	π_6	α	b	c	Optimum	Iterations
0.45929	0.002926	0.74704	0.30594	0.78244	1.5747	1.4495	1.5784	1.3386	0.21628	279
E_{rim} [GPa]	ρ_{rim} [kg/m ³]	$t_{mult,rim}$ [-]	E_{spokes} [GPa]	ρ_{spokes} [kg/m ³]	D_{spokes} [mm]	N [N]	ρ_{gear} [kg/m ³]	ρ_{hub} [kg/m ³]		
99.86	4599	1.5747	217.43	6156	1.3	798	4816	1883		

By reverse calculating the free variables and the π groups from the solution of table 4.3, the below table represents the expected solution for the substitution problem.

Table 4.9 – Results of the reverse calculations

E_{rim} [GPa]	ρ_{rim} [kg/m ³]	$t_{mult,rim}$ [-]	E_{spokes} [GPa]	ρ_{spokes} [kg/m ³]	D_{spokes} [mm]	N [N]	ρ_{gear} [kg/m ³]	ρ_{hub} [kg/m ³]
50.95	2978	1.984	168.93	8645	1.77	773.83	6162	2313
π_1	π_2	π_3	π_4	π_5	π_6	α	b	c
0.3	0.001462	0.3445	0.2675	0.7128	1.984	1.61	2.2165	1.77

Below are the graphical solutions to the substitution problem.

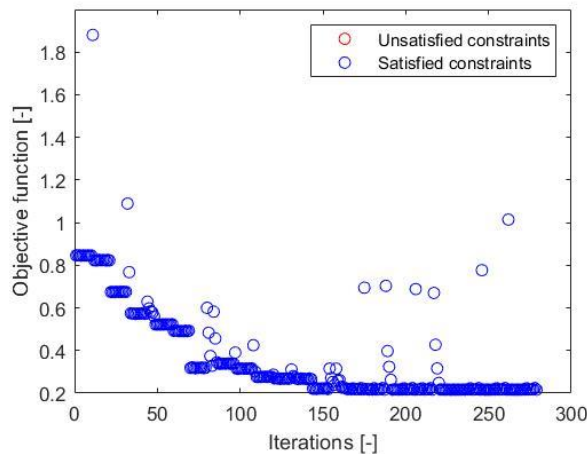


Figure 4.9 – Objective function

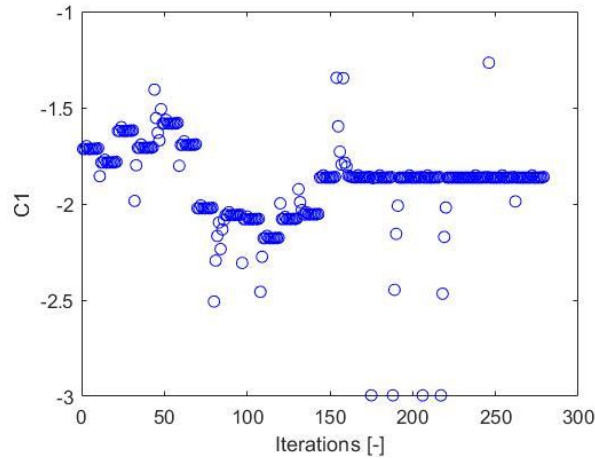


Figure 4.10 – Constraint

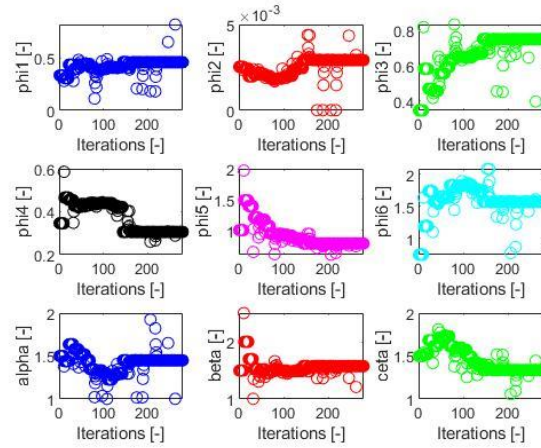


Figure 4.11 – Decision variables

In what follows, a graphical representation, evaluation tracking and comparison of the original problem and the second combination will be demonstrated. The set up has been performed manually and the following results have been obtained.

For the first representation, the initial values were taken without any modifications. It is worthy to point out that the starting conditions for both optimisations are the same. Hence, figures 4.12 and 4.13 correspond to both optimisations. For simplicity, the optimisation with original setting will be referred to as the classical optimisation while the second one will be referred to as the reduction optimisation. Another note worth mentioning is that the model has 200 modes and 2146 nodes. Nevertheless, for representative purposes, the MACs will be calculated using only 19 modes and 65 nodes, to generate meaningful and comparable figures. However, due to using only 65 nodes out of the 2146 available nodes, the values of the manually calculated relative errors will be augmented but the evolution of the procedure should still be evident.

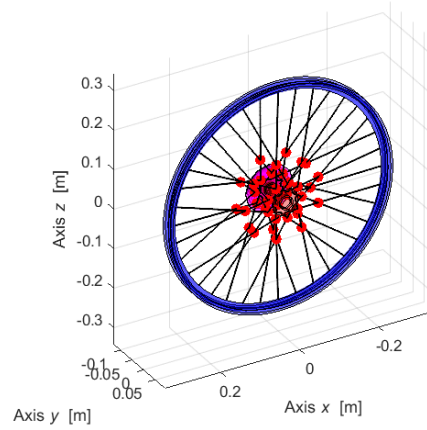


Figure 4.12 – Model of starting parameters

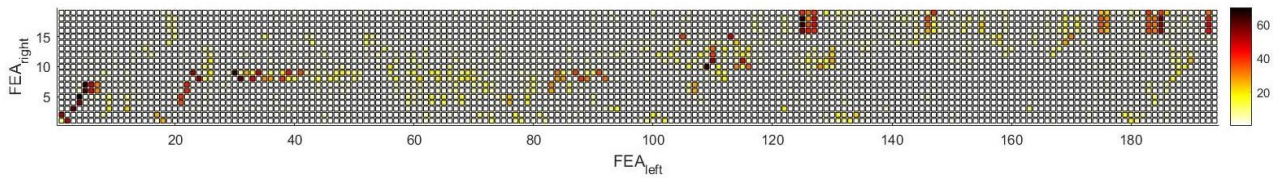


Figure 4.13 – MAC between EMA (FEA_{right}) and FEA (FEA_{left}) values for the 1st iteration

The value of the relative error has been computed according to equation (4.6) to be 0.762 and the relative error of MACW2 alone to be 0.4568. Both these values are high and undesirable but expected for the unoptimized problem.

The following evaluations will be performed at the 25%, 50% and 75% of the iterations computed and at the optimum iteration for each optimisation setting.

Table 4.10 – Values of the parameters at 25%, 50%, 75% and optimum iteration.

	Optimisation Setting	Iteration	E_{rim} [GPa]	ρ_{rim} [kg/m ³]	$t_{mult,rim}$ [-]	E_{spokes} [GPa]	ρ_{spokes} [kg/m ³]	D_{spokes} [mm]	N [N]	ρ_{gear} [kg/m ³]	ρ_{hub} [kg/m ³]
25%	Classical	41	49.27	3103	2.171	180	8055	1.957	865.8	7670	2575
	Reduction	30	58.64	1915	1.414	150.5	3913	1.419	430.6	4109	1599
50%	Classical	82	50.89	2981	1.986	169.2	8666	1.778	779.7	6163	2312
	Reduction	60	75.63	3505	2.016	207.2	5387	1.227	458.3	6688	2627
75%	Classical	123	50.93	2978	1.985	169.3	8651	1.773	776.3	6162	2312
	Reduction	91	149.3	920.3	4.271	264.4	6876	1.363	828.5	13243	4554
Optimum	Classical	128	50.95	2978	1.984	168.9	8645	1.77	773.8	6162	2312
	Reduction	89	76.7	3430	2.288	189.9	4938	1.299	555.7	5950	2242

Table 4.11 – Results of the 20%, 50%, 75% and optimum iteration.

	Optimisation	Classical	Reduction
25%	RE(MACW2)	0.4643	0.5548
	Overall relative error	0.6172	0.9554

50%	RE(MACW2)	0.467	0.4406
	Overall relative error	0.479	0.4808
75%	RE(MACW2)	0.4671	0.6818
	Overall relative error	0.4776	1.2457
Optimum	RE(MACW2)	0.4675	0.4201
	Overall relative error	0.4774	0.4301

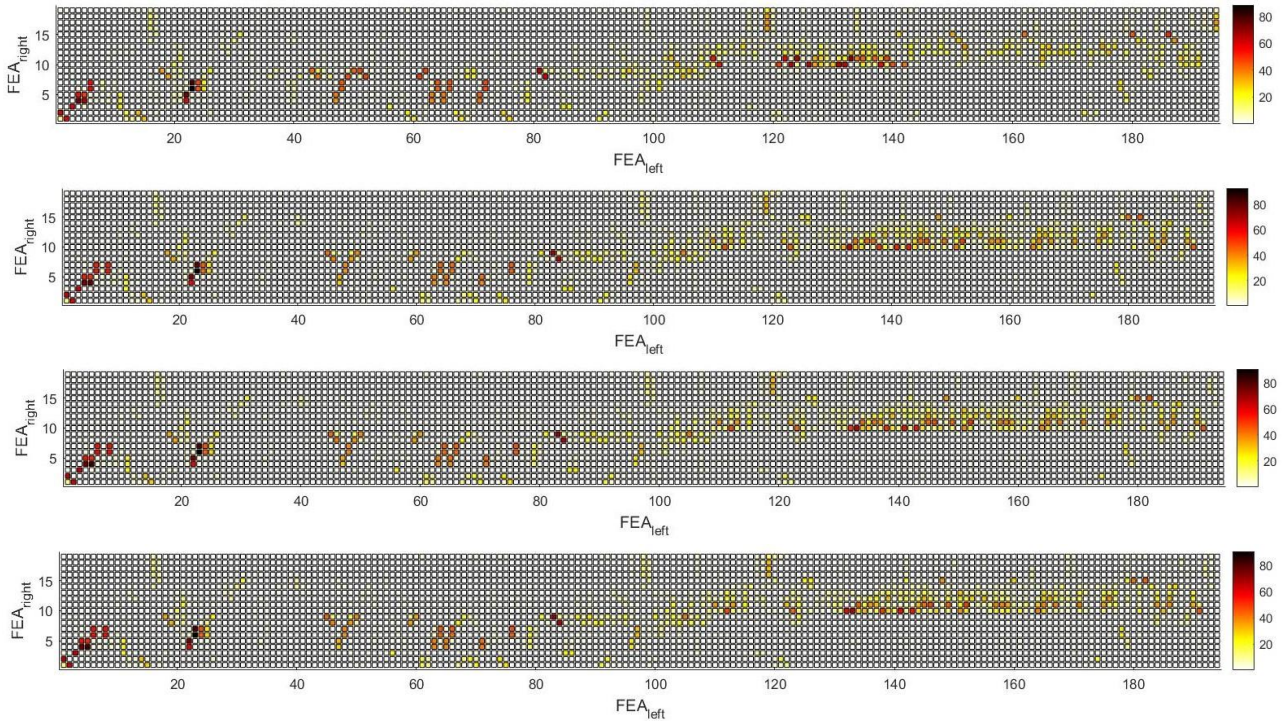
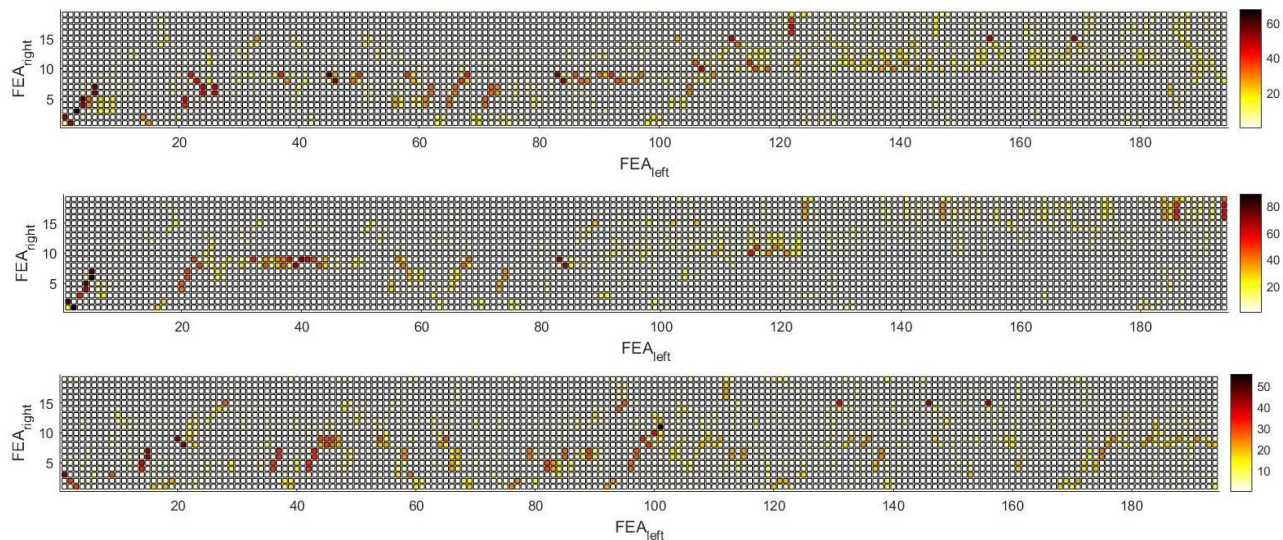


Figure 4.14 – MAC between EMA (FEA_{right}) and FEA (FEA_{left}) values for the 25% (top), 50% (top middle), 75% (bottom middle) and the optimum iteration (bottom) of the classical optimisation

After the 50th iteration, the rate of change of the relative error decreased; indicating that the algorithm has reached the region of the optimum without successfully capturing the optimum.



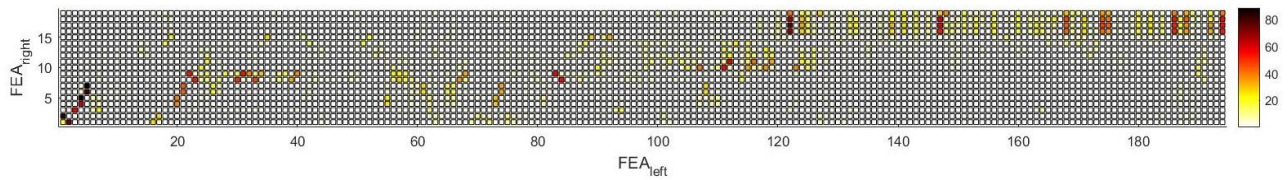


Figure 4.15 – MAC between EMA (FEA_{right}) and FEA (FEA_{left}) values for the 25% (top), 50% (top middle), 75% (bottom middle) and the optimum iteration (bottom) of the reduction optimisation

The trajectory of the optimisation is more evident in the figures of the reduction optimisation than in those of the classical optimisation. The classical optimisation was faster in the first quarter of the optimisation, but afterwards it slowed down in the neighbourhood of the optimum. On the other hand, the reduction optimisation exhibited a steady pace.

The improvement for both optimisations is visible as higher values of MAC shift from one mode to another with each representation.

Remarks:

1. It is known that the final iteration does not represent the optimum iteration in many cases. It simply means that the convergence criteria for `fmincon` have not been met at the optimum iteration, altering the path of the optimisation from the optimum as the algorithm continues looping. This is observed in the reduction optimisation, as the optimum iteration occurs before the 91st (75%) iteration.
2. `Fmincon` studies the step value of each variable before deciding the values for the following iteration. The 91st (75%) iteration was an undesirable test value that backtracked the optimisation as shown from the relative error values and the MAC figure 4.15 (bottom middle).

5. Uniqueness of optimisation

Is optimization unique? Theoretically, in some exceptional cases, it might be so. However, the human capability is limited, and the computation algorithms are blind and straightforward. They cannot deviate from the path laid ahead of them, the path they were programmed to follow.

Different algorithms yield different yet close solutions to the same optimisation problem. If so, then on what basis is a given solution an optimum? Perfection is an illusion and from a mathematical point of view, knowing that 0.99 is equal to 1, an optimum solution can have many contradicting values. However, from a physical point of view, it is not so. In the case study of the thesis, a reduction approach was implemented to change the form of the variables without altering the objective function or the optimization algorithm, to discover other feasible solutions. Expanding from this point, what other techniques exist to readdress the direction of the optimisation sequence? The following problem has been used as an example to implement the different strategies:

$$\begin{aligned}
 \min f(\mathbf{x}) &= (1-x_1)^2 + (x_2 - x_1^2)^2 \\
 x_1^2 + x_2^2 - 1 &\leq 0 \\
 \mathbf{x}_i &= [2 \ -4] \\
 -10 &\leq x_1 \leq 10 \\
 -20 &\leq x_2 \leq 20
 \end{aligned} \tag{5.1}$$

Where f is the objective function, \mathbf{x} is the vector of decision variables and \mathbf{x}_i is the vector of initial values.

5.1 Initial point

After solving problem (5.1) using `fmincon`, the following table has been constructed:

Table 5.1.1 – Optimisation results of problem (5.1)

	Initial Values	20 th Iteration	23 rd Iteration (optimum)	# Iterations
x_1	2	0.845745285	0.814125626	59
x_2	-4	0.581483001	0.726686687	
$f(x_1, x_2)$	65	0.041697515	0.038630723	

The problem has been solved again but with taking the values of the 20th iteration as the initial values:

Table 5.1.2 – Optimisation results of problem (5.1) with new initial values

	Initial Values (manually inputted)	Initial Values (1 st Iteration)	62 nd Iteration (optimum)	# Iterations
x_1	0.845745285	0.84575	0.808169382	62
x_2	0.581483001	0.58148	0.588950089	
$f(x_1, x_2)$	-	0.041698998	0.040919042	

The chosen initial values were rounded up during the 1st iteration and hence a new optimum was calculated based on this new path. Usually, if the initial values were taken as is, the new iterations would have ended at 40 (59 – 19) and the same results from iteration 20 to iteration 59 (Table 5.1.1) would have been repeated for the modified problem. Even if choosing new initial values changes the route of the optimisation, choosing a value nearer to the optimum does not necessarily decrease the number of iterations (the optimum is not reached faster) as observed from table 5.1.2.

5.2 Redundant constraints

Adding or omitting constraints will drastically change the route of optimisation even if the unconstrained optimum already exists within the pre and/or post-defined constraints.

The simplest useless constraint is a zero constraint, since $0 \leq 0$. Logically, the addition of such a requirement should not change anything throughout the optimisation. However, fmincon is an algorithm that is not aware of the form of the objective function or the constraints. It calculates the next iteration based on the returned values of these equations and their dimensions. Hence, it is sufficient to just change the dimension of the problem to alter its path.

Table 5.2.1 – Optimisation results of problem (5.1) with zero constraint

	Initial Values	142 nd Iteration (optimum)	# Iterations
x_1	2	0.808168802	142
x_2	-4	0.588949002	
$f(x_1, x_2)$	65	0.040919284	

It is worth mentioning that increasing the dimension of the problem does not have to incite an increase in computation time. This is a feature strictly consistent with the zero constraint.

5.3 Introducing new variables

This section was inspired by the successful implementation of the Buckingham phi theorem in the section of the case study. However, the Buckingham theorem is usually applied within the physical domain. In what follows, the reduction method will be implemented in the mathematical domain. Equation (5.3.1) represents a general format to approach the transformation:

$$x_i = \sum_{j=1}^n \alpha_{ij} x_j + y_i \quad (5.3.1)$$

Where a_{ij} is a constant coefficient, n is the size of the \mathbf{x} vector and \mathbf{y} is a vector of variable and zero components. The non-zero $a_{ij}x_j$ and y_i components are the new variables of the system.

Applying the format to problem (5.1), the following is obtained:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + y_1 \\ a_{21}x_1 + a_{22}x_2 + y_2 \end{bmatrix} \quad (5.3.2)$$

Note: multiplying by or/and adding a constant to the decision variables only will not alter the optimisation results in any way, such as the next example:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + 1 \\ 3x_2 \end{bmatrix} \quad (5.3.3)$$

The optimisation of problem (5.3.4), a modified version of problem (5.1), has been tabulated in table 5.3.1.

$$\begin{aligned} \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} x_1^* \\ x_1 + x_2^* \end{bmatrix} \Rightarrow \mathbf{x}^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 - x_1 \end{bmatrix} \\ \min f(\mathbf{x}) &= (1 - x_1^*)^2 + (x_2^* + x_1^* - x_1^{*2})^2 \\ x_1^{*2} + (x_1^* + x_2^*)^2 - 1 &\leq 0 \\ \mathbf{x}_i^* &= [2 \ -6] \\ -10 &\leq x_1^* \leq 10 \\ -10 &\leq x_2^* \leq 10 \end{aligned} \quad (5.3.4)$$

Where \mathbf{x}^* is the vector of the altered decision variables.

Table 5.3.1 – Optimisation results of problem (5.3.4)

	Initial Values	54 th Iteration (optimum)	# Iterations
x_1^*	2	0.808168655	56
x_2^*	-6	-0.219220648	
x_1	2	0.808168655	
x_2	-4	0.588948007	
$f(x_1, x_2)$	65	0.040919437	

Each optimum calculated from each method is different but almost the same. Ideally, as the complexity of the problem increases, wider variations in the solutions should become more visible (refer to the case study).

Final remarks

Fmincon and other similar algorithms will only show one possible solution even in the presence of others. For functions with multiple optima, the algorithm, unaware, will lead to the one in its path, ignoring the possibility of there being another solution or a better solution. If unsatisfied with the results of the optimisation, altering the path can reveal new possibilities.

An optimum is any solution that is significantly better than the starting point, even if not optimal. The optimisation problem does not need to include an abundance of constraints if they are not likely to change the course of the optimisation. A careful observation is advised to distinguish between relevant and omittable constraints.

Conclusions

In the study of the `fmincon` function used in the creation of the Optimisation Panel, it became evident in many cases how disappointing this tool really is. In the presence of multiple feasible solutions, `fmincon` will only point towards one fooling the user to believe that it is the only optimum solution. Even if the solution satisfies all the constraints and boundaries, a better one might exist that satisfies other unwritten constraints or ambitions. Perhaps, the most common way to turn the direction of optimisation is through the choice of the initial point. However, that might require the user to have a vague idea on the location of the optimum. In the case study of the thesis, a reduction approach was implemented to change the form of the variables without altering the objective function or the optimisation algorithm, in an attempt to discover other feasible solutions, which inspired the continued research into the nature of optimisation algorithms.

Reference

- [1] Alizadeh F., Goldfarb D., “Second-order cone programming”, *Mathematical Programming*, 95, 2003, pp. 3-51.
- [2] Diwekar U.M., “Introduction to applied optimization”. *Springer, Cham*, 2020.
- [3] Boyles S.D., “Basic optimization strategies”, Spring 2015.
Stephen D. Boyles basic optimization strategies.pdf
- [4] <https://www.ibm.com/docs/en/icos/12.9.0?topic=variables-all-different-constraint> available in 2022-02-16.
- [5] <https://www.coursera.org/articles/project-management-triangle> available in 2022-02-16.
Coursera what is the project management triangle.pdf
- [6] Ouni, Ali, Marouane Kessentini, and Houari Sahraoui. "Multiobjective optimization for software refactoring and evolution." In *Advances in computers*, vol. 94, pp. 103-167. Elsevier, 2014.
Science direct topics fitness function.pdf
- [7] Kumar, B. Vinoth, G. R. Karpagam, and Yanjun Zhao. "Evolutionary algorithm with memetic search capability for optic disc localization in retinal fundus images." In *Intelligent Data Analysis for Biomedical Applications*, pp. 191-207. Academic Press, 2019.
Science direct topics constraint function.pdf
- [8] Levy D., “George B. Dantzig, operations research professor, dies at 90”, *Stanford*, 2005.
Stanford report George B. Dantzig, operations research professor, dies at 90.pdf
- [9] <https://uk.mathworks.com/discovery/matlab-gui.html> available in 2022-03-20.
MathWorks MATLAB GUI.pdf
- [10] <https://uk.mathworks.com/help/matlab/ref/matlab.ui.figureappd-properties.html> available in 2022-03-20.
MathWorks control UI figure appearance and behavior for uifigure-based apps.pdf
- [11] https://uk.mathworks.com/help/matlab/ref/uicontrol.html?s_tid=srchtitle_uicontrol_1 available in 2022-03-20.
MathWorks MATLAB uicontrol.pdf
- [12] <https://uk.mathworks.com/help/matlab/ref/uitable.html> available in 2022-03-20.
MathWorks MATLAB uitable.pdf
- [13] Boelkins M., Austin D., Schlicker S., “Taylor polynomial and Taylor series”, *LibreTexts*, 2020.
Libretexts Taylor polynomials and Taylor series.pdf
- [14] https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods available in 2022-05-27.
Trust region methods.pdf
- [15] “The Simplification Principle”, *Deloitte*, 2021
Deloitte - the simplification principle.pdf
- [16] <https://ludovicarnold.com/teaching/optimization/problem-statement/> available in 2022-06-22.
Ludovic Arnold - problem statement.pdf
- [17] Jin, Chao. “A Sequential Process Monitoring Approach using Hidden Markov Model for Unobservable Process”, *Drift*, 2015
- [18] Bäck, T., David F., Zbigniew M., “Evolutionary computation 1: basic algorithms and operators”, *CRC press*, 2018.
- [19] E. Bonisoli, A.D. Vella, S. Venturini, *Uncertainty Effects on Bike Spoke Wheel Modal Behaviour*, Politecnico di Torino, Torino, Italy.
- [20] <https://www.solver.com/problem-types> available in 2022-01-25.
Solver Optimization problem types - overview.pdf
- [21] Nocedal J., and Wright S.J., “Numerical optimization”, *Springer New York*, 1999.

- [22] Potra, Florian A., and Stephen J. Wright. "Interior-point methods." *Journal of computational and applied mathematics* 124, no. 1-2 (2000): 281-302.
- [23] Senning, Jonathan R. "Computing and estimating the rate of convergence.", 2007.
- [24] Newton, Isaac, Joseph Fourier, Gottfried Leibniz, Leonhard Euler, Émile Picard, Józef Maria Hoene-Wroński, Ernst Lindelöf et al. "Rate of convergence."
- [25] Conn, Andrew R., Nicholas IM Gould, and Philippe L. Toint. "Trust region methods". *Society for Industrial and Applied Mathematics*, 2000.
- [26] Byrd, Richard H., Robert B. Schnabel, and Gerald A. Shultz. "A trust region algorithm for nonlinearly constrained optimization." *SIAM Journal on Numerical Analysis* 24, no. 5 (1987): 1152-1170.