BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

OPTIMIZATION IN ENGINEERING PROBLEMS

DIPLOMA THESIS

AUTHOR DIPESH LAMICHHANE

SUPERVISOR Ing. Mgr. EVA MRÁZKOVÁ, Ph.D.

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Mathematics
Student:	**Dipesh Lamichhane**
Degree programm:	Applied and Interdisciplinary Mathematics
Branch:	no specialisation
Supervisor:	**Ing. Mgr. Eva Mrázková, Ph.D.**
Academic year:	2024/25

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Optimization in engineering problems

Brief Description:

Optimization methods are often needed in solving engineering problems. These methods will be used for appropriate selected problems and corresponding mathematical models will be formulated. These models will be implemented in optimization software (GAMS, Python...) and an adequate discussion of the obtained results will be carried out.

Master's Thesis goals:

– Studying the basic properties of mathematical programming problems (linear, nonlinear, integer, multiobjective, resp. stochastic as needed).
– Formulation of mathematical models for selected engineering problems.
– Implementation in optimization software (GAMS, Python,...) and an adequate discussion of the obtained results.

Recommended bibliography:

BAZARAA, M. S., SHERALI, H. D., SHETTY, C. M. Nonlinear Programming: Theory and Algorithms. New York: Wiley and Sons, 1993.

RAO, S. S. Engineering Optimization: Theory and Applications. New Jersey: Wiley and Sons, 2009.

RARDIN, R. L. Optimization in Operations Research. New Jersey: Prentice Hall, 1998.

TAHA, A. H. Operations research: An introduction. New Jersey: Prentice Hall, 1997.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2024/25

In Brno,

L. S.

doc. Mgr. Petr Vašík, Ph.D. Director of the Institute	doc. Ing. Jiří Hlinka, Ph.D. FME dean

Abstract

This thesis addresses the application of optimization methods to engineering design problems, focusing on structural systems. It reviews linear programming (LP), nonlinear programming (NLP), and multi-objective optimization techniques. Four case studies are solved: (1) a scaffolding support system (LP); (2) a beam design problem (NLP); (3) a truss design problem (NLP); and (4) a water-tank column design with competing objectives (minimize mass, maximize natural frequency). For each problem, equilibrium equations and constraints are derived from mechanics, and the optimization model is implemented in MATLAB and Python. The results yield optimal designs under given constraints, with engineering insights drawn from load distribution, weight reduction, and vibration analysis. All computational results (numerical from python and graphical from MATLAB) agree, validating the formulations. Overall, the thesis demonstrates how combining optimization algorithms with mechanics principles can yield effective structural designs.

Keywords

Optimization, linear programming, nonlinear programming, multi-objective optimization, structural design.

LAMICHANE, Dipesh. *Optimization in engineering problems*. Master's Thesis. Eva MRAZKOVÁ (supervisor). Brno: Brno University of Technology, Faculty of Mechanical Engineering, 2025.

I declare that I wrote the diploma thesis *Optimization in engineering problems* independently under the guidance of *Ing. Mgr. Eva Mrázková, Ph.D.* using the literature included in the list of references.

Dipesh Lamichhane

I would like to express my sincere gratitude to my supervisor, Ing. Mgr. Eva Mrázková, Ph.D., for her dedicated guidance, valuable advice, and continuous support throughout the duration of my master's thesis.

Dipesh Lamichhane

Contents

1 Introduction

Optimization plays a crucial role in modern engineering design, enabling engineers to make the best possible decisions under given constraints. In many structural and mechanical problems, one must balance competing objectives, such as minimizing weight while maximizing stiffness or natural frequency subject to physical and design constraints. This thesis is motivated by the need to systematically formulate optimization problems and solve them using computational techniques. The scope of this work is to introduce fundamental optimization concepts, review the necessary engineering mechanics principles, and apply these tools to solve a set of representative engineering design problems. By integrating theory with computation, the thesis demonstrates how optimization models can be formulated and solved for practical engineering systems. The organization of the thesis is as follows.

Chapter 2 reviews optimization theory, beginning with discussion on two main type of optimization: unconstrained and constrained optimizations. We discuss feasible, local, and global solutions and provide examples of graphical solution methods for simple problems. Linear programming (LP) methods are covered next, including the simplex method, the dual simplex method, and interior-point methods. Nonlinear programming (NLP) is then introduced, with a discussion of optimality conditions and methods such as Sequential Quadratic Programming and Sequential Least Squares Quadratic Programming . Finally, multi-objective optimization (MOO) is examined, including approaches like the weighted-sum method and evolutionary algorithms (e.g. NSGA-II) for finding Pareto-optimal solutions.

Chapter 3 covers the basic mechanics principles needed to model the structural design problems. We review static equilibrium, forces and moments, stress and strain relations, and bending of beams under loads (including the flexure formula and beam deflection under uniform or point loads). The concepts of buckling (Euler's critical load for columns) and basic vibration theory (natural frequencies of a simple column or beam) are also introduced. These mechanics fundamentals provide the physical constraints and performance criteria used in later optimization problems.

Chapter 4 describes the computational tools used in this thesis. We introduce MATLAB and Python as the primary platforms for implementation, along with optimization libraries such as SciPy (in Python) and built-in solvers in MATLAB. This chapter explains how linear and nonlinear programs can be implemented in these environments and highlights the specific functions (e.g. linprog, fmincon, SciPy's minimize, and Python-based evolutionary algorithms) that will be employed in the case studies.

Chapter 5 presents four example problems that demonstrate the application of optimization to engineering systems. The problems include (1) a scaffolding support system, (2) a beam design optimization, (3) a truss design optimization, and (4) a water-tank column optimization with multi-objective criteria (minimizing mass and maximizing natural frequency). For each problem, we derive the equilibrium equations and constraints from mechanics, formulate the optimization model (objectives and constraints), and solve it using the methods and tools discussed earlier. Solutions are obtained using MATLAB and Python, and results are analyzed to draw engineering insights.

Chapter 6 concludes the thesis by summarizing the key findings and highlighting the main contributions of the work.

2 Optimization

2.1 Unconstrained and Constrained Optimization

Optimization problems can be broadly classified into two categories, unconstrained and constrained, depending on whether the decision variables are subject to any restrictions beyond the objective.
In an unconstrained problem, one seek to maximize or minimize the objective function without any further conditions on the decision variables.

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad \text{or} \quad \max_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

Such problems arise when the only goal is to drive the performance measure as far as possible in one direction, for example, locating the highest peak of a smooth surface. Unconstrained problems can also arise as a reformulations of constrained optimization problems, in which the constraints are replaced by penalization terms added to objective function that discourage constraint violations [12]. The optimality conditions are given by the vanishing of the gradient ($\nabla f(\mathbf{x}^*) = \mathbf{0}$) and, when f is twice differentiable, the positive (for minimization) definiteness of the Hessian ($\nabla^2 f(\mathbf{x}^*)$).

In this thesis, we focus on constrained optimization, where the constraints play a crucial role in determining the solution. Most real-world engineering design problems impose requirements on the variables, limits on material stresses, physical dimensions, or allowable deflections. Depending on the mathematical form of the objective and constraint functions, these problems fall into either linear or nonlinear categories. Furthermore, based on how many objectives are being optimized, they are classified as single-objective or multi-objective problems. Each of these classes will be examined in detail in the following subsections.

Decision, Constraint and Objective

The three main aspects of formulating optimization models are: (a) the available decision options, (b) the constraints that restrict these choices, and (c) the objectives that determine which decisions are more preferable [16].

- In optimization models, variables represent the decisions to be made.
- Variable-type constraints define the permissible values for decision variables, specifying their domain.
- Main constraints describe the relationships and limitations beyond variable types that govern how decision variables can be assigned values. These constraints can be either equality or inequality constraints.
- Objective functions express the outcomes of decisions in a quantifiable form, which the model seeks to either maximize or minimize.

A typical optimization model is formulated as:

min or max (objective function(s))
subject to:
(main constraints)
(variable-type constraints)

Feasible, Optimal, Local and Global Solutions

A choice of values for the decision variable that satisfies all the constraints of the given problem is called a feasible solution. The set of all such feasible solutions is known as the feasible region (or

set). The best feasible solution i.e., the solution that optimizes (either minimizes or maximizes) the objective function is called the optimal solution.

A solution is called a local solution (local minimum for a minimization problem) if it is feasible and contains no points that are both feasible and have a smaller objective value for a sufficiently small neighborhood surrounding it. A solution is called a global solution (global minimum for a minimization problem) if it is feasible and no other feasible solution has a smaller objective value [16].

Graphical Solution

Optimization problems involving two decision variables can be displayed graphically, and the solution can be obtained by inspection [18]. By visualizing the problem graphically, we can gain deeper insights into the underlying mathematical structures, patterns, and relationships. To find the solution, we plot the equality or inequality constraints with respect to the two variables and overlay the contours of the objective function on the graph. Contours are lines or curves where all points share the same function value. The approximate optimal solution can then be identified by visually inspecting the feasible region within the graph.

2.2 Linear Programming

Linear Programming (LP) is a mathematical technique to find the optimal solution (i.e. minimum or maximum) for a problem with linear relationships. LP minimizes or maximizes a linear objective function subject to linear equalities and inequalities constraints. LP has been widely used to solve military, economic, industrial, and societal problems [17]. A maximization (or minimization) problem can be easily converted into a minimization (or maximization) problem by multiplying the objective function's coefficients by -1 ,with the original objective value being the negative of the new problem's optimal objective value. Any general LP problem can be formulated in canonical and standard form [2]:

Canonical Form:

When a minimization problem is in canonical form, all variables are non-negative and all constraints are of the $\geq$ type.

$$\text{Minimize } \mathbf{c}^T\mathbf{x}$$
$$\text{Subject to: } \mathbf{A}\mathbf{x} \geq \mathbf{b},$$
$$\mathbf{x} \geq \mathbf{0}$$

Where:

- $\mathbf{x}$ = Column vector of decision variables

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$$

- $\mathbf{c}$ = Column vector of coefficients in the objective function.

$$\mathbf{c} = \begin{bmatrix} c_1 & c_2 & \dots & c_n \end{bmatrix}^T$$

- $\mathbf{A}$ = Constraint matrix (each row represents a constraint, and each column represents a variable).

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- **b** = Column vector of resource limits (right-hand side of constraints).

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_m \end{bmatrix}^T$$

Standard form:
When a linear problem is in standard form, all constraints are equalities and all variables are non-negative.

$$\text{Minimize } \mathbf{c}^T\mathbf{x}$$

$$\text{Subject to: } \mathbf{A}\mathbf{x} = \mathbf{b},$$

$$\mathbf{x} \geq \mathbf{0}$$

The problem in canonical form above can be easily converted to standard form by introducing non-negative surplus variables to transform inequalities into equalities.

$$\text{Minimize } \mathbf{c}^T\mathbf{x}$$

$$\text{Subject to: } \mathbf{A}\mathbf{x} - \mathbf{s} = \mathbf{b},$$

$$\mathbf{x} \geq \mathbf{0}, \mathbf{s} \geq \mathbf{0}$$

where **s** = Column vector of surplus variables.

$$\mathbf{s} = \begin{bmatrix} s_1 & s_2 & \dots & s_m \end{bmatrix}^T$$

Duality Theory
Every linear programming problem has a corresponding linear program, known as its dual, where a solution to the original (primal) problem also provides the solution to its dual [17]. The LP in canonical form can be easily expressed in its symmetric dual form as follows:
Primal:

$$\text{Minimize } f = \mathbf{c}^T\mathbf{x}$$

$$\text{Subject to: } \mathbf{A}\mathbf{x} \geq \mathbf{b},$$

$$\mathbf{x} \geq \mathbf{0}$$

Dual:

$$\text{Maximize } v = \mathbf{b}^T\mathbf{y}$$

$$\text{Subject to: } \mathbf{A}^T\mathbf{y} \leq \mathbf{c},$$

$$\mathbf{y} \geq \mathbf{0}$$

where **y** = Column vector of decision variables for the dual problem

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix}^T$$

Duality Theorems:
These theorems are taken from [14].

Theorem 2.1. *The dual of the dual is the primal.*

Theorem 2.2. *Any feasible solution of the primal gives an f value greater than or at least equal to the v value obtained by any feasible solution of the dual.*

Theorem 2.3. *If both primal and dual problems have feasible solutions, both have optimal solutions and minimum f = maximum v.*

Theorem 2.4. *If either the primal or the dual problem has an unbounded solution, the other problem is infeasible.*

The Simplex Method

The simplex method, introduced by G.B. Dantzig, is an iterative algorithm for solving linear programming problems in *standard form*. At each iteration, one moves from one basic feasible solution to another by pivoting on the basis matrix until optimality (or unboundedness) is detected. We follow the version of simplex algorithm as described in [2]. Before presenting the simplex steps we first need to understand the notion of a basic feasible solution.
Consider the linear system

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{0},$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Assume $\operatorname{rank}(\mathbf{A}, \mathbf{b}) = \operatorname{rank}(\mathbf{A}) = m$. By rearranging the columns of $\mathbf{A}$ if necessary, write

$$\mathbf{A} = \big[\, \mathbf{B} \,,\, \mathbf{N} \big],$$

where $\mathbf{B} \in \mathbb{R}^{m\times m}$ is nonsingular and $\mathbf{N} \in \mathbb{R}^{m\times(n-m)}$. Correspondingly partition $\mathbf{x}$ as

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{pmatrix}.$$

Then

$$\mathbf{x}_B = \mathbf{B}^{-1}\,\mathbf{b}, \qquad \mathbf{x}_N = \mathbf{0}$$

is called a *basic solution* of $\mathbf{Ax} = \mathbf{b}$. If additionally $\mathbf{x}_B \geq \mathbf{0}$, it is a *basic feasible solution* (BFS). Here $\mathbf{B}$ is the *basic matrix* (or *basis*) and $\mathbf{N}$ the *nonbasic matrix*. The entries of $\mathbf{x}_B$ are the *basic (dependent) variables*, and those of $\mathbf{x}_N$ the *nonbasic (independent) variables*. A BFS is *nondegenerate* if $\mathbf{x}_B > \mathbf{0}$ (strictly) and *degenerate* otherwise.

The Simplex Algorithm (Minimization)
Initialization:
Choose an initial basis $\mathbf{B}$ whose columns are a feasible set of m columns of $\mathbf{A}$, compute $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$, and set $\mathbf{x}_N = \mathbf{0}$. If $\bar{\mathbf{b}} \geq \mathbf{0}$, this gives an initial basic feasible solution.
Main Loop

1. **Compute current solution and objective.**
   Solve
   $$\mathbf{B}\,\mathbf{x}_B = \mathbf{b} \quad \Longrightarrow \quad \mathbf{x}_B = \bar{\mathbf{b}}, \quad \mathbf{x}_N = \mathbf{0}.$$
   The current objective value is $z = \mathbf{c}_B^\top\,\mathbf{x}_B$. This yields the basic solution $(\mathbf{x}_B, \mathbf{x}_N)$.
2. **Pricing (choose entering variable).**
   Compute the vector of simplex multipliers
   $$\mathbf{w}^\top = \mathbf{c}_B^\top\,\mathbf{B}^{-1},$$

where $\mathbf{c} \in \mathbb{R}^n$ is cost-coefficient vector, partitioned $\mathbf{c} = (\mathbf{c}_B, \mathbf{c}_N)$ then for each $j \in J$ (nonbasic index) compute the reduced cost

$$z_j - c_j = \mathbf{w}^\top \mathbf{a}_j - c_j.$$

where $\mathbf{a}_j$ is the j-th column of $\mathbf{A}$ and c_j is the j-th component of $\mathbf{c}$
Let $z_k - c_k = \max_{j \in J}\{z_j - c_j\}$. If $z_k - c_k \leq 0$, then stop: the current BFS is optimal. Otherwise, go to Step 3 with x_k entering the basis.

3. **Boundedness check.**
Solve

$$\mathbf{B}\,\mathbf{y}_k = \mathbf{a}_k \quad \Longrightarrow \quad \mathbf{y}_k = \mathbf{B}^{-1}\mathbf{a}_k.$$

If $\mathbf{y}_k \leq \mathbf{0}$, stop: the problem is unbounded in direction x_k. Otherwise proceed.

4. **Minimum-ratio test (choose leaving variable).**
Let $\bar{b}_i$ be the ith component of $\bar{\mathbf{b}}$ and y_{ik} the ith component of $\mathbf{y}_k$. Compute

$$\frac{\bar{b}_r}{y_{rk}} = \min_{1 \leq i \leq m} \{\frac{\bar{b}_i}{y_{ik}} : y_{ik} > 0\}$$

Index r identifies the leaving variable x_{B_r}. Replace column $\mathbf{a}_{B_r}$ (r-th column in $\mathbf{B}$) with $\mathbf{a}_k$ (k-th column in $\mathbf{A}$), update the basis indices and J, and return to Step 1.

The Dual Simplex Method

When it is hard to obtain an initial basis with $\bar{b}_i \geq 0$ for all i, one can instead look for a basis that is *dual feasible*, i.e. satisfies $z_j - c_j \leq 0 \quad$ for all nonbasic j in a minimization problem, even if some basic variables are negative. The dual simplex method takes such a starting tableau and performs pivots that maintain *dual feasibility* and *complementary slackness* while steadily eliminating primal infeasibilities. The dual simplex algorithm as described in [2] is presented below.

Dual Simplex Algorithm (Minimization Problem)
Initialization Step
Choose an initial basis $\mathbf{B}$ for which

$$z_j - c_j = \mathbf{c}_B^\top \mathbf{B}^{-1}\mathbf{a}_j \; - \; c_j \; \leq 0 \quad \text{for all } j.$$

Main Step

1. **Check primal feasibility**
Compute $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$.
   - If $\bar{b}_i \geq 0$ for all i, stop; the current basis is both primal- and dual-feasible (optimal).
   - Otherwise, choose a leaving row r such that $\bar{b}_r = \min_{1 \leq i \leq m} \{\bar{b}_i\} < 0$.
2. **Check dual unboundedness & select entering variable**
For each nonbasic column j, let y_{rj} be the rth entry of $\mathbf{y}_j = \mathbf{B}^{-1}\mathbf{a}_j$.
   - If $y_{rj} \geq 0$ for all j, stop; the dual is unbounded $\Longrightarrow$ the primal is infeasible.
   - Otherwise, select the pivot column k from the following minimum ratio test

$$\frac{z_k - c_k}{y_{rk}} = \min_j \{\frac{z_j - c_j}{y_{rj}} : y_{rj} < 0\}.$$

3. **Pivot**
Perform the basis pivot on the element y_{rk} in row r and column k. Update $\mathbf{B}, \mathbf{N}, \bar{\mathbf{b}}$ and return to Step 1.

Interior Point Method

The simplex method navigates along the edges of the feasible polytope, evaluating a sequence of vertices until it identifies the optimal one. In contrast, interior-point methods move towards the boundary of the feasible set only in the limit, approaching the solution from inside the region without ever touching the boundary. The primal-dual interior point method as described in [12] is presented below.

Primal–Dual Interior-Point Method

We consider the primal-dual pair of linear programs in standard form:

$$\min_{\mathbf{x}} \ \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{A}\,\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}, \tag{P}$$

$$\max_{\boldsymbol{\lambda}} \ \mathbf{b}^\top \boldsymbol{\lambda} \quad \text{s.t.} \quad \mathbf{A}^\top \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c},\ \mathbf{s} \geq \mathbf{0}, \tag{D}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ has full row rank, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, and $\mathbf{x}, \mathbf{s} \in \mathbb{R}^n$, $\boldsymbol{\lambda} \in \mathbb{R}^m$. Optimality is characterized by the Karush–Kuhn–Tucker (KKT) conditions:

$$\mathbf{A}^\top \boldsymbol{\lambda} + \mathbf{s} = \mathbf{c}, \tag{2.1a}$$

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}, \tag{2.1b}$$

$$\mathbf{X}\,\mathbf{S}\,\mathbf{e} = \mathbf{0}, \tag{2.1c}$$

$$\mathbf{x}, \mathbf{s} \geq \mathbf{0}, \tag{2.1d}$$

where $\mathbf{X} = \operatorname{diag}(x_1, \ldots, x_n)$, $\mathbf{S} = \operatorname{diag}(s_1, \ldots, s_n)$, and $\mathbf{e} = (1, \ldots, 1)^\top$.

Primal–dual interior-point methods generate a sequence $\{(\mathbf{x}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k)\}$ satisfying the strict inequalities $\mathbf{x}^k > \mathbf{0}$, $\mathbf{s}^k > \mathbf{0}$ and drive the pairwise products $x_i s_i$ toward zero. Define the *duality measure*

$$\mu = \frac{\mathbf{x}^\top \mathbf{s}}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i s_i,$$

which is positive whenever $\mathbf{x}, \mathbf{s} > \mathbf{0}$. At each iteration we select a *centering parameter* $\sigma \in [0, 1]$ and solve a damped Newton system for the perturbed KKT residuals,

$$\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = \begin{pmatrix} \mathbf{A}^\top \boldsymbol{\lambda} + \mathbf{s} - \mathbf{c} \\ \mathbf{A}\,\mathbf{x} - \mathbf{b} \\ \mathbf{X}\,\mathbf{S}\,\mathbf{e} \end{pmatrix} = \mathbf{0},$$

with the modification $\mathbf{X}\,\mathbf{S}\,\mathbf{e} = \sigma\,\mu\,\mathbf{e}$. Writing $\mathbf{r}_b = \mathbf{A}\,\mathbf{x} - \mathbf{b}$, $\mathbf{r}_c = \mathbf{A}^\top \boldsymbol{\lambda} + \mathbf{s} - \mathbf{c}$, the Newton system becomes

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}^\top & \mathbf{I} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{S} & \mathbf{0} & \mathbf{X} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{pmatrix} = - \begin{pmatrix} \mathbf{r}_c \\ \mathbf{r}_b \\ \mathbf{X}\,\mathbf{S}\,\mathbf{e} - \sigma\,\mu\,\mathbf{e} \end{pmatrix}.$$

A suitable step length $\alpha \in (0, 1]$ is then chosen so that $\mathbf{x} + \alpha\,\Delta\mathbf{x} > \mathbf{0}$ and $\mathbf{s} + \alpha\,\Delta\mathbf{s} > \mathbf{0}$. Repeating this process with a decreasing schedule of σ and updating $\mu = \frac{(\mathbf{x}+\alpha\Delta\mathbf{x})^\top (\mathbf{s}+\alpha\Delta\mathbf{s})}{n}$ yields convergence to an optimal primal-dual pair.

Algorithm (Primal–Dual Path-Following).

1. **Initialization.** Choose $(\mathbf{x}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ with $\mathbf{x}^0 > \mathbf{0}$, $\mathbf{s}^0 > \mathbf{0}$. Set tolerances $\epsilon_{\text{pri}}, \epsilon_{\text{dual}}, \epsilon_{\text{comp}} > 0$.
2. **For** $k = 0, 1, 2, \ldots$ **do:**

(a) Compute residuals:

$$\mathbf{r}_b^k = \mathbf{A}\mathbf{x}^k - \mathbf{b}, \quad \mathbf{r}_c^k = \mathbf{A}^\top \boldsymbol{\lambda}^k + \mathbf{s}^k - \mathbf{c}, \quad \mu^k = \frac{(\mathbf{x}^k)^\top \mathbf{s}^k}{n}.$$

(b) Termination Check:
If

$$\|\mathbf{r}_b^k\| \le \epsilon_{\text{pri}}, \quad \|\mathbf{r}_c^k\| \le \epsilon_{\text{dual}}, \quad \mu^k \le \epsilon_{\text{comp}},$$

then stop and return $(\mathbf{x}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k)$ as the solution.

(c) Infeasibility Detection:
If residuals remain above tolerance for many iterations while $\mu^k \to 0$, declare:

- Primal infeasibility if $\|\mathbf{r}_b^k\|$ does not decrease.
- Dual infeasibility or unbounded primal if $\|\mathbf{r}_c^k\|$ stagnates and μ^k remains bounded away from 0.

(d) Select centering parameter $\sigma_k \in [0, 1]$.

(e) Solve for $\Delta\mathbf{x}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}$:

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}^\top & \mathbf{I} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{S}^k & \mathbf{0} & \mathbf{X}^k \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{pmatrix} = - \begin{pmatrix} \mathbf{r}_c^k \\ \mathbf{r}_b^k \\ \mathbf{X}^k\mathbf{S}^k\mathbf{e} - \sigma_k\,\mu^k\,\mathbf{e} \end{pmatrix}.$$

(f) Compute step length $\alpha_k = \max\{\alpha \in (0, 1] : \mathbf{x}^k + \alpha\Delta\mathbf{x} > \mathbf{0},\ \mathbf{s}^k + \alpha\Delta\mathbf{s} > \mathbf{0}\}$.

(g) Update

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\,\Delta\mathbf{x}, \quad \boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \alpha_k\,\Delta\boldsymbol{\lambda}, \quad \mathbf{s}^{k+1} = \mathbf{s}^k + \alpha_k\,\Delta\mathbf{s},$$

3. **End for.**

2.3 Non-linear Programming

Non-linear programming (NLP) is another type of optimization where the objective function or the constraints are nonlinear. NLP is widely used in engineering because engineering design problems are mostly nonlinear [18]. The primary approaches to designing minimum-weight structures rely on mathematical programming methods, such as NLP, in combination with structural analysis techniques and other advanced numerical methods [3]. A general constrained non-linear programming problem can be formulated as following:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to: } g_i(\mathbf{x}) \le 0, \quad i = 1, 2, \ldots, m, \\ &\qquad\qquad h_j(\mathbf{x}) = 0, \quad j = 1, 2, \ldots, l. \end{aligned} \tag{2.2}$$

where,

- $\mathbf{x}$ is a decision vector of n components,

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$$

- $f, g_i, h_j : \mathbb{R}^n \to \mathbb{R}$ are differentiable functions
- $f(\mathbf{x})$ is the objective function
- $g_i(\mathbf{x})$ for $i = 1, \ldots, m$ are the inequality constraint functions
- $h_j(\mathbf{x})$ for $j = 1, \ldots, l$ are the equality constraint functions

Optimality Conditions

Consider the problem (2.2) and denote by M the set of feasible solutions. Define a Lagrangian function by

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \, g_i(\mathbf{x}) + \sum_{j=1}^{l} \mu_j \, h_j(\mathbf{x}), \quad \lambda_i \geq 0$$

Convexity:
We say that problem (2.2) is convex if functions $f, g_i, \forall i$ are convex and $h_i, \forall j$ are affine.

Basic Constraint Qualification (CQ) Conditions:

- **Slater CQ:** $\exists \mathbf{x}^* \in M$ such that $g_i(\mathbf{x}^*) < 0,$ for all i and the gradients $\nabla_{\mathbf{x}} h_j(\mathbf{x}^*), \quad j = 1, \ldots, l$ are linearly independent.
- **Linear Independence CQ:** for $\mathbf{x}^* \in M$, all active constraint gradients

  $$\nabla_{\mathbf{x}} g_i(\mathbf{x}^*) \text{ with } g_i(\mathbf{x}^*) = 0, \quad \nabla_{\mathbf{x}} h_j(\mathbf{x}^*), \quad j = 1 \ldots l$$

  are linearly independent.

Karush–Kuhn–Tucker Conditions:
If $\mathbf{x}^*$ is a local minimizer of the problem (2.2) and satisfies the CQ condition then

1. $g_i(\mathbf{x}^*) \leq 0$ for $1 \leq i \leq m \quad , h_j(\mathbf{x}^*) = 0$ for $1 \leq j \leq l \quad$ (*feasibility*)
2. $\lambda_i \, g_i(\mathbf{x}^*) = 0, \quad \lambda_i \geq 0, \quad i = 1, \ldots, m \quad$ (*complementarity*)
3. $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{0} \quad$ (*stationarity*)

Any point$(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu})$ that satisfies the above conditions is called a KKT point.

Second Order Sufficient Condition (SOSC)
If problem (2.2) is convex and satisfies KKT conditions at $\mathbf{x}^*$ then $\mathbf{x}^*$ is a global minima.
When the problem is not convex, solutions of the KKT conditions need not correspond to global optima. The (SOSC) can be used to verify whether the KKT point is at least a local minimum. Let

$$I_g(\mathbf{x}) \;=\; \{\, i : g_i(\mathbf{x}) = 0\}$$

be the index set of active inequality constraints at $\mathbf{x}$, and partition it as

$$I_g(\mathbf{x}) \;=\; I_g^0(\mathbf{x}) \;\cup\; I_g^+(\mathbf{x}),$$

where

$$I_g^0(\mathbf{x}) = \{\, i \in I_g(\mathbf{x}) : \lambda_i = 0\},$$
$$I_g^+(\mathbf{x}) = \{\, i \in I_g(\mathbf{x}) : \lambda_i > 0\}.$$

Assume all functions are twice continuously differentiable. We say that the SOSC holds at a KKT point $(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu})$ if for every nonzero direction $\mathbf{z} \in \mathbb{R}^n$ satisfying

$$\begin{aligned}
\mathbf{z}^T \nabla_{\mathbf{x}} g_i(\mathbf{x}^*) \;&=\; 0, \quad i \in I_g^+(\mathbf{x}^*),\\
\mathbf{z}^T \nabla_{\mathbf{x}} g_i(\mathbf{x}^*) \;&\leq\; 0, \quad i \in I_g^0(\mathbf{x}^*),\\
\mathbf{z}^T \nabla_{\mathbf{x}} h_j(\mathbf{x}^*) \;&=\; 0, \quad j = 1, \ldots, l
\end{aligned}$$

it holds that

$$\mathbf{z}^T \nabla^2_{\mathbf{xx}} \mathcal{L}\big(\mathbf{x}^*,\, u^*,\, v^*\big)\, \mathbf{z} \;>\; 0.$$

Then $\mathbf{x}^*$ is a strict local minimum of the nonlinear programming problem (2.2).
The optimality conditions described in this subsection are based on [5].

NLP method: SQP

Sequential Quadratic Programming (SQP) is one of the most effective methods for solving general constrained problems with smooth objective and constraint functions. At each iteration, SQP builds a *quadratic* model of the Lagrangian function and a *linear* model of the constraints, solving a Quadratic Program (QP) subproblem that yields a search direction approximating the Newton step for the Karush–Kuhn–Tucker (KKT) conditions.
We consider the general nonlinear program in the form of (2.2):

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \\ &\text{s.t.} \quad \begin{cases} h_j(\mathbf{x}) = 0, & j = 1,\dots,l, \\ g_i(\mathbf{x}) \le 0, & i = 1,\dots,m, \end{cases} \end{aligned} \tag{2.3}$$

where f, g_i, h_j are twice continuously differentiable and the feasible set is nonempty.

Lagrangian and KKT conditions.
Define the Lagrangian

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \;=\; f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i\, g_i(\mathbf{x}) + \sum_{j=1}^{l} \mu_j\, h_j(\mathbf{x}), \quad \lambda_i \ge 0.$$

The KKT conditions at $(\mathbf{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*)$ are

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*) = \mathbf{0}, \tag{2.4a}$$
$$h_j(\mathbf{x}^*) = 0, \quad j = 1,\dots,l, \tag{2.4b}$$
$$g_i(\mathbf{x}^*) \le 0,\ \lambda_i^* \ge 0, \quad i = 1,\dots,m, \tag{2.4c}$$
$$\lambda_i^*\, g_i(\mathbf{x}^*) = 0, \quad i = 1,\dots,m. \tag{2.4d}$$

QP subproblem at $(\mathbf{x}^k,\boldsymbol{\lambda}^k,\boldsymbol{\mu}^k)$:
Linearize the constraints and use a quadratic model of the Lagrangian:

$$\begin{aligned} &\min_{\delta\mathbf{x}\in\mathbb{R}^n} \tfrac{1}{2}\,\delta\mathbf{x}^{\top} Y^k\,\delta\mathbf{x} \;+\; \nabla f(\mathbf{x}^k)^{\top}\delta\mathbf{x}, \\ &\text{s.t. } \nabla h(\mathbf{x}^k)\,\delta\mathbf{x} \;=\; -\,h(\mathbf{x}^k), \quad \nabla g(\mathbf{x}^k)\,\delta\mathbf{x} \;\le\; -\,g(\mathbf{x}^k), \end{aligned} \tag{QP^k}$$

where

$$Y^k \;=\; \nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k,\boldsymbol{\lambda}^k,\boldsymbol{\mu}^k) \;=\; \nabla^2 f(\mathbf{x}^k) + \sum_{i=1}^{m} \lambda_i^k\,\nabla^2 g_i(\mathbf{x}^k) + \sum_{j=1}^{\ell} \mu_j^k\,\nabla^2 h_j(\mathbf{x}^k),$$

and

$$\nabla g(\mathbf{x}^k) = \begin{bmatrix} \nabla g_1(\mathbf{x}^k)^{\top} \\ \vdots \\ \nabla g_m(\mathbf{x}^k)^{\top} \end{bmatrix}, \quad g(\mathbf{x}^k) = \begin{pmatrix} g_1(\mathbf{x}^k) \\ \vdots \\ g_m(\mathbf{x}^k) \end{pmatrix}, \quad \nabla h(\mathbf{x}^k) = \begin{bmatrix} \nabla h_1(\mathbf{x}^k)^{\top} \\ \vdots \\ \nabla h_l(\mathbf{x}^k)^{\top} \end{bmatrix}, \quad h(\mathbf{x}^k) = \begin{pmatrix} h_1(\mathbf{x}^k) \\ \vdots \\ h_\ell(\mathbf{x}^k) \end{pmatrix}.$$

Let $\delta\mathbf{x}^k$ be the solution of (QP^k) and $(\widehat{\boldsymbol{\lambda}}^{k+1}, \widehat{\boldsymbol{\mu}}^{k+1})$ the associated multipliers. Then update

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{x}^k + \delta\mathbf{x}^k, \\ \lambda_i^{k+1} &= \widehat{\lambda}_i^{k+1}, \quad i = 1,\dots,m, \\ \mu_j^{k+1} &= \widehat{\mu}_j^{k+1}, \quad j = 1,\dots,l. \end{aligned}$$

Terminate when $\|\delta \mathbf{x}^k\| \le \varepsilon$.

SQP Algorithm:

1. **Initialization.** Choose $\mathbf{x}^0$ satisfying $h(\mathbf{x}^0) = \mathbf{0}$, $g(\mathbf{x}^0) \le \mathbf{0}$, multipliers $\boldsymbol{\lambda}^0 \ge \mathbf{0}$, $\boldsymbol{\mu}^0$, and tolerance $\varepsilon > 0$. Set $k \leftarrow 0$.
2. **Form Hessian and Jacobians:**

$$Y^k = \nabla^2 f(\mathbf{x}^k) + \sum_{i=1}^{m} \lambda_i^k \nabla^2 g_i(\mathbf{x}^k) + \sum_{j=1}^{l} \mu_j^k \nabla^2 h_j(\mathbf{x}^k),$$

$$\nabla g(\mathbf{x}^k), \quad \nabla h(\mathbf{x}^k).$$

3. **Solve QP subproblem** (QP^k)**.** Obtain $\delta \mathbf{x}^k$ and $(\widehat{\boldsymbol{\lambda}}^{k+1}, \widehat{\boldsymbol{\mu}}^{k+1})$.
4. **Update:**

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta \mathbf{x}^k, \quad (\boldsymbol{\lambda}^{k+1}, \boldsymbol{\mu}^{k+1}) = (\widehat{\boldsymbol{\lambda}}^{k+1}, \widehat{\boldsymbol{\mu}}^{k+1}).$$

If $\|\delta \mathbf{x}^k\| \le \varepsilon$, stop; otherwise set $k \leftarrow k + 1$ and return to Step 2.

Notations:

- $\mathbf{x} \in \mathbb{R}^n$: primal vector; $\boldsymbol{\lambda} \in \mathbb{R}^m$, $\boldsymbol{\mu} \in \mathbb{R}^l$: multipliers.
- $f : \mathbb{R}^n \to \mathbb{R}$, $g_i, h_j : \mathbb{R}^n \to \mathbb{R}$: twice-differentiable.
- $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$: Lagrangian of (2.2).
- $Y^k = \nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^k, \boldsymbol{\lambda}^k, \boldsymbol{\mu}^k)$: Lagrangian Hessian.
- $\nabla g(\mathbf{x}^k)$, $\nabla h(\mathbf{x}^k)$: Jacobians of constraints.
- $g(\mathbf{x}^k)$, $h(\mathbf{x}^k)$: constraint values at iterate k.
- $\delta \mathbf{x}^k$: QP step; $(\widehat{\boldsymbol{\lambda}}^{k+1}, \widehat{\boldsymbol{\mu}}^{k+1})$: QP multipliers.
- ε: termination tolerance.

The SQP method described here is based on [1], which provides a detailed explanation.

NLP method: SLSQP

Sequential Least-Squares Quadratic Programming (SLSQP) is a popular variant of SQP that replaces the full QP solve by a linear-constrained least-squares subproblem and updates the Hessian approximation in factored form. As in SQP, at iterate $\mathbf{x}^k$ we maintain an approximation

$$\mathbf{B}_k \approx \nabla^2_{\mathbf{xx}} \mathcal{L}\big(\mathbf{x}^k, \boldsymbol{\lambda}^k, \boldsymbol{\mu}^k\big),$$

together with its symmetric $LDL^\top$ factorization

$$\mathbf{B}_k = \mathbf{L}_k \, \mathbf{D}_k \, \mathbf{L}_k^\top, \quad \mathbf{L}_k \text{ unit-lower-triangular}, \ \mathbf{D}_k \text{ diagonal}.$$

Define

$$\mathbf{R}_k = \mathbf{D}_k^{1/2} \mathbf{L}_k^\top, \qquad \text{and choose } \mathbf{q}_k \text{ such that } \mathbf{R}_k^\top \mathbf{q}_k = -\nabla f(\mathbf{x}^k).$$

Then solve the least-squares subproblem

$$\begin{aligned} \min_{\mathbf{d} \in \mathbb{R}^n} \ & \tfrac{1}{2} \left\| \mathbf{R}_k \, \mathbf{d} - \mathbf{q}_k \right\|_2^2 \\ \text{s.t. } & \nabla h(\mathbf{x}^k)^\top \mathbf{d} \ = \ -h(\mathbf{x}^k), \\ & \nabla g(\mathbf{x}^k)^\top \mathbf{d} \ \le \ -g(\mathbf{x}^k), \end{aligned} \tag{LSQ}$$

where

$$h(\mathbf{x}) = \begin{pmatrix} h_1(\mathbf{x}) \\ \vdots \\ h_l(\mathbf{x}) \end{pmatrix}, \quad g(\mathbf{x}) = \begin{pmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}) \end{pmatrix},$$

and

$$\nabla h(\mathbf{x}^k) = \begin{bmatrix} \nabla h_1(\mathbf{x}^k)^\top \\ \vdots \\ \nabla h_l(\mathbf{x}^k)^\top \end{bmatrix}, \quad \nabla g(\mathbf{x}^k) = \begin{bmatrix} \nabla g_1(\mathbf{x}^k)^\top \\ \vdots \\ \nabla g_m(\mathbf{x}^k)^\top \end{bmatrix}.$$

The solution $\mathbf{d}^k$ gives both a search direction and the multipliers $(\widehat{\boldsymbol{\lambda}}^{k+1}, \widehat{\boldsymbol{\mu}}^{k+1})$. After a line search along the merit function with step length α_k, set

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \, \mathbf{d}^k.$$

Rather than refactorizing from scratch, update the $LDL^\top$ factors by the formula

$$\mathbf{L}_{k+1} \, \mathbf{D}_{k+1} \, \mathbf{L}_{k+1}^\top = \mathbf{L}_k \, \mathbf{D}_k \, \mathbf{L}_k^\top + \frac{\mathbf{r}_k \, \mathbf{r}_k^\top}{\mathbf{r}_k^\top \mathbf{s}_k} - \frac{\mathbf{B}_k \, \mathbf{s}_k \, \mathbf{s}_k^\top \, \mathbf{B}_k}{\mathbf{s}_k^\top \, \mathbf{B}_k \, \mathbf{s}_k},$$

where

$$\mathbf{s}_k = \alpha_k \, \mathbf{d}^k, \qquad \mathbf{r}_k = \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k).$$

By replacing full QP solves and refactorizations with constrained least-squares subproblems and cheap rank-two factor updates, SLSQP achieves faster iterations, greater numerical robustness, and still retains SQP's strong local convergence properties. The SLSQP method described above is based on [10].

2.4 Multi-Objective Programming

In the previous sections, we discussed optimizations involving only one objective function. However, in practical applications, there may be multiple objective functions that need to be optimized. These objectives can sometimes conflict with each other. Therefore, multi-objective optimization (MOO) aims to find a solution that balances these conflicting objectives. In contrast to single-objective optimization (SOO), which usually yields a single solution (or sometimes multiple optima, such as local and global optima in non-convex problems), a multi-objective problem typically yields many optimal solutions, with the exception of when the objectives are not conflicting, in which case there is only one unique solution [13]. A general multiobjective optimization problem can be formulated as:

$$\begin{aligned} &\text{Minimize } \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T \\ &\text{Subject to:} \\ &\quad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ &\quad h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, l \end{aligned} \tag{2.5}$$

where,

- $\mathbf{x}$ is the decision vector of n components.
- $\mathbf{f}(\mathbf{x})$ contains the multiple objective functions.
- g_i and h_j represent inequality and equality constraint functions

Pareto Optimality

As the objectives in MOO are often conflicting, there will be many optimal solutions, all of which are equally good in the sense that each one of them is better than the rest in at least one objective. This means when we choose one optimal solution over another, one or more objectives improve while at least another one becomes worse. The solutions of an MOO problem are called the Pareto-optimal solutions or Edgeworth-Pareto optimal solutions after the two economists, Edgeworth and Pareto [13].

Definition: The set $\mathbf{x}^*, \mathbf{f}(\mathbf{x}^*)$ is said to be a Pareto-optimal solution for the problem (2.5), if and only if, no other feasible $\mathbf{x}$ exists such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, ..., k$ and $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ for at least one i. In this case we say that $\mathbf{f}(\mathbf{x}^*)$ dominates $\mathbf{f}(\mathbf{x})$.

When we map the Pareto-optimal set through the objective functions we obtain a subset of non-dominated points in the k-dimensional objective space, known as the Pareto front.

MOO Method: Normalized Weighted Approach

This method converts the MOO problem into SOO problems to obtain one Pareto optimal solution. The weighted sum method is one of the simplest ways to convert multiple objectives of the MOO problem into a single objective optimization problem. When objectives of the MOO problem have different units or scales, normalized weighted sum can be used to obtain a single objective function. Normalization ensures that each objective contributes comparably when the weight is adjusted.

The objectives in problem (2.5) can be converted into a single objective using the weighted sum of the objectives as following:

$$F(\mathbf{x}) = \sum_{i=1}^{k} w_i f_i(\mathbf{x}), \quad \sum_{i=1}^{k} w_i = 1, \quad w_i \geq 0.$$

For the normalized weighted approach, the optimal solution is computed for every objective independent of other objectives as an SOO problem, and the optimal values of the objective functions are then used to normalize the weighted sum of the objectives.
Let $\mathbf{x}_i^*$ be the optimal solution for

$$\min f_i(\mathbf{x}) \quad i = 1, \ldots, k$$

subject to constraints from the MOO (problem (2.5))

$$f_i^* = f_i(\mathbf{x}_i^*) \qquad i = 1, \ldots, k$$

$$F_{\text{norm}}(\mathbf{x}) = \sum_{i=1}^{k} w_i \frac{f_i(\mathbf{x})}{f_i^*}, \qquad \sum_{i=1}^{k} w_i = 1, \; w_i \geq 0.$$

Now the MOO problem can be formulated as SOO problem as following:

$$\begin{aligned} &\text{Minimize } F_{\text{norm}}(\mathbf{x}) \\ &\text{Subject to:} \\ &\quad g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ &\quad h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, l \end{aligned}$$

Different Pareto optimal trade-offs can be obtained by varying the weight vector $\mathbf{w} = (w_1, \ldots, w_k)$.

MOO Method: NSGA-II

The Nondominated Sorting Genetic Algorithm II (NSGA-II), introduced by Deb *et al.* [6], is an evolutionary method for multiobjective optimization. It addresses two central challenges in earlier multiobjective genetic algorithm frameworks: *elitism* (preserving the best solutions across generations) and *diversity maintenance* without user-tuned parameters. NSGA-II achieves these via three key mechanisms: (i) a fast $O(kN^2)$ nondominated-sorting procedure, (ii) explicit elitism by merging parent and offspring populations before truncation, and (iii) the crowding-distance metric, which quantifies the density of solutions in objective space to guide diversity-preserving selection.

i. Fast Nondominated Sorting

Fast nondominated sorting is the procedure by which NSGA-II partitions a population $\mathbf{R}$ of size $2N$ into a sequence of Pareto-fronts $F_1, F_2, \ldots$. Each front F_ℓ contains those solutions that are nondominated within the remaining set. Concretely, for each individual $\mathbf{x} \in \mathbf{R}$ one maintains

- a domination count $n(\mathbf{x})$, initially zero,
- a list $S(\mathbf{x})$ of individuals dominated by $\mathbf{x}$.

Then for every ordered pair $(\mathbf{p}, \mathbf{q})$:

1. If $\mathbf{p}$ dominates $\mathbf{q}$ (i.e. $f_i(\mathbf{p}) \leq f_i(\mathbf{q})$ for all i and $<$ for some i), add $\mathbf{q}$ to $S(\mathbf{p})$.
2. Else if $\mathbf{q}$ dominates $\mathbf{p}$, increment $n(\mathbf{p})$.

All $\mathbf{x}$ with $n(\mathbf{x}) = 0$ form the first front F_1. Removing F_1, decrement $n(\mathbf{q})$ for each $\mathbf{q} \in S(\mathbf{x})$, $\mathbf{x} \in F_1$; those that reach zero become F_2, and so on.

ii. Elitism

Elitism in NSGA-II ensures that the best solutions found so far are never lost. At each generation:

$$\mathbf{R}_t = \mathbf{P}_t \cup \mathbf{Q}_t$$

combines the parent population $\mathbf{P}_t$ and the offspring $\mathbf{Q}_t$. The next parent population $\mathbf{P}_{t+1}$ is then selected by filling from the lowest-rank fronts $F_1, F_2, \ldots$ up to size N, using the crowded-comparison operator to break ties if a front would overfill. By merging parents and children before truncation, NSGA-II guarantees that any nondominated solution discovered in $\mathbf{P}_t$ that remains nondominated in $\mathbf{R}_t$ will survive into $\mathbf{P}_{t+1}$.

iii. Crowding Distance

The crowding distance $d(\mathbf{x})$ is a measure of how isolated an individual $\mathbf{x}$ is from others in its Pareto front, promoting diversity. For each front F_ℓ:

1. Initialize $d(\mathbf{x}) = 0$ for all $\mathbf{x} \in F_\ell$.
2. For each objective $m = 1, \ldots, k$:
   - Sort F_ℓ in ascending order of f_m.
   - Assign $d(\mathbf{x}_{\min}) = d(\mathbf{x}_{\max}) = \infty$ to the boundary solutions.
   - For $i = 2, \ldots, |F_\ell| - 1$, add the normalized gap

$$d(\mathbf{x}_i) \mathrel{+}= \frac{f_m(\mathbf{x}_{i+1}) - f_m(\mathbf{x}_{i-1})}{f_m^{\max} - f_m^{\min}}.$$

Higher $d(\mathbf{x})$ indicates a less crowded location in objective space, so when selecting among individuals of equal rank, NSGA-II prefers those with larger crowding distance under the operator $\prec_n$.

NSGA-II Algorithm

1. **Initialization.** Generate an initial population $\mathbf{P}_0$ of size N at random and evaluate $\mathbf{f}(\mathbf{x})$ for each $\mathbf{x} \in \mathbf{P}_0$.
2. **For** $t = 0, 1, 2, \ldots$ **until termination:**
   (a) *Combine & sort:* Form $\mathbf{R}_t = \mathbf{P}_t \cup \mathbf{Q}_t$. Perform fast nondominated sorting on $\mathbf{R}_t$ to identify fronts $F_1, F_2, \ldots$, and compute $r(\mathbf{x})$.
   (b) *Crowding distance assignment:* For each front F_ℓ, sort its members in ascending order of each objective and assign $d(\mathbf{x})$ as the sum of normalized gaps to neighbors.
   (c) *Selection of new parents:* Fill $\mathbf{P}_{t+1}$ by adding entire fronts $F_1, F_2, \ldots$ in order until $|\mathbf{P}_{t+1}| + |F_\ell| > N$. Then sort F_ℓ by $\prec_n$ and take the best $N - |\mathbf{P}_{t+1}|$ members.
   (d) *Create offspring:* Apply binary-tournament selection using $\prec_n$, then crossover and mutation, to $\mathbf{P}_{t+1}$ to form $\mathbf{Q}_{t+1}$ of size N. Evaluate $\mathbf{f}$ for each $\mathbf{x} \in \mathbf{Q}_{t+1}$.
3. **End For.**

Upon termination one obtains the final nondominated front(s) in $\mathbf{P}_t$ as an approximation to the Pareto set.

Notations:

$\mathbf{f}(\mathbf{x})$ objective vector of individual $\mathbf{x}$.

$r(\mathbf{x})$ nondomination rank (level) of $\mathbf{x}$.

$\prec_n$ crowded-comparison operator: $\mathbf{x} \prec_n \mathbf{y}$ if $\big(r(\mathbf{x}) < r(\mathbf{y})\big)$ or $\big(r(\mathbf{x}) = r(\mathbf{y})$ and $d(\mathbf{x}) > d(\mathbf{y})\big)$.

3 Basic Concepts of Mechanics

3.1 Force, Moment and Equilibrium

Force:
Force is the action of one body on another that can cause a body to move or change shape in the direction of its action. It is characterized by its magnitude (measured in Newtons), its direction, and its point of application.
The force vector can be resolved into two-dimensional rectangular components using the parallelogram rule:

$$\mathbf{F} = \mathbf{F}_x + \mathbf{F}_y,$$

where $\mathbf{F}_x$ and $\mathbf{F}_y$ are the components in the x- and y-directions, respectively. These components are given by:

$$F_x = F\cos\theta, \quad F_y = F\sin\theta,$$

and the magnitude of the force can be recovered by:

$$F = \sqrt{F_x^2 + F_y^2},$$

with θ being the angle between the applied force and its x component.

Moment:
Force tends to rotate a body about an axis (which may be any line not intersecting or parallel to the force's line of action). This rotational effect is called the *moment* of the force. Moment $\mathbf{M}$ is a vector perpendicular to the plane of the body. It is defined as:

$$\mathbf{M} = \mathbf{r} \times \mathbf{F},$$

where $\mathbf{r}$ is the position vector from the reference point to any point on the line of action of $\mathbf{F}$.

If d is the perpendicular distance from the reference point to the line of action, then the magnitude of the moment is

$$M = Fd.$$

Equilibrium:
A body is in equilibrium when the resultant force and the total moment acting on it are zero:

$$\sum \mathbf{F} = 0 \quad \text{and} \quad \sum \mathbf{M} = 0.$$

In two dimensions, this is expressed as:

$$\sum F_x = 0, \quad \sum F_y = 0, \quad \sum M_z = 0.$$

The definitions in this subsection are taken from [11], [9].

3.2 Stress and Strain

Stress:
Stress is defined as the internal force per unit area produced within a body by an external force. It is denoted by σ and given by:

$$\sigma = \frac{F}{A},$$

where F is the applied force and A is the cross-sectional area. When a body is stretched by F, the resulting stresses are called *tensile stresses*, and when compressed, they are called *compressive stresses*.

Strain:
Strain is a measure of deformation of the body due to the action of force. In another word it tells how much the body is stretched or compressed compared to its original size. It is denoted by ϵ and defined as:

$$\epsilon = \frac{\Delta L}{L},$$

with ΔL being the change in length and L the original length.

Elasticity :
Elasticity is the property of a material that allows it to return to its original shape and size after removing the external force.

Young's Modulus:
Young's Modulus or Modulus of Elasticity (E) is the measure of stiffness of a material. It is defined as the ratio of stress to strain in the elastic region of material's deformation .

$$E = \frac{\sigma}{\epsilon}.$$

Yield stress: Yield stress is the minimum stress at which the material begins to deform plastically, meaning it will not return to its original shape even if the load is removed.

The definitions in this subsection are taken from [4], [7] and [8].

3.3 Axial Force, Shear Force, and Bending Moment

Axial Force:
Axial force is the tensile or compressive force acting along the longitudinal axis of a structural member.

Shear Force:
Shear force is the force acting perpendicular to the longitudinal axis of a beam or structure, causing it to shear.

Point Load:
A point load (also known as concentrated load) is a force applied at a single point on a beam. It is measured in Newtons(N) or kilonewtons(kN)

Uniformly Distributed Load (UDL):
A uniformly distributed load is a force that is spread constantly along the axis of the beam. It is measured as force per unit length, in N/m or kN/m.

Bending Moment:
The bending moment (BM) is a measure of the tendency of a beam to bend due to applied forces. It is calculated as the product of the applied force and the perpendicular distance from the force to the point of interest.
The definitions in this subsection are taken from [7].

Flexure Formula

The flexure formula is used to calculate the bending stress in a beam subjected to a bending moment. It gives the relationship between the internal stress at a point in a beam to the applied moment, the moment of inertia of cross –section and the distance from neutral axis [7]. It is used in beam design to ensure that the bending stress does not exceed the material's strength.

$$\sigma = -\frac{My}{I},$$

where:

- M is the bending moment at the section,
- y is the distance from the neutral axis,
- I is the moment of inertia of the cross-section.

The maximum bending stress is given by:

$$\sigma_{\max} = \frac{M_{\max} y}{I}.$$

3.4 Moment of Inertia (MOI)

Moment of inertia also known as second moment of area is a geometric property that measures how an object's cross-sectional area is distributed relative to a specific axis [9]. It determines the resistance of a structural member (beam, column) to bending. Mathematically, MOI of a plane area with respect to the x and y axes are given by:

$$I_x = \int y^2 \, dA, \quad I_y = \int x^2 \, dA,$$

where x and y are the coordinates of the differential area element dA.

For a solid rectangular area with width b and height h:

$$I_x = \frac{bh^3}{12}, \quad I_y = \frac{hb^3}{12}.$$

For a solid circular area with radius r:

$$I_x = I_y = \frac{\pi r^4}{4}.$$

For an annulus with an outer radius r_o and an inner radius r_i:

$$I_x = I_y = \frac{\pi}{4}\left(r_o^4 - r_i^4\right).$$

3.5 Deflection of a Beam

Elastic Curve Equation:

The deflection δ of a loaded beam is described by its elastic curve [4]:

$$\frac{d^2\delta}{dx^2} = \frac{M(x)}{EI},$$

where:

- E is the modulus of elasticity,
- I is the moment of inertia,
- $M(x)$ is the bending moment at position x.

Maximum Deflection Due to a Uniformly Distributed Load (UDL)

Consider a simply supported beam AB of length L loaded with a uniformly distributed load of intensity P_0 (N/m). The reaction forces at the supports are:

$$R_A = R_B = \frac{P_0 L}{2}.$$

The bending moment at a distance x from the left support is:

$$M(x) = \frac{P_0 L}{2} x - \frac{P_0 x^2}{2}.$$

Thus, the elastic curve equation becomes:

$$EI \frac{d^2\delta}{dx^2} = \frac{P_0 L}{2} x - \frac{P_0 x^2}{2}.$$

$$\Rightarrow \frac{d^2\delta}{dx^2} = \frac{P_0}{2EI} \left(Lx - x^2 \right).$$

Then we integrate to find the deflection
First Integration (Slope):

$$\frac{d\delta}{dx} = \int \frac{P_0}{2EI} \left(Lx - x^2 \right) dx = \frac{P_0 L}{4EI} x^2 - \frac{P_0}{6EI} x^3 + C_1.$$

Second Integration (Deflection):

$$\delta(x) = \int \left(\frac{P_0 L}{4EI} x^2 - \frac{P_0}{6EI} x^3 + C_1 \right) dx = \frac{P_0 L}{12EI} x^3 - \frac{P_0}{24EI} x^4 + C_1 x + C_2.$$

Applying the boundary conditions for a simply supported beam:

$$\delta(0) = 0 \quad \Rightarrow \quad C_2 = 0,$$

$$\delta(L) = 0 \quad \Rightarrow \quad \frac{P_0 L^4}{12EI} - \frac{P_0 L^4}{24EI} + C_1 L = 0.$$

Solving for C_1:

$$C_1 = -\frac{P_0 L^3}{24EI}.$$

By symmetry, the maximum deflection occurs at midspan, $x = \frac{L}{2}$, and is given by:

$$\delta_{\text{UDL,max}} = \frac{5 P_0 L^4}{384 EI}.$$

Maximum Deflection Due to a Central Point Load

Again consider the same beam AB with a point load P applied at midspan, the reactions are:

$$R_A = R_B = \frac{P}{2}.$$

For $0 \leq x \leq \frac{L}{2}$, the bending moment is:

$$M(x) = \frac{P}{2}x.$$

The elastic curve equation is:

$$EI\frac{d^2\delta}{dx^2} = \frac{P}{2}x.$$

First Integration (Slope):

$$\frac{d\delta}{dx} = \int \frac{P}{2EI}x\,dx = \frac{P}{4EI}x^2 + C_1.$$

By symmetry (zero slope at midspan $x = \frac{L}{2}$):

$$C_1 = -\frac{PL^2}{16EI}.$$

Second Integration (Deflection):

$$\delta(x) = \int \left(\frac{P}{4EI}x^2 - \frac{PL^2}{16EI}\right)dx = \frac{P}{12EI}x^3 - \frac{PL^2}{16EI}x + C_2.$$

With $\delta(0) = 0$, we have $C_2 = 0$. Evaluating at midspan ($x = \frac{L}{2}$):

$$\delta\left(\frac{L}{2}\right) = \frac{P}{12EI}\left(\frac{L}{2}\right)^3 - \frac{PL^2}{16EI}\left(\frac{L}{2}\right) = \frac{PL^3}{96EI} - \frac{PL^3}{32EI} = -\frac{PL^3}{48EI}.$$

Therefore maximum deflection due to point load is,

$$\delta_{\text{PL,max}} = \frac{PL^3}{48EI}.$$

3.6 Buckling and Critical Load

Buckling is the failure of structural member (e.g. column) by lateral deflection (bending sideways) when it is subject to high compressive loads.

The maximum axial compressive load that a structure (such as a column) can support without buckling is called the **critical load**. When the applied load reaches this value, the structure becomes unstable and begins to deflect laterally, even if the material has not yet yielded.

Euler's Formula for Critical Load:
For an ideal, slender, elastic column with constant cross-section and perfectly aligned axial loads, Euler's formula for a pinned-pinned (simply supported) column is:

$$P_{cr} = \frac{\pi^2 EI}{L^2},$$

where E is the Young's modulus, I is the moment of inertia, and L is the column length.

For other boundary conditions, an effective length factor K is introduced:

$$P_{cr} = \frac{\pi^2 EI}{(KL)^2}.$$

Typical values of K are:

- Pinned-Pinned: $K = 1$,
- Fixed-Fixed: $K \approx 0.5$,
- Fixed-Free (Cantilever): $K = 2$,
- Fixed-Pinned: $K \approx 0.7$.

The definitions and formulas in this subsection are taken from [4], [7].

3.7 Natural Frequency of Vibration

Natural frequency is the rate at which a system oscillates freely after an initial disturbance, without any external forces acting on it [15].The natural frequency f is given by:

$$f = \frac{1}{2\pi}\sqrt{\frac{k}{M_{eff}}},$$

For a cantilever beam of length l, mass m with a concentrated end mass M, the stiffness k and effective mass M_{eff} are given as:

$$\begin{aligned} k &= \frac{3EI}{l^3}, \\ M_{\text{eff}} &= M + m_{\text{eq}}, \\ m_{\text{eq}} &= \frac{33}{140}m, \end{aligned}$$

where$E$ is the modulus of elasticity, I is the moment of inertia, and m_{eq} is the "modal" or equivalent mass of the beam in its first bending mode. Substituting into the frequency formula gives.

$$f = \frac{1}{2\pi}\sqrt{\frac{3EI}{M + \frac{33}{140}m}}$$

4 Implementation tools: Python and MATLAB

I used Python instead of other optimization tools because of its free and rich open-source libraries. Python is a popular language for optimization problems due to its simplicity and the wide variety of libraries available. SciPy, PuLP, Pyomo, and pymoo are some of the well-known libraries used for optimization. In this work, I primarily used SciPy and pymoo to implement my solutions.

scipy.optimize is a module in SciPy (a scientific computing library in Python) that provides various optimization algorithms for solving different types of mathematical optimization problems, including linear programming, nonlinear programming, and constrained optimization.

scipy.optimize.linprog is used for solving linear programming (LP) problems, where both the objective function and constraints are linear. The HiGHS solver in scipy.optimize.linprog provides high-performance methods for solving LP and mixed-integer programming (MIP) problems. It uses the following algorithms to solve linear problems:

- Simplex Method
- Dual Simplex Method
- Interior Point Method

scipy.optimize.minimize is used to solve nonlinear programming (NLP) problems and provides the SLSQP algorithm for smooth nonlinear cases. It efficiently handles both equality and inequality constraints using a successive quadratic approximation approach.

For the multi-objective optimization problem, the Python library pymoo is used along with the NSGA-II algorithm. pymoo is a versatile library for evolutionary and multi-objective optimization, offering algorithms such as Genetic Algorithm (GA) and NSGA-II. NSGA-II features fast non-dominated sorting, explicit elitism, and crowding-distance based diversity control to efficiently approximate the Pareto front in multi-objective design problems.

MATLAB is used to plot the graphical solutions for my problems. MATLAB (Matrix Laboratory) is a high-level programming language and environment developed by MathWorks, primarily used for numerical computing, data visualization, and algorithm development. It was chosen for graphical visualization in this work due to its powerful built-in plotting functions, interactive tools, and support for high-quality 2D and 3D graphics.

5 Solved Problems

5.1 Problem 1: Scaffolding system

A scaffolding system consists of four beams and eight ropes, as shown in Figure 5.1. Each of the ropes A and B can carry a load of W_1, each of the ropes C and D can carry a load of W_2, each of the middle ropes E and F can carry a load of W_3, and each of the bottom ropes G and H can carry a load of W_4. Beam 1 is 4 m in length and all other beams are 5 m in length. There is a 1 m gap between ropes B and C. The loads acting on beams 1, 2, 3, and 4 are x_1, x_2, x_3, and x_4, respectively, and the weights of beams and ropes are negligible. The goal is to formulate the problem of finding the maximum load, i.e., maximizing $(x_1 + x_2 + x_3 + x_4)$, that can be supported by the system considering that each beams must support at least 50 N.

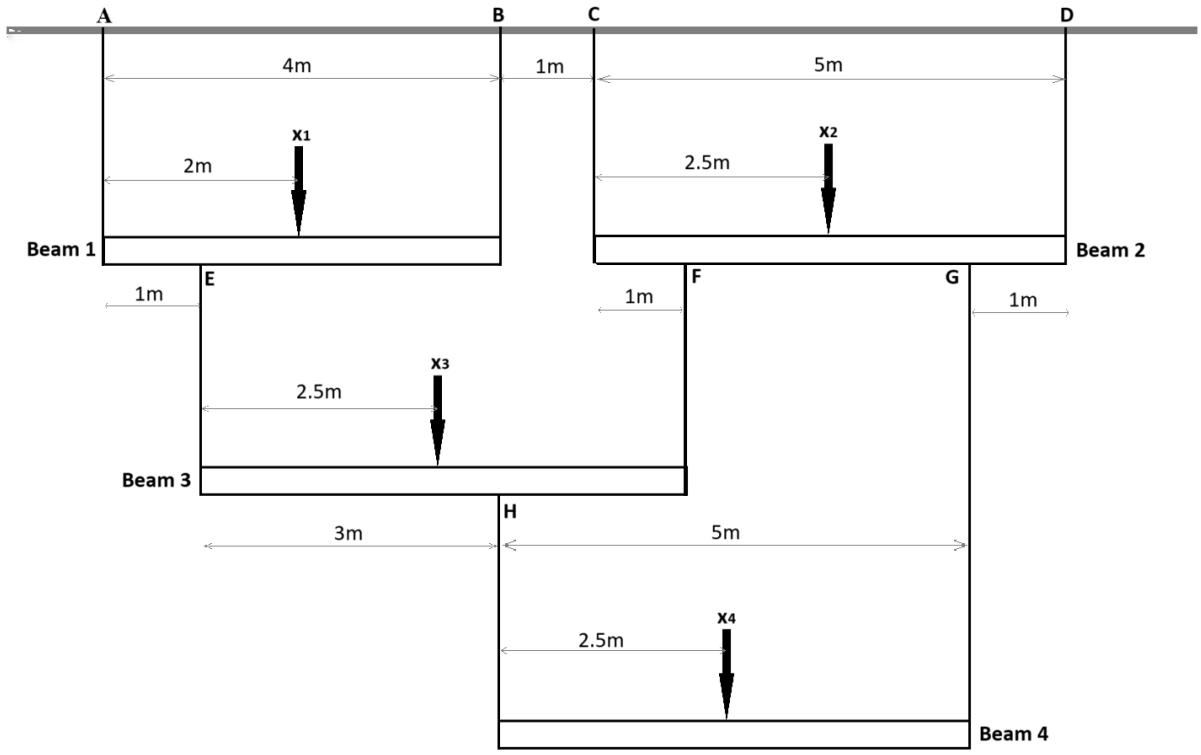

Figure 5.1: Scaffolding system with four beams

Equations of Equilibrium

The equations of equilibrium for vertical forces and moments for each beam are as follows:

Beam 4

$$T_G + T_H = x_4,$$
$$\frac{5}{2}x_4 - 5T_H = 0.$$

Beam 3

$$T_E + T_F - T_H = x_3,$$
$$\frac{5}{2}x_3 + 3T_H - 5T_F = 0.$$

Beam 2

$$T_C + T_D - T_F - T_G = x_2,$$
$$\frac{5}{2}x_2 + 1T_F + 4T_G - 5T_D = 0.$$

Beam 1

$$T_A + T_B - T_E = x_1,$$
$$2x_1 + 1T_E - 4T_B = 0.$$

where T_i denotes the tension in rope i, solving these equations yields:

$$\begin{aligned}
T_H &= \frac{1}{2}x_4,\\
T_G &= \frac{1}{2}x_4,\\
T_F &= \frac{1}{2}x_3 + \frac{3}{10}x_4,\\
T_E &= \frac{1}{2}x_3 + \frac{1}{5}x_4,\\
T_D &= \frac{1}{2}x_2 + \frac{1}{10}x_3 + \frac{23}{50}x_4,\\
T_C &= \frac{1}{2}x_2 + \frac{2}{5}x_3 + \frac{17}{50}x_4,\\
T_B &= \frac{1}{2}x_1 + \frac{1}{8}x_3 + \frac{1}{20}x_4,\\
T_A &= \frac{1}{2}x_1 + \frac{3}{8}x_3 + \frac{3}{20}x_4.
\end{aligned}$$

Optimization Problem Formulation

Design vector and objective function
Choose the design vector as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

The objective is to maximize the total supported load:

$$\text{maximize } (x_1 + x_2 + x_3 + x_4).$$

For standard linear programming formulation (minimization), this can be written as:

$$\text{minimize } -(x_1 + x_2 + x_3 + x_4).$$

Constraints
The system must satisfy the following constraints based on the capacities of the ropes:

$$\begin{aligned} T_A &\leq W_1, \\ T_B &\leq W_1, \\ T_C &\leq W_2, \\ T_D &\leq W_2, \\ T_E &\leq W_3, \\ T_F &\leq W_3, \\ T_G &\leq W_4, \\ T_H &\leq W_4. \end{aligned}$$

Additionally, the constraints due to the given lower bound can be written as:

$$x_1 \geq 50, \quad x_2 \geq 50, \quad x_3 \geq 50, \quad x_4 \geq 50.$$

Since all equations and inequalities are linear, the problem is a linear programming problem.

Problem Data and Numerical Solution

Assume all the ropes can withstand the same amount of tension of 100 N.

$$W_1 = W_2 = W_3 = W_4 = 100 \text{ N}$$

The optimal solution is:

$$\begin{aligned} x_1 &= 125.00 \text{ N} \\ x_2 &= 75.00 \text{ N} \\ x_3 &= 50.00 \text{ N} \\ x_4 &= 125.00 \text{ N} \end{aligned}$$

The total load that can be supported by the scaffolding system is,

$$(x_1 + x_2 + x_3 + x_4) = 375.00 \text{ N}$$

The tensions on the ropes (constraint value) at the optimal solution are,

$$\begin{aligned} T_A &= 100.00 \text{ N} \\ T_B &= 75.00 \text{ N} \\ T_C &= 100.00 \text{ N} \\ T_D &= 100.00 \text{ N} \\ T_E &= 50.00 \text{ N} \\ T_F &= 62.50 \text{ N} \\ T_G &= 62.50 \text{ N} \\ T_H &= 62.50 \text{ N} \end{aligned}$$

5.2 Problem 2: Beam design optimization

A simply supported beam with a uniform rectangular cross section is subjected to both distributed and concentrated loads, as shown in the Figure 5.2. It is desired to find the cross section of the beam to minimize the weight of the beam while ensuring that the maximum stress induced in the beam does not exceed the permissible stress (σ_0) of the material and the maximum deflection of the beam does not exceed a specified limit (δ_0). Assume that the cross-sectional dimensions of the beam are restricted as $x_1 \leq x_2$, $0.04 \text{ m} \leq x_1 \leq 0.12 \text{ m}$, and $0.06 \text{ m} \leq x_2 \leq 0.20 \text{ m}$. This problem is taken from the exercise in [14].

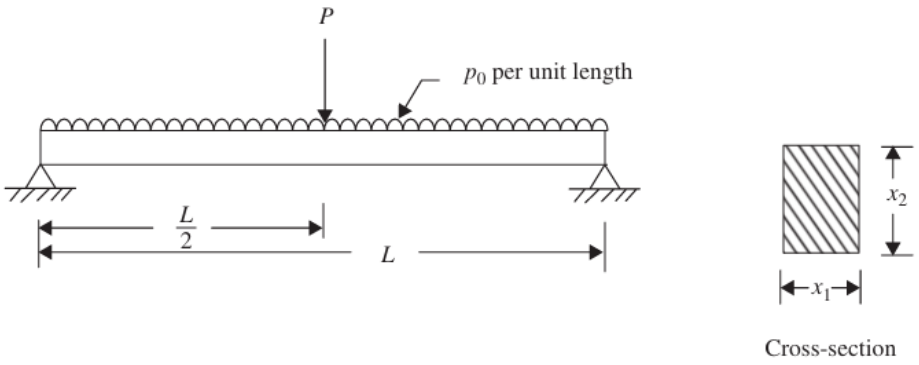

Figure 5.2: A simply supported beam subjected to concentrated and distributed load

The data of the problem are:

$$E = 207 \cdot 10^9 \text{ Pa}, \quad L = 1\text{m}, \quad P = 10^5\text{N}, \quad P_0 = 10^6 \text{ N/m}$$

$$\text{Weight density } = 76.5 \cdot 10^3 \text{ N/m}^3$$

$$\sigma_0 = 220 \text{ MPa}, \quad \delta_0 = 0.02\text{m}$$

Objective Function

The weight of the beam:

$$W = \rho_w V$$

$$W = \rho_w A L$$

$$W = \rho_w x_1 x_2 L$$

Since L and $\rho_w$is fixed, the objective function is:

$$\text{Minimize } W = 76.5 \cdot 10^3 x_1 x_2$$

Constraints

1. Stress constraint
The stress at any point is given by the flexure formula:

$$\sigma = \frac{My}{I}$$

where M is the bending moment, I is the moment of inertia, and y is the distance from the neutral axis. For a rectangular cross section:

$$I = \frac{x_1 x_2^3}{12}, \quad y = \frac{x_2}{2}$$

Maximum bending moment:

$$M_{\max} = \frac{PL}{4} + \frac{P_0 L^2}{8}$$

$$\sigma_{\max} = \frac{6M_{\max}}{x_1 x_2^2}$$

The maximum stress should not exceed the permissible stress:

$$\sigma_{\max} \leq \sigma_0$$

Substituting values:

$$\frac{(6PL + 3P_0L^2)}{4x_1x_2^2} \leq 220 \cdot 10^6$$

$$\frac{9 \cdot 10^5}{x_1x_2^2} \leq 220 \cdot 10^6$$

$$\frac{1}{x_1x_2^2} \leq \frac{2200}{9}$$

2. Deflection constraint

Using the equation of the elastic curve, we have

$$\frac{d^2\delta}{dx^2} = \frac{M(x)}{EI}$$

where,

E = Modulus of elasticity of material,
$M(x)$ = Bending moment of section at x,
δ = deflection at distance x,
I = Moment of inertia of the beam cross section

Deflection due to distributed load:

$$\delta_{udl,\max} = \frac{5P_0L^4}{384EI}$$

Deflection due to point load:

$$\delta_{pl,\max} = \frac{PL^3}{48EI}$$

Total maximum deflection should not exceed the allowable deflection δ_0:

$$\delta_{\max} = \delta_{udl,\max} + \delta_{pl,\max} \leq \delta_0$$

Substituting $I = \frac{x_1x_2^3}{12}$:

$$\delta_{\max} = \frac{5P_0L^4}{384E(\frac{x_1x_2^3}{12})} + \frac{PL^3}{48E(\frac{x_1x_2^3}{12})}$$

$$\frac{1}{4Ex_1x_2^3}\left(\frac{5P_0L^4}{8} + PL^3\right) \leq 0.02$$

$$\frac{1}{4 \cdot 207 \cdot 10^9 x_1 x_2^3}\left(\frac{5 \cdot 10^6 \cdot 1^4}{8} + 10^5 \cdot 1^3\right) \leq 0.02$$

$$\frac{1}{x_1 x_2^3} \leq \frac{1656 \cdot 10^4}{725}$$

3. Geometric constraint

$$x_1 \leq x_2,$$

$$0.04\ \text{m} \leq x_1 \leq 0.12\ \text{m}, \quad \text{and} \quad 0.06\ \text{m} \leq x_2 \leq 0.20\ \text{m}.$$

Final Formulation

$$\text{minimize } W = 76.5 \cdot 10^3 x_1 x_2$$

$$\text{subject to: } x_1 x_2^2 \geq \frac{9}{2200},$$

$$x_1 x_2^3 \geq \frac{725}{1656 \cdot 10^4},$$

$$x_2 \geq x_1,$$

$$0.04\ \text{m} \leq x_1 \leq 0.12\ \text{m}, \quad \text{and} \quad 0.06\ \text{m} \leq x_2 \leq 0.20\ \text{m}.$$

Graphical Solution

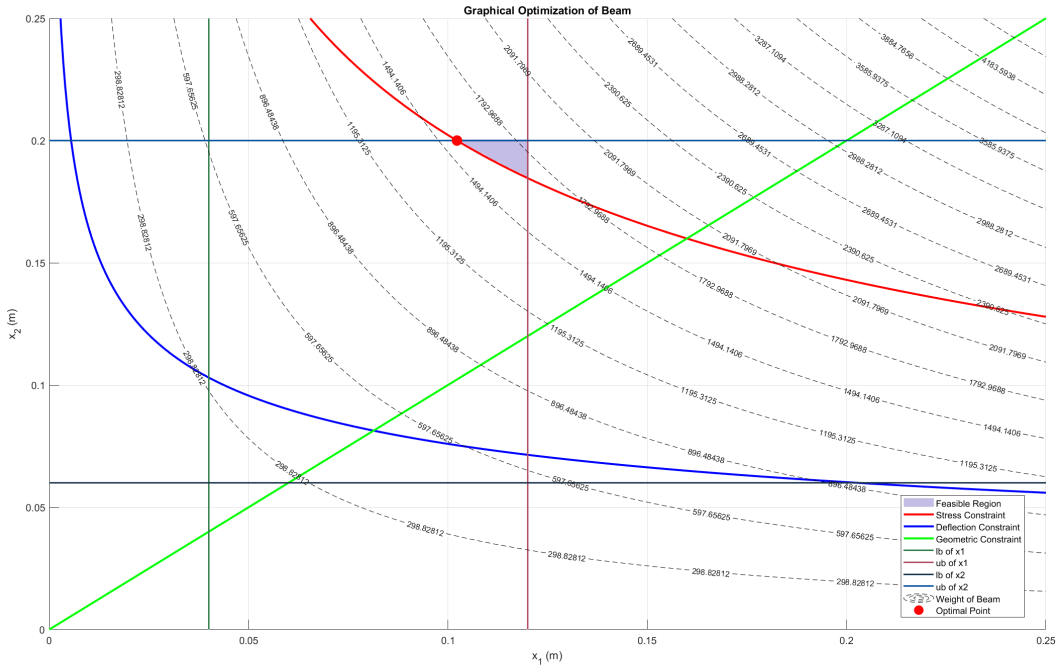

Figure 5.3: Graphical solution of beam optimization problem

Numerical Solution

The exact values of the optimal solutions are:

$$x_1 = 102.2727 \text{ mm}$$

$$x_2 = 200.0000 \text{ mm},$$

And the weight of the beam is,

$$W = 1564.7727 \text{ N}$$

The numerical solution corresponds with the graphical solution.

Verifying Using Optimality Conditions

The objective function is

$$f(\mathbf{x}) = W = 76.5 \cdot 10^3 x_1 x_2,$$

and the constraints are

$$\begin{aligned}
g_1(\mathbf{x}) &= \frac{9}{2200} - x_1 x_2^2 \leq 0, \\
g_2(\mathbf{x}) &= \frac{725}{1656 \cdot 10^4} - x_1 x_2^3 \leq 0, \\
g_3(\mathbf{x}) &= x_1 - x_2 \leq 0, \\
g_4(\mathbf{x}) &= 0.04 - x_1 \leq 0, \\
g_5(\mathbf{x}) &= x_1 - 0.12 \leq 0, \\
g_6(\mathbf{x}) &= 0.06 - x_2 \leq 0, \\
g_7(\mathbf{x}) &= x_2 - 0.20 \leq 0.
\end{aligned}$$

Convexity Check

The Hessian of the objective function is

$$\nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} 0 & 76.5 \cdot 10^3 \\ 76.5 \cdot 10^3 & 0 \end{pmatrix}.$$

The eigenvalues of this matrix are $\pm 76.5 \cdot 10^3$, so it is indefinite. Hence, f is not convex. Each nonlinear constraint,

$$g_1(\mathbf{x}) = \tfrac{9}{2200} - x_1 x_2^2, \quad g_2(\mathbf{x}) = \tfrac{725}{1656 \cdot 10^4} - x_1 x_2^3,$$

also has an indefinite Hessian. Therefore the problem is non-convex and global optimality cannot be inferred from convexity.

Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{7} \lambda_i \, g_i(\mathbf{x}), \quad \lambda_i \geq 0$$

KKT Conditions at the candidate point

1. **Feasibility**: At $\mathbf{x}^* = (\frac{9}{88}, \frac{1}{5})$

$$g_1(\mathbf{x}^*) = \frac{9}{2200} - \frac{9}{88}\left(\frac{1}{5}\right)^2 = \frac{9}{2200} - \frac{9}{2200} = 0 \ \leq 0,$$

$$g_2(\mathbf{x}^*) = \frac{725}{1656 \cdot 10^4} - \frac{9}{88}\left(\frac{1}{5}\right)^3 \approx 4.3768 \cdot 10^{-5} - 8.1818 \cdot 10^{-4} = -7.7441 \cdot 10^{-4} \ \leq 0,$$

$$g_3(\mathbf{x}^*) = \frac{9}{88} - \frac{1}{5} = -0.0977 \ \leq 0,$$

$$g_4(\mathbf{x}^*) = 0.04 - \frac{9}{88} = -0.0623 \ \leq 0,$$

$$g_5(\mathbf{x}^*) = \frac{9}{88} - 0.12 = -0.0177 \ \leq 0,$$

$$g_6(\mathbf{x}^*) = 0.06 - \frac{1}{5} = -0.14 \ \leq 0,$$

$$g_7(\mathbf{x}^*) = \frac{1}{5} - 0.20 = 0 \ \leq 0.$$

   All seven constraints satisfy $g_i(\mathbf{x}^*) \leq 0$, so the point is feasible and the active constraints are

$$g_1(\mathbf{x}^*) = 0, \quad g_7(\mathbf{x}^*) = 0,$$

2. **Complementarity:**

$$\lambda_i g_i(\mathbf{x}^*) = 0, \quad \lambda_i \geq 0, \quad i = 1 \ldots 7$$

   Since, $g_i(\mathbf{x}^*) < 0 \quad$ for $i = 2 \ldots 6$

$$\implies \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = 0$$

   Then the Lagrangian is

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = 76.5 \cdot 10^3 \, x_1 x_2 + \lambda_1 \left(\tfrac{9}{2200} - x_1 x_2^2\right) + \lambda_7 \, (x_2 - 0.20), \quad \lambda_i \geq 0.$$

3. **Stationarity**
   $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{0}$ yields

$$\begin{cases} 76.5 \cdot 10^3 \, x_2 - \lambda_1 \, x_2^2 = 0, \\ 76.5 \cdot 10^3 \, x_1 - 2\lambda_1 \, x_1 x_2 + \lambda_7 = 0, \end{cases}$$

   which at the point $\mathbf{x}^*$ gives $\lambda_1 = 382500$ and $\lambda_7 \approx 7824$, both nonnegative.

4. **Linear Independence CQ**
   The active gradients

$$\nabla g_1(\mathbf{x}^*) = \begin{bmatrix} -x_2^2 \\ -2x_1 x_2 \end{bmatrix} \Big|_{(\frac{9}{88}, \frac{1}{5})} = \begin{bmatrix} -\frac{1}{25} \\ -\frac{9}{220} \end{bmatrix}, \quad \nabla g_7(\mathbf{x}^*) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

   are linearly independent. Hence, all KKT conditions are satisfied leading to the KKT point $(\mathbf{x}^*, \boldsymbol{\lambda})$.

Second-Order Sufficient Condition (SOSC)

Index sets of active constraints

$$I_g((\mathbf{x}^*)) = \{1, 7\}, \qquad I_g^+((\mathbf{x}^*)) = \{1, 7\}, \qquad I_g^0((\mathbf{x}^*)) = \varnothing.$$

For nonzero direction $\mathbf{z} \in \mathbb{R}^2$

$$\mathbf{z}^T \nabla g_i((\mathbf{x}^*)) = 0 \quad \forall\, i \in I_g^+((\mathbf{x}^*)) \quad \Longrightarrow \quad \begin{cases} \mathbf{z}^T \nabla g_1((\mathbf{x}^*)) = 0, \\ \mathbf{z}^T \nabla g_7((\mathbf{x}^*)) = 0. \end{cases}$$

$$\Longrightarrow \begin{cases} -\frac{1}{25} z_1 - \frac{9}{220} z_2 = 0, \\ z_2 = 0. \end{cases} \quad \Longrightarrow \begin{cases} z_1 = 0, \\ z_2 = 0. \end{cases}$$

Since no $\mathbf{z} \neq \mathbf{0}$ exists, the SOSC holds vacuously, and $\mathbf{x}^*$ is a strict local minimum. (It is not necessary to compute $\nabla^2_{xx}\mathcal{L}$ to check the SOSC.)

5.3 Problem 3: Truss design optimization

A two-bar truss is to be designed to carry $2W$ as shown in Figure 5.4. Both bars have a tubular section of mean diameter d and wall thickness t. The material of the bar has Young's modulus E and yield stress σ_y. The design problem involves the determination of the values of d and t so that the weight of the truss is minimum and neither yielding nor buckling occurs on any of the bars. This problem is taken from the exercise in [14].

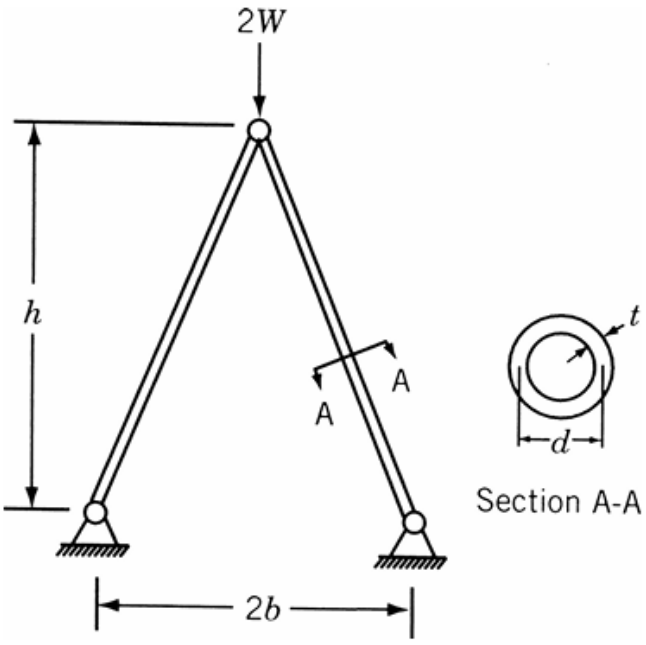

Figure 5.4: Two bar truss

Objective Function

- Cross-sectional area of the tubular section:

$$A = \pi dt$$

- Total weight of the bars:

$$W_{\text{total}} = 2A\rho L$$

  where $L = \sqrt{h^2 + b^2}$ is the length of one bar.
- Thus, the objective is to minimize:

$$W_{\text{total}} = 2\pi dt\rho\sqrt{h^2 + b^2}$$

Constraints

1. **Yield Stress Constraint**:
   - Axial stress must not exceed the yield stress σ_y:

$$\sigma = \frac{F}{A} \leq \sigma_y$$

   - Force equilibrium (resolving vertical forces):

$$2F\sin\theta = 2W \quad \text{where} \quad \sin\theta = \frac{h}{\sqrt{h^2 + b^2}}$$

   - Thus, the yield stress constraint is:

$$\frac{W\sqrt{h^2 + b^2}}{\pi dth} \leq \sigma_y$$

2. **Buckling Constraint** :
   - Euler's formula for critical buckling load:

$$P_{\text{cr}} = \frac{\pi^2 EI}{(KL)^2}$$

     where,

$$\begin{aligned} E &= \text{Modulus of elasticity of material,} \\ K &= \text{Effective length factor} \quad (K = 1 \text{ (pinned ends)}), \\ L &= \text{Length of the tube} \quad \left(L = \sqrt{h^2 + b^2}\right), \\ I &= \text{Moment of inertia of tube cross section} \end{aligned}$$

$$I = \frac{\pi}{64}\left[(d+t)^4 - (d-t)^4\right] = \frac{\pi}{8}\left(d^3t + dt^3\right)$$

   - Thus, critical buckling load is:

$$P_{\text{cr}} = \frac{\pi^3 E\left(d^3t + dt^3\right)}{8\left(h^2 + b^2\right)}$$

   - For Truss to avoid buckling:

$$F \leq P_{\text{cr}}$$

$$\frac{W\sqrt{h^2 + b^2}}{h} \leq \frac{\pi^3 E\left(d^3t + dt^3\right)}{8\left(h^2 + b^2\right)}$$

3. **Geometric Constraints**:
   - Practical limits for d and t: For ensuring valid tubular section

$$t \leq \frac{d+t}{2} \quad \text{and} \quad t, d > 0$$

Final Formulation

$$\text{minimize } W_{total} = (2\pi\rho\sqrt{h^2 + b^2})dt$$

$$\text{subject to: } dt \geq \frac{W\sqrt{h^2 + b^2}}{\pi h \sigma_y},$$

$$(d^3t + dt^3) \geq \frac{8W(h^2 + b^2)^{3/2}}{\pi^3 E h},$$

$$d \geq t, \quad t > 0, \quad d > 0.$$

Problem Data

Assume the truss is made with structural steel with the following data:
Young's modulus $E = 200 \cdot 10^9$ Pa
Yield stress $\sigma_y = 310 \cdot 10^6$ Pa
Weight density $\rho = 77 \cdot 10^3 \text{N/m}^3$
Half base length b = 0.5 m
Height h = 1 m
Load W = 25000 N
Bounds:

$$0.02 \text{ m} \leq d \leq 0.05 \text{ m}, \quad \text{and} \quad 0.005 \text{ m} \leq t \leq 0.05 \text{ m}.$$

Graphical Solution

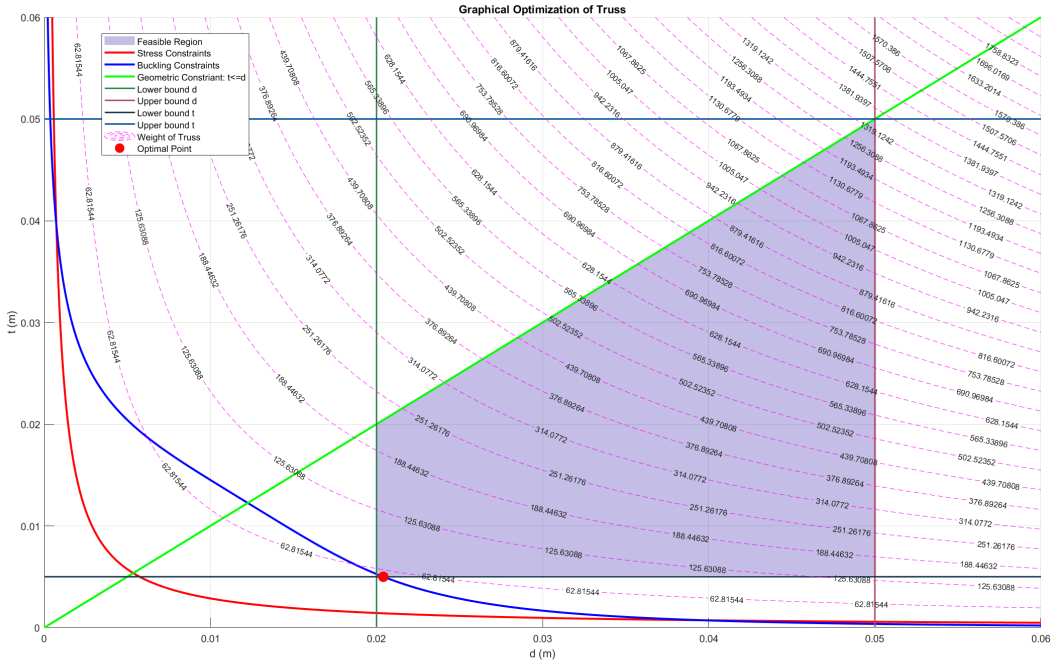

Figure 5.5: Graphical solution of truss optimization problem

Numerical Solution

The exact values of the optimal solutions are:

$$d = 20.4145 \text{ mm}$$

$$t = 5.0000 \text{ mm},$$

And the weight of the beam is,

$$W_{total} = 55.2120 \text{ N}$$

The numerical solution corresponds with the graphical solution.

5.4 Problem 4: Water tank column design optimization

A uniform column of rectangular cross-section is to be constructed for supporting a water tank of mass M (see Figure 5.6). The objectives are:

1. Minimize the mass of the column for economy.
2. Maximize the natural frequency of transverse vibration of the system to avoid possible resonance due to wind.

In addition, the column must be designed to avoid failure due to direct compression and buckling. Assume the permissible compressive stress to be $\sigma_{\max}$. This problem is taken from the exercise in [14].

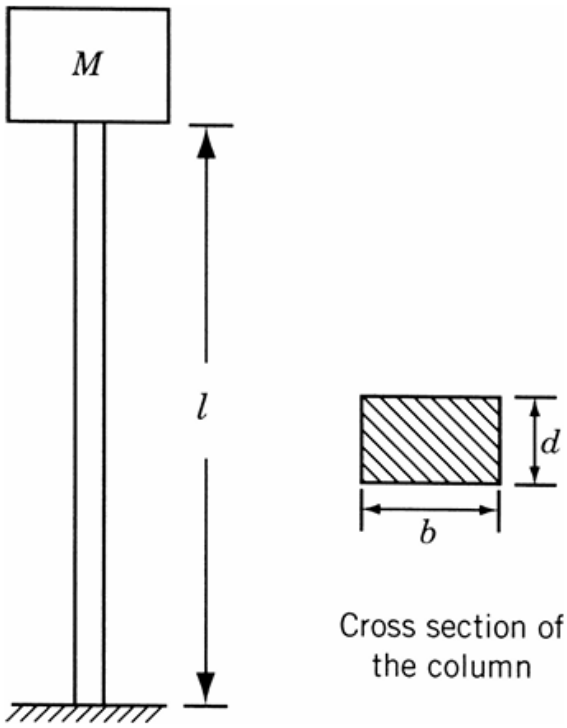

Figure 5.6: Water tank column

Model Equations

Let $x_1 = b$ and $x_2 = d$ denote the cross-sectional dimensions of the column. The mass of the column (m) is given by:

$$m = \rho\, bd\, l = \rho\, l\, x_1 x_2,$$

where ρ is the density and l is the height of the column.

The natural frequency of transverse vibration of the water tank (f), modeled as a cantilever beam with a tip mass M, is:

$$f = \frac{1}{2\pi}\left[\frac{3EI}{\left(M + \frac{33}{140}m\right) l^3}\right]^{1/2},$$

where E is the Young's modulus and I is the area moment of inertia of the column. For a rectangular cross-section, the moment of inertia is:

$$I = \frac{1}{12}\, bd^3 = \frac{1}{12}\, x_1 x_2^3.$$

Substituting $m = \rho\, l\, x_1 x_2$ into the expression for f, we rewrite it as:

$$f = \frac{1}{2\pi}\left[\frac{E\, x_1 x_2^3}{4l^3\left(M + \frac{33}{140}\rho\, l\, x_1 x_2\right)}\right]^{1/2}.$$

The direct compressive stress in the column due to the water tank is:

$$\sigma_c = \frac{Mg}{bd} = \frac{Mg}{x_1 x_2}.$$

The buckling stress for a fixed-free column is given by:

$$\sigma_b = \left(\frac{\pi^2 EI}{4l^2}\right)\frac{1}{bd} = \frac{\pi^2 E x_2^2}{48l^2}.$$

Optimization Problem Formulation

Design Variables and Objective Functions

Define the design vector as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The multi-objective problem is to:

- Minimize the mass of the column:

$$f_1(\mathbf{x}) = \rho\, l\, x_1 x_2.$$

- Maximize the natural frequency of the water tank. For optimization purposes, we consider minimizing the negative natural frequency:

$$f_2(\mathbf{x}) = -\frac{1}{2\pi}\left[\frac{E\, x_1 x_2^3}{4l^3\left(M + \frac{33}{140}\rho\, l\, x_1 x_2\right)}\right]^{1/2}.$$

Constraints
The design must satisfy the following constraints:

- The direct compressive stress must not exceed the permissible stress:

$$g_1(\mathbf{x}) = \frac{Mg}{x_1 x_2} - \sigma_{\max} \leq 0.$$

- To avoid buckling, the buckling stress must be at least as high as the direct compressive stress:

$$g_2(\mathbf{x}) = \frac{Mg}{x_1 x_2} - \frac{\pi^2 E x_2^2}{48 l^2} \leq 0.$$

- The design variables must be positive:

$$g_3(\mathbf{x}) = -x_1 \leq 0, \quad g_4(\mathbf{x}) = -x_2 \leq 0.$$

Weighted Approach Formulation:

Let W_{min} be the weight of the column when we consider only the weight minimization objective and f_{max} be the natural frequency of vibration when we consider only the frequency maximization objective. Let $\alpha \in [0,1]$ be the trade-off parameter between the two objectives. The normalized weighted objective can be formulated as:

$$f(\mathbf{x}) = \frac{\alpha}{W_{min}} f_1(\mathbf{x}) + \frac{1-\alpha}{f_{max}} f_2(\mathbf{x})$$

$$f(\mathbf{x}) = \frac{\alpha}{W_{min}} \rho\, l\, x_1 x_2 - \frac{1-\alpha}{f_{max}} \frac{1}{2\pi} \left[\frac{E\, x_1 x_2^3}{4l^3 \left(M + \frac{33}{140} \rho\, l\, x_1 x_2 \right)} \right]^{1/2}$$

Subject to the same constraints as in the previous formulation.

Problem Data

Assume the column is made of structural steel with the following data:
Density of the material $\rho = 7850$ kg/m^3
Column length $l = 5$ m
Young's modulus $E = 200 \cdot 10^9$ Pa
Mass of the tank $M = 1000$ kg
Acceleration due to gravity $g = 9.81$ m/s^2
Maximum permissible stress $\sigma_{\max} = 250 \cdot 10^6$ Pa
Bounds:

$$0.04 \text{ m} \leq x_1 \leq 0.5 \text{ m}, \quad \text{and} \quad 0.04 \text{ m} \leq x_2 \leq 0.5 \text{ m}.$$

Graphical Solutions

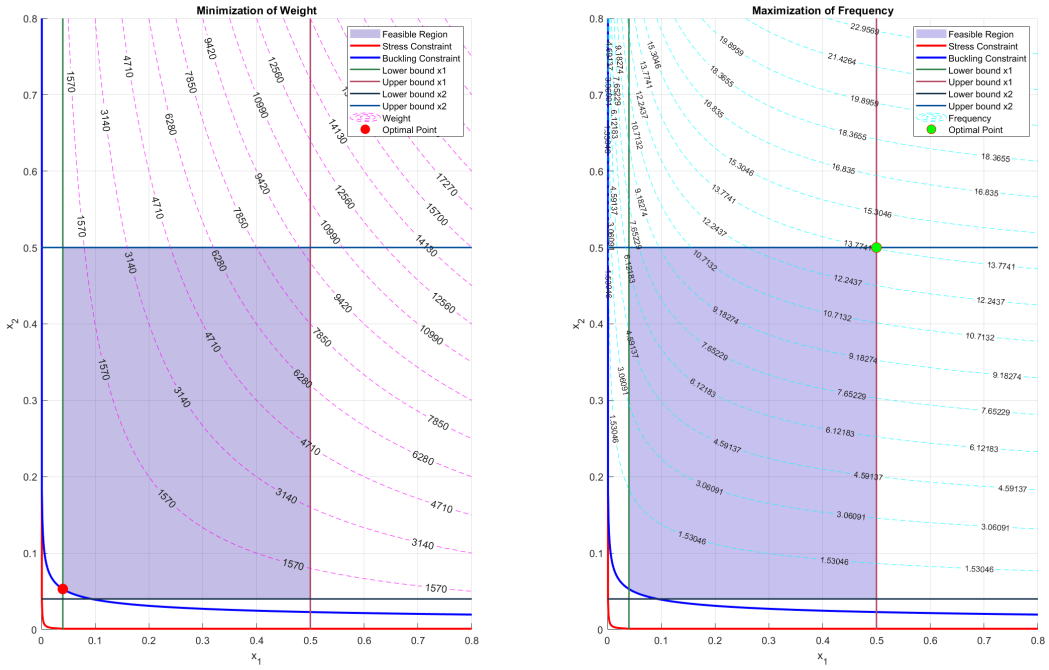

Figure 5.7: SOO of water tank column optimization

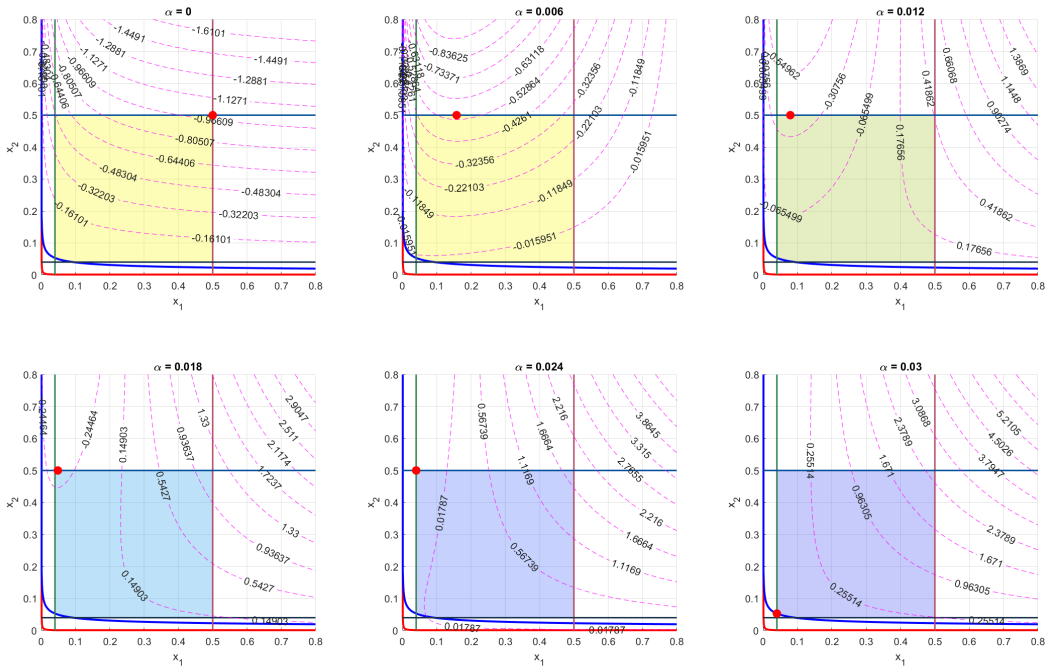

Figure 5.8: Normalized Weighted Optimization for different alpha

Figure 5.7 shows the single-objective-optimization (SOO) plot, from which it is evident that the two objectives are in direct conflict: any improvement in one comes at the expense of the other.

Figures 5.8, 5.9 presents the normalized weighted-sum solutions as the weight parameter α varies:

- At $\alpha = 0$, the optimizer completely favors the second objective, yielding

$$(x_1, x_2) \;=\; (0.50\text{ m},\ 0.50\text{ m}).$$

- As α increases from 0 up to approximately 0.03, the optimal point moves continuously along the line

$$x_2 = 0.50\text{ m},$$

  with x_1 decreasing from 0.50 m down to 0.04 m.
- Near $\alpha \approx 0.03$, the solution undergoes a sudden transition (a "jump") from

$$(0.04\text{ m},\ 0.50\text{ m}) \quad \longrightarrow \quad (0.04\text{ m},\ 0.053\text{ m}).$$

- For all $\alpha > 0.03$, the optimal solution remains fixed at

$$(0.04\text{ m},\ 0.053\text{ m}),$$

  indicating that beyond this threshold the trade-off no longer shifts the solution.

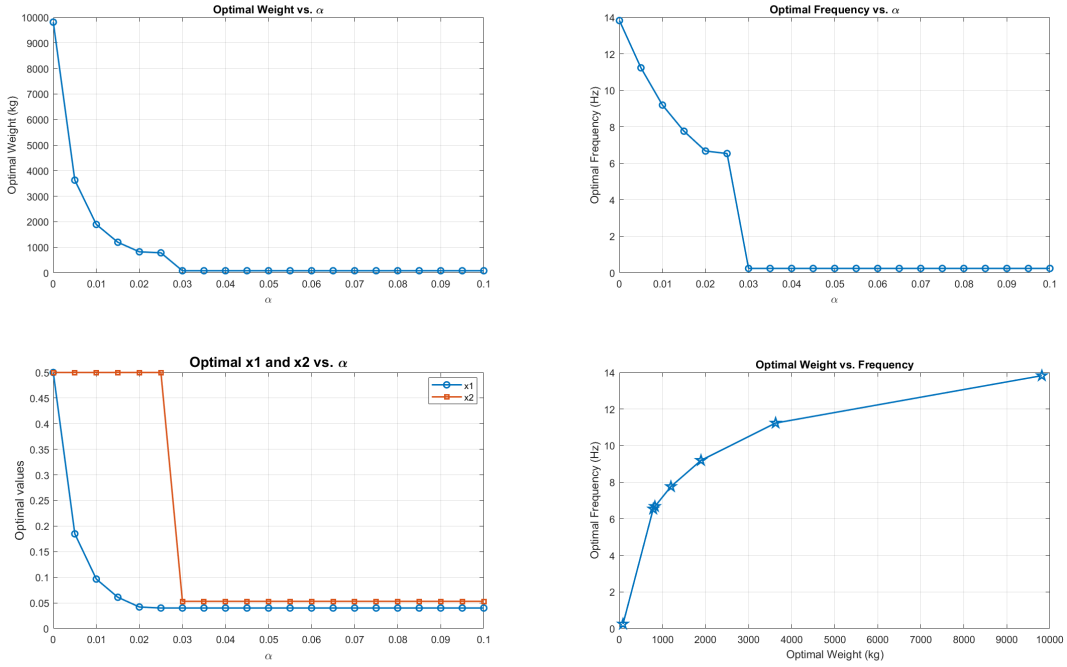

Figure 5.9: MOO of water tank column optimization

Pareto Solution

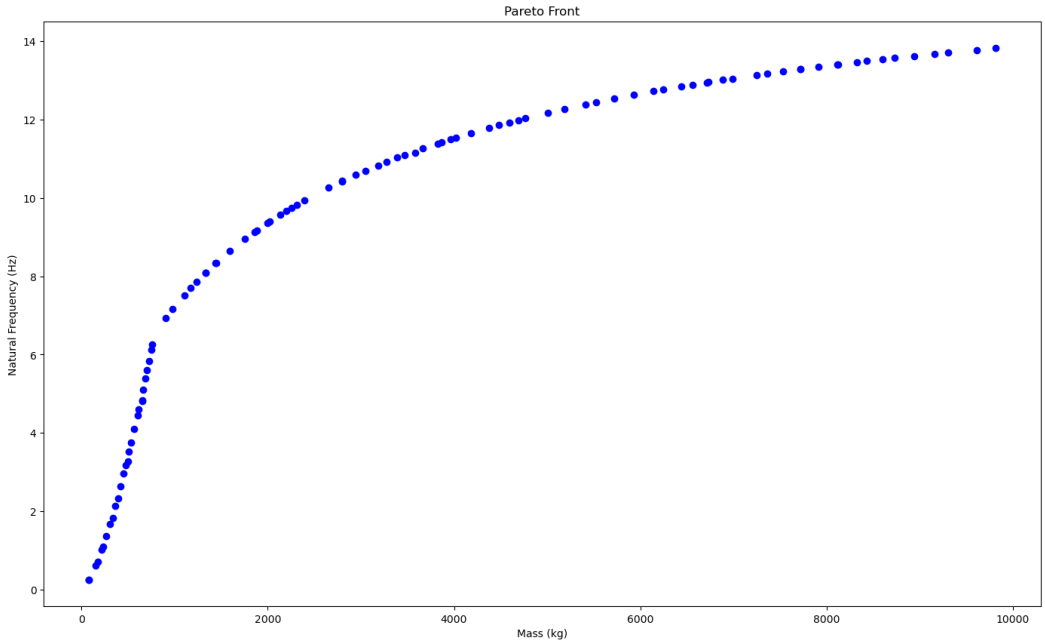

Figure 5.10: Pareto Front

Figure 5.10 shows the Pareto front obtained by plotting the natural frequency (Hz) against the mass (kg) for our two-objective design trade-off. A few key observations are:

- **Monotonic trade-off.** As mass increases, natural frequency also increases. Thus, heavier designs yield strictly higher frequencies, but at the expense of larger mass.
- **Increasing and Diminishing returns.** The Pareto front initially exhibits a concave-upward (convex) shape, indicating a region of increasing returns, where small increases in mass lead to disproportionately large gains in natural frequency. This trend continues until approximately 825 kg and 6.67 Hz, beyond which the curve transitions into a concave-downward regime, characteristic of diminishing returns; further increases in mass yield progressively smaller improvements in frequency.
- **Extremes.**
  - min-mass solution: at around $(m, f) \approx (83.25\,\text{kg},\ 0.24\,\text{Hz})$, frequency is lowest but mass is minimal.
  - max-frequency solution: at around $(m, f) \approx (9812.50\,\text{kg},\ 13.83\,\text{Hz})$, frequency is highest but mass is very large.

This Pareto front presents a family of equally non-dominated designs, giving engineers the flexibility to select the most appropriate compromise based on project requirements. In practice, choosing a point on the curve amounts to deciding how much additional mass one is willing to accept in exchange for a desired increase in natural frequency.

6 Conclusions

This thesis demonstrated the formulation and solution of several engineering design problems using optimization techniques and computational tools. By combining mathematical optimization methods with mechanics principles, we obtained optimal designs for structural systems under various objectives and constraints. In the scaffolding problem (Chapter 5.1), a linear programming model was constructed to maximize the load supported by a system of ropes and beams, demonstrating how LP yields the global optimum efficiently. In the beam design problem (Chapter 5.2) and truss design problem (Chapter 5.3), nonlinear programming (using SLSQP) was employed to minimize structural weight subject to stress and deflection constraints; these examples showed how gradient-based methods can handle engineering constraints and design variables effectively.

In the water-tank column problem (Chapter 5.4), we explored multi-objective optimization. Single-objective solutions were found separately for minimum mass and maximum frequency. Then, a weighted-sum approach and a genetic algorithm (NSGA-II) were used to generate the Pareto front illustrating the trade-off between mass and natural frequency. The shape of the Pareto front reveals how the relationship between these objectives evolves, offering engineers a clear framework for selecting an efficient design compromise based on performance priorities. All numerical solutions were implemented with Python (using SciPy and pymoo libraries) and validated where appropriate by MATLAB results. Graphical solutions (from MATLAB) and numerical results (from Python) were in agreement, confirming the correctness of the formulations and code.

Overall, the key findings are that the selected optimization methods are well-suited to the engineering problems considered. The formulation of constraints based on fundamental mechanics principles was appropriately applied, ensuring realistic and physically meaningful models. The optimization algorithms (simplex for LP, SLSQP for NLP, and NSGA-II for MOO) consistently produced valid and efficient solutions. These case studies illustrate how optimization can effectively support engineering design by quantifying trade-offs, enhancing performance, minimizing material usage, and ensuring adherence to mechanical constraints.

References

[1] Andreas Antoniou and Wu-Sheng Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer, New York, NY, 2007.

[2] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, Hoboken, NJ, 4th edition, 2010.

[3] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, Hoboken, NJ, 3rd edition, 2006.

[4] Ferdinand P. Beer, E. Russell Johnston, John T. DeWolf, and David F. Mazurek. *Mechanics of Materials*. McGraw-Hill, New York, NY, 6th edition, 2014.

[5] Martin Branda. Optimization with application in finance - exercises. Lecture notes, Faculty of Mathematics and Physics, Charles University, Prague, February 2023.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tej Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[7] James M. Gere and Barry J. Goodno. *Mechanics of Materials*. Cengage Learning, Boston, MA, 7th edition, 2012.

[8] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of Physics*. Wiley, Hoboken, NJ, 10th edition, 2014.

[9] R. C. Hibbeler. *Engineering Mechanics: Statics*. Pearson, Boston, MA, 12th edition, 2013.

[10] Yingjie Ma, Xi Gao, Chao Liu, and Jie Li. Improved sqp and slsqp algorithms for feasible path-based process optimisation. *Computers & Chemical Engineering*, 188:108751, 2024.

[11] J. L. Meriam and L. G. Kraige. *Engineering Mechanics: Statics*. Wiley, Hoboken, NJ, 7th edition, 2012.

[12] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, 2nd edition, 2006.

[13] Gade Pandu Rangaiah. *Multi-Objective Optimization: Techniques and Applications in Chemical Engineering*, volume 1 of *Advances in Process Systems Engineering*. World Scientific, Singapore, 2009.

[14] Singiresu S. Rao. *Engineering Optimization: Theory and Practice*. John Wiley & Sons, Hoboken, NJ, 4th edition, 2009.

[15] Singiresu S. Rao. *Mechanical Vibrations*. Prentice Hall, Upper Saddle River, NJ, 5th edition, 2010.

[16] Ronald L. Rardin. *Optimization in Operations Research*. Pearson, Boston, MA, 2nd edition, 2015.

[17] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization: Methods and Applications*. John Wiley & Sons, New York, NY, 2nd edition, 2000.

[18] P. Venkataraman. *Applied Optimization with MATLAB Programming*. John Wiley & Sons, Hoboken, NJ, 2002.

7 Appendix: Python Codes

7.1 Problem 1: Scaffolding Optimization

```python
from scipy.optimize import linprog

# Define the maxmum tension that the ropes can withstand
W1 = 100
W2 = 100
W3 = 100
W4 = 100

# Objective function coefficients (minimizing -x1 - x2 - x3 - x4)
c = [-1, -1, -1, -1]

# Inequality constraints matrix (A_ub * x <= b_ub)
A_ub = [
    [1/2,0,3/8,3/20],     # TA <= W1
    [1/2,0,1/8,1/20],     # TB <= W1
    [0, 1/2, 2/5, 17/50], # TC <= W2
    [0, 1/2, 1/10, 23/50],# TD <= W2
    [0, 0, 1/2, 1/5],      # TE <= W3
    [0, 0, 1/2, 3/10],     # TF <= W3
    [0, 0, 0, 1/2]         # TG & TH<= W4 (same expression)
]
# Compute right-hand side values for inequality constraints
b_ub = [
    W1,   # TA
    W1,   # TB
    W2,   # TC
    W2,   # TD
    W3,   # TE
    W3,   # TF
    W4    # TG & TH (same expression)
]

# Define variable bounds (x1, x2, x3, x4 must be at least 50)
x_bounds = [(50, None), (50, None), (50, None), (50, None)]

# Solve the linear programming problem
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=x_bounds, method='highs'
    )

# Check if an optimal solution was found
if result.success:
    x1, x2, x3, x4 = result.x
    total = x1 + x2 + x3 + x4

    # Display optimal solution
    print("Optimal solution found:")
    print(f"x1 = {x1:.2f}")
```

```
    print(f"x2 = {x2:.2f}")
    print(f"x3 = {x3:.2f}")
    print(f"x3 = {x4:.2f}")
    print(f"Total (x1 + x2 + x3 +x4) = {total:.2f}")

    # Calculate actual constraint values using the optimal solution
    TA = (1/2)*x1 + (3/8)*x3 + (3/20)*x4
    TB = (1/2)*x1 + (1/8)*x3 + (1/20)*x4
    TC = (1/2)*x2 + (2/5)*x3 + (17/50)*x4
    TD = (1/2)*x2 + (1/10)*x3 + (23/50)*x4
    TE = (1/2)*x3 + (1/5)*x4
    TF = (1/2)*x3 + (3/10)*x4
    TG = (1/2)*x4
    TH = (1/2)*x4

    # Display constraint values at the optimal solution
    print("\nConstraint Values at Optimal Solution:")
    print(f"TA = {TA:.2f} (<= W1 = {W1})")
    print(f"TB = {TB:.2f} (<= W1 = {W1})")
    print(f"TC = {TC:.2f} (<= W2 = {W2})")
    print(f"TD = {TD:.2f} (<= W2 = {W2})")
    print(f"TE = {TE:.2f} (<= W3 = {W3})")
    print(f"TF = {TF:.2f} (<= W3 = {W3})")
    print(f"TG = {TG:.2f} (<= W4 = {W4})")
    print(f"TH = {TH:.2f} (<= W4 = {W4})")
else:
    # Display error message if no solution was found
    print("No optimal solution found. Message:", result.message)
```

7.2 Problem 2: Beam Optimization

```
import numpy as np
from scipy.optimize import minimize

# Define the objective function (to be minimized)
def objective(vars):
    x1, x2 = vars
    return x1 * x2

# Define the inequality constraints (must be >= 0)
def constraint1(vars):
    x1, x2 = vars
    return x1 * x2**2 - (9 / 2200)  # Constraint 1

def constraint2(vars):
    x1, x2 = vars
    return x1 * x2**3 - (725 / (1656 * 10**4))  # Constraint 2

def constraint3(vars):
    x1, x2 = vars
    return x2 - x1  # Ensures x2 >= x1
```

```

# Define variable bounds for x1 and x2
bounds = [(0.04, 0.12), (0.06, 0.20)]  # (min, max) values

# Initial guess for the optimization
initial_guess = [0.04, 0.06]

# Define the constraints in dictionary format
constraints = [
    {'type': 'ineq', 'fun': constraint1},  # constraint1 >= 0
    {'type': 'ineq', 'fun': constraint2},  # constraint2 >= 0
    {'type': 'ineq', 'fun': constraint3}   # constraint3 >= 0
]

# Solve the optimization problem using SLSQP
result = minimize(fun=objective, x0=initial_guess, method='SLSQP', bounds
   =bounds,
    constraints=constraints)

# Display the results
if result.success:
    print("Optimization was successful!")
    print(f"Optimal values: x1 = {1000*result.x[0]:.4f} mm, x2 = {1000*
       result.x[1]:.4f} mm")
    print(f"Minimum W = {(76.5*10**3)*result.fun:.4f} N")  # Compute the
       minimum W value
else:
    print("Optimization failed:", result.message)
```

7.3 Problem 3: Truss Optimization

```
import numpy as np
from scipy.optimize import minimize

# Constants (Problem Data)
b = 0.5  # Half base in meters
h = 1  # Height in meters
W = 25000  # Load in Newtons
E = 200e9  # Young's modulus (Pa)
sigma_y = 310e6  # Yield stress (Pa)
rho = 7850  # Weight density (N/m^3)

# Define the objective function (to be minimized)
def objective(x):
    d, t = x
    return d * t

# Define the inequality constraints (must be >= 0)
def constraint_g1(x):
    d, t = x
    return d * t - (W * np.sqrt(b**2 + h**2)) / (np.pi * h * sigma_y)
```

```
21
22 def constraint_g2(x):
23     d, t = x
24     return (d**3 * t + t**3 * d) - (8 * W * (b**2 + h**2)**1.5) / (np.pi
       **3 * E * h)
25
26 def constraint_g3(x):
27     d, t = x
28     return d - t  # Ensuring t <= d (d - t >= 0)
29
30 # Initial guess
31 x0 = [0.02, 0.02]
32
33 # Bounds for d and t
34 bounds = [(0.02, 0.05), (0.005, 0.05)]  # (min, max) values
35
36 # Define constraints in dictionary format
37 constraints = [
38     {'type': 'ineq', 'fun': constraint_g1},
39     {'type': 'ineq', 'fun': constraint_g2},
40     {'type': 'ineq', 'fun': constraint_g3}
41 ]
42
43 # Solve the optimization problem using SLSQP
44 result = minimize(objective, x0, method='SLSQP', bounds=bounds,
       constraints=constraints)
45
46 # Display the results
47 if result.success:
48     print("Optimal␣Solution:")
49     print(f"d␣=␣{1000␣*␣result.x[0]:.4f}␣mm")
50     print(f"t␣=␣{1000␣*␣result.x[1]:.4f}␣mm")
51     print(f"Minimum␣Weight␣(d*t)␣=␣{np.sqrt(b**2␣+␣h**2)␣*␣np.pi␣*␣2␣*␣
           rho␣*␣result.fun:.4f}␣N")
52 else:
53     print("Optimization␣failed:", result.message)
```

7.4 Problem 4: Water Tank Column Optimization

Weighted Multi Objective Approach

```
1  import numpy as np
2  from scipy.optimize import minimize
3
4  # Constants
5  rho = 7850        # Density of the material (kg/m^3)
6  l = 5             # Column length (m)
7  E = 200e9         # Young's modulus (Pa)
8  M = 1000         # Mass of the tank (kg)
9  g = 9.81          # Acceleration due to gravity (m/s^2)
10 sigma_max = 250e6  # Maximum permissible stress (Pa)
```

```
alpha = 0.02     # Weighting factor for mass and frequency optimization

# Objective function to minimize mass
def objective1(x):
    x1, x2 = x
    mass = rho * l * x1 * x2  # Compute mass of the column
    return mass

# Objective function to maximize frequency (negative for minimization
   approach)
def objective2(x):
    x1, x2 = x
    term = (E * x1 * x2**3) / (4 * l**3 * (M + (33/140) * rho * l * x1 *
       x2))
    frequency = np.sqrt(term)
    return -frequency  # Negative since we minimize by default

# Define the inequality constraints (must be >= 0)
def constraint1(x):
    x1, x2 = x
    return sigma_max - (M * g) / (x1 * x2)

def constraint2(x):
    x1, x2 = x
    return (np.pi**2 * E * x2**2) / (48 * l**2) - (M * g) / (x1 * x2)

# Bounds for design variables (x1 and x2)
bounds = [(0.04, 0.5), (0.04, 0.5)]

# Constraints in dictionary format
constraints = [
    {'type': 'ineq', 'fun': constraint1},
    {'type': 'ineq', 'fun': constraint2}
]

# Initial guess for optimization
x0 = [0.1, 0.1]

# Solve for minimum mass
result1 = minimize(objective1, x0, method='SLSQP', bounds=bounds,
   constraints=constraints)
print("\nMin Mass Solution:")
print(f"x1 = {result1.x[0]:.6f} m")
print(f"x2 = {result1.x[1]:.6f} m")
mass1 = rho * l * result1.x[0] * result1.x[1]
print(f"Mass = {mass1:.2f} kg")
term1 = (E * result1.x[0] * result1.x[1]**3) / (4 * l**3 * (M + (33/140)
   * rho * l * result1.x[0] * result1.x[1]))
frequency1 = np.sqrt(term1) / (2 * np.pi)
print(f"Frequency = {frequency1:.2f} Hz")

# Solve for maximum frequency
```

```
result2 = minimize(objective2, x0, method='SLSQP', bounds=bounds,
    constraints=constraints)
print("\nMax Frequency Solution:")
print(f"x1 = {result2.x[0]:.6f} m")
print(f"x2 = {result2.x[1]:.6f} m")
mass2 = rho * l * result2.x[0] * result2.x[1]
term2 = (E * result2.x[0] * result2.x[1]**3) / (4 * l**3 * (M + (33/140)
    * rho * l * result2.x[0] * result2.x[1]))
frequency2 = np.sqrt(term2) / (2 * np.pi)
print(f"Mass = {mass2:.2f} kg")
print(f"Frequency = {frequency2:.2f} Hz")

# Multi-objective optimization balancing mass and frequency
def objective(x):
    x1, x2 = x
    mass = rho * l * x1 * x2
    term = (E * x1 * x2**3) / (4 * l**3 * (M + (33/140) * rho * l * x1 *
        x2))
    frequency = np.sqrt(term)
    return (alpha / mass1) * mass - ((1 - alpha) / frequency2) *
        frequency

# Solve for weighted multi-objective optimization
result = minimize(objective, x0, method='SLSQP', bounds=bounds,
    constraints=constraints)

# Display final results
print(f"\nNormalized Weighted Solution When alpha = {alpha}")
print(f"x1 = {result.x[0]:.6f} m")
print(f"x2 = {result.x[1]:.6f} m")
mass = rho * l * result.x[0] * result.x[1]
term = (E * result.x[0] * result.x[1]**3) / (4 * l**3 * (M + (33/140) *
    rho * l * result.x[0] * result.x[1]))
frequency = np.sqrt(term) / (2 * np.pi)
print(f"Mass = {mass:.2f} kg")
print(f"Frequency = {frequency:.2f} Hz")
```

NSGA-II Method

```
import numpy as np
from pymoo.core.problem import Problem
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
import matplotlib.pyplot as plt

# Define the optimization problem
class ColumnProblem(Problem):
    def __init__(self):
        # Constants
        self.rho = 7850       # Density (kg/m^3)
        self.l = 5            # Column length (m)
```

```
        self.E = 200e9          # Young's modulus (Pa)
        self.M = 1000           # Mass of tank (kg)
        self.g = 9.81           # Gravity (m/s^2)
        self.sigma_max = 250e6  # Permissible stress (Pa)

        super().__init__(
            n_var=2,            # Variables: x1, x2
            n_obj=2,            # Objectives: minimize mass, maximize
               frequency
            n_constr=2,         # Constraints: g1, g2
            xl=[0.04, 0.04],    # Lower bounds
            xu=[0.5, 0.5]       # Upper bounds
        )

    def _evaluate(self, X, out, *args, **kwargs):
        x1, x2 = X[:, 0], X[:, 1]

        # Objectives
        f1 = self.rho * self.l * x1 * x2  # Minimize mass
        term = (self.E * x1 * x2**3) / (4 * self.l**3 * (self.M +
           (33/140) * self.rho * self.l * x1 * x2))
        f2 = -np.sqrt(term)/(2*np.pi)  # Maximize frequency (minimize
           negative)

        # Constraints (g <= 0)
        g1 = (self.M * self.g) / (x1 * x2) - self.sigma_max  # Stress
           constraint
        g2 = (self.M * self.g) / (x1 * x2) - (np.pi**2 * self.E * x2**2)
           / (48 * self.l**2)  # Buckling

        out["F"] = np.column_stack([f1, f2])
        out["G"] = np.column_stack([g1, g2])

# Solve the problem
problem = ColumnProblem()
algorithm = NSGA2(pop_size=100)
res = minimize(
    problem,
    algorithm,
    ('n_gen', 100),  # Terminate after 100 generations
    seed=1,
    verbose=False
)

# Plot Pareto front (mass vs. natural frequency)
plt.scatter(res.F[:, 0], -res.F[:, 1], c="blue")
plt.xlabel("Mass (kg)")
plt.ylabel("Natural Frequency (Hz)")
plt.title("Pareto Front")
plt.show()

# Extract solutions (x1, x2) from `res.X`
```

```
60 print("Optimal␣designs␣(x1,␣x2):")
61 print(res.X)
```