Topology Optimisation: a Modification of the SIMP Method for Generally Orthotropic Lamina

Nathan Dale Haskins

October 24, 2023

Acknowledgements

Thank you to my supervisor, Sa-aadat Parker, for allowing me the freedom to explore the field of topology optimisation. This course, and our weekly discussions on topology optimisation, have been a highlight of my undergraduate degree.

Thank you to my friend, Dino Claro, for providing an outsider's perspective, enabling me to realise my aim of creating a comprehensive resource for students wishing to learn about topology optimisation.

Abstract

This document presents a modification of the Solid Isotropic Material with Penalization (SIMP) method extending its use case to orthotropic materials. The document contains the necessary mathematical formulations for orthotropic materials and topology optimisation theory, and provides a comprehensive guide for Mechanical Engineering students, not only allowing for the extension in understanding of composite mechanics and `Python`-based optimisation, but also providing a framework for students to develop their own topology optimisation code.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

BEYOND THE BEAM
An Introduction

"Somewhere, something incredible is waiting to be known."

- Carl Sagan

1.1 Background

One of the fundamental aspects of engineering is optimisation. Optimisation is the procedure whereby a system's design variables are iteratively modified to meet some objective, all while adhering to a defined set of constraints [1]. In structural mechanics, it is desirable to minimise the weight of a structure, while maintaining or maximising its stiffness. Intuitively, one might consider a solid beam and imagine that the beam would be as stiff, or *stiffer* if it had less material, but perhaps had a geometry that was triangulated like a truss beam.

This is a simple example, but how would one approach this problem not having previously seen a truss beam? Topology optimisation is one solution and involves discretizing a design domain into finite elements, small units representing the structure, and analyses the mechanical behaviour at their connecting points or nodes. By interpreting the nodal displacements, the changes in positions of these nodes under applied forces, and updating the design accordingly, an optimal material distribution is achieved.

1.2 Scope and Objectives

This project aims to present a comprehensive guide for readers wishing to implement a topology optimisation program that is capable of modelling generally orthotropic lamina. To achieve this, the following objectives have been identified:

(a) Develop a `Python`-based topology optimisation framework and documentation thereof.

(b) Documentation of mathematical derivations that are not readily available in the existing literature.

(c) Validate the developed framework against existing benchmarks in literature.

(d) Analysis of topology-optimised structures and a comparison of the results at differing fibre angles.

1.3 Document Outline

This document is composed of 6 chapters as described below:

Chapter 1: Introduction presents a brief overview of the significance of topology optimisation as a structural optimisation tool. The aim and objectives are also presented in this chapter.

Chapter 2: A Literature Review provides an overview of the field of topology optimisation, its use in engineering industries, and the some educational resources on topology optimisation available to students. This chapter also introduces the SIMP method, which is the basis for the topology optimisation program developed in this project.

Chapter 3: Methodology - Methodological Framework presents the mathematical theory that underpins the SIMP method.

Chapter 4: Methodology - Numerical Implementation presents the numerical implementation of the SIMP method. A `Python` program developed by the author is also presented in this chapter.

Chapter 5: Results and Discussion presents the results of the work conducted in this project. This chapter focuses on how orthotropic materials, with fibre angles, affect converged material distribution and compliance.

Chapter 6: Conclusion concludes this report, summarising the information presented in the previous chapters.

In addition to the chapters described above, this document contains three appendices. Appendix A provides derivations relating to solid mechanics of materials, including a derivation of the local and reduced stiffness matrices for generally orthotropic lamina, and a derivation of the elemental stiffness matrix. Appendix B provides additional information on topology optimisation, including information on sensitivity analysis and the Optimality Criteria method. Appendix C contains the complete `Python` code for a topology optimisation program called `Toppie`, developed by the author of this document.

Chapter 2

ON TOPOLOGY OPTIMISATION
A Literature Review

"You have to know the past to understand the present."

- Carl Sagon

2.1 Background

Shape, size, and topology optimisation are all subfields of structural and multidisciplinary optimisation (SMO) and are used in a variety of structural mechanics problems. Shape optimisation focuses on refining the external geometry of a structure while keeping material distribution constant, achieved by adjusting its shape, height, or thickness. In size optimisation, each structural dimension is considered as a design variable, which will then be varied to minimise or maximise an objective function. By far the most widely used method is topology optimisation, which is concerned with the optimal distribution of material within a design domain [2]. It differs from the previous two methods in that the physical shape and size of the optimised structure are unknown. It merely optimises the material distribution to satisfy pre-defined loading conditions and constraints.

In the late 1980s, shape optimisation often involved the final design looking very similar to the initial design, which necessitated the recomputation of finite element grids. In 1988, Bendsøe and Kikuchi collaborated to develop a "material distribution problem" approach by blending homogenization techniques with structural optimisation theory and finite element analysis principles, which ignited extensive subsequent research in the field of topology optimisation [3].

Topology optimisation can be implemented using a variety of methods from density-based approaches [4], level set methods [5], and methods influenced by nature like particle swarm optimisation and evolutionary-based optimisation [6, 7], among others. In each method, finite element analysis discretises the design domain and the aim is then to seek which elements within the design domain should have material and which should remain void, thereby optimising the material distribution. These approaches all have their merits and all seek to solve the problem of determining the optimal material distribution to achieve some objective.

2.2 Types of Topology Optimisation Objectives

As the name suggests, topology optimisation is an optimisation procedure and so it involves the minimisation or maximisation of some objective function. The material distribution of a structure or component will often influence the structure's properties, be that in stiffness, strength, or fatigue and buckling resistivity, for example. The objective of a topology optimisation problem thus seeks to find an optimal material distribution that satisfies some pre-defined performance criteria.

In the field of topology optimisation, many types of problems have been solved. Concerning mechanical performance optimisation, structures have been optimised to maximise stiffness (known as compliance minimisation problems) [8, 9], maximise strength where the objective is for a structure to handle a maximum load without failure [10], vibrational optimisation which often involves seeking a material distribution of a structure operating at different eigenmodes or natural frequencies [11], and fatigue and buckling resistance problems. [12, 13].

Thermal objective topology optimisation is concerned with seeking topologies that optimise a structure's heat transfer properties, reduce thermoelastic stress, improve thermal insulation, or heat conduction [14, 15, 16]. Fluid flow path topology problems have been solved leading to optimal designs of diffusers that have minimum power dissipation [17]. It is evident from the examples listed above that topology optimisation is involved in a wide variety of industries including aerospace, automotive and HVAC.

2.3 Educational Resources in the Field: the SIMP Method

Given topology optimisation's vast applicability, it is unsurprising that several educational resources have been produced in the field. Perhaps because of its ease of implementation and widespread use, the Solid Isotropic Material with Penalization (SIMP) method is a popular choice used in educational papers on topology optimisation [2].

The SIMP method, which is a density-based approach was initially proposed by Bendsøe in 1989 [18], however, the method's official definition was later defined by Rozany, Bendsøe et al. in the early 1990s [19, 20, 21]. Rather than using a discrete 0-1 approach (void and material), the SIMP method relaxes the problem by introducing an intermediate density design variable for each element in the design domain. Each element then takes on an intermediate density between 0 and 1, which is then penalised to approach a value of 0 or 1 through the use of a penalisation exponent. The advantage of this approach is that the discrete binary (0-1) optimisation procedure is extremely computationally expensive and by relaxing this discrete condition, the combinatorial optimisation problem becomes a continuous one, which can be solved using efficient gradient-based methods.

Initially proposed as the "direct approach" by Bendsøe in 1989, the SIMP method was criticised by Bendsøe *himself* when he highlighted the lack of physical meaning for the method as it resulted in intermediate densities or areas of partial materiality - a phenomenon not seen in nature [18]. This objection was later retracted by Bendsøe and Sigmund in 1999 in a paper where they set out to settle the long ongoing discussion on the physical relevance of the SIMP interpolation scheme, with a particular focus on "grey-scale density-like interpolation schemes" for composites [22]. Having settled this,

the SIMP method became incredibly popular. This is especially evident in several education papers that make use of the method, of which some are mentioned below.

One of the most widely cited papers is an educational paper by Ole Sigmund published in 2001 on standard compliance minimisation and included a 99-line `MATLAB` code (`top99`) [8]. Another education paper was published by Andreassen et al. in 2011 which included an 88-line `MATLAB` program (`top88`) that modified Sigmund's code, improving efficiency by vectorising `for` loops and including a density filter to avoid common checkerboarding effects[1] in the final topology [9]. While `top99` and `top88` dealt with compliance minimisation of 2D structures, Kai Liu and Andrés Tovar implemented a 169-line `MATLAB` code that dealt with compliance minimisation of 3D structures [23]. Another 3D optimisation code that was inspired by top99 and top88, has been implemented in `C#` by Nikos D. Lagaros in 2019 and has integrated the use of SAP2000, a structural design and analysis software [24]. Considering parallel computing, Balay et al. have implemented a 3D topology optimisation framework using a `Python` wrapper called PETSc and is capable of computing large-scale problems with over 100 million design variables [25, 2].

2.4 Topology Optimisation of Orthotropic Materials

Unlike isotropic materials whose mechanical properties are the same in all directions, orthotropic materials have differing mechanical properties along their three orthogonal principal axes [26]. A lamina is a thin layer of material that is used as the building blocks for laminates (stacked laminae) forming larger composite materials, such as glass-epoxy and graphite-epoxy composites. A laminate is often stacked with multiple unidirectional laminae at differing fibre angles. The reason for this is because unidirectional lamina, whose fibres are all aligned in the same direction, are highly anisotropic meaning that they have different mechanical properties in different directions. Specifically, unidirectional laminae have improved stiffness in their fibre direction compared to their transverse direction [27].

Given the excellent specific strength and stiffness, damping properties and corrosion resistance of composites, especially those composed of orthotropic materials, extensive research has taken place in the field of topology optimisation with a particular focus on orthotropic materials.

Shu Li and S. N. Atluri optimised both topology and fibre angle with the objective of mean compliance of anisotropic structures in a paper published in 2008 and made use of a spacial discretization algorithm called the Meshless Local Petrov-Galerkin method [28]. In 2015, Robert Hoglund and Douglas E. Smith published a paper exploring how the direction of the deposited bead in the Fused Deposition Modeling (FDM) process significantly affects the properties of orthotropic materials reinforced with short fibres [29]. Hong-Ling Ye et al. produced a paper in 2017 on plate/shell topology optimisation of orthotropic materials under buckling conditions [13]. A recent paper published in 2022 by Derek Harvey and Pascal Hubert implemented a minimum compliance 3D topology optimisation of sandwich structures containing anisotropic shells [30].

[1] Checkerboarding is a common numerical instability issue in topology optimisation. For more information refer to Section 3.2.3

This present work seeks to bridge the gap between elementary and advanced optimisation applications, enabling a more seamless transition to 3D topology optimisation of layered shell structures, such as the work by Harvey and Hubert. By concentrating on a single-layer generally orthotropic lamina, this work provides the necessary groundwork for advanced studies with specialised composite configurations, whether they be multi-layered, thermally conductive, or subject to buckling constraints, as explored in a variety of preceding works mentioned above.

Chapter 3

THE SIMP METHOD
Methodological Framework

"Biology is the most sophisticated manufacturing technology we know of today. Its material property range and resolution is unparalleled, offering opportunities we can only begin to imagine."

- Neri Oxman

This chapter provides a mathematical framework for a compliance minimisation topology optimisation problem that can handle both isotropic and orthotropic materials. In addition, this chapter provides the necessary theory for the numerical implementation in Chapter 4.

3.1 Finite Element Mathematical Formulation

Consider the generalised design domain, $\boldsymbol{\Omega}$, composed of a linearly elastic homogeneous isotropic or orthotropic material, and occupying the space $\boldsymbol{\Omega} \subset \mathbb{R}^2$, as shown in Figure 3.1 below. $\boldsymbol{\Omega}$ is located within the x-y global coordinate system and is bounded by the displacement boundary condition, $\boldsymbol{\Gamma}_{\mathrm{d}}$, and a load, $\boldsymbol{F}$, located at boundary $\boldsymbol{\Gamma}_{\mathrm{F}}$. $\boldsymbol{\Omega}$ can be discretized into a finite element grid, $\boldsymbol{\mathcal{G}}$, and each element in this grid can be represented as ξ_i, where i represents the elemental index.

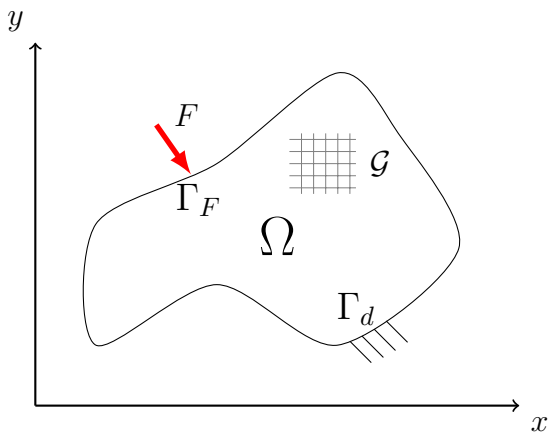

Figure 3.1: Generalized Design Domain ($\boldsymbol{\Omega}$)

In the finite element method, each element, ξ_i, will have an associated elemental stiffness. Considering homogeneous materials, the stiffness of each element can be represented with an associated elemental stiffness matrix, $\boldsymbol{K}_e$. These elemental stiffness matrices will all have equivalent stiffness and can be calculated with the formula below.

$$\boldsymbol{K}_e = \int_{\xi_i} \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \, d\xi_i \tag{3.1}$$

Where $\boldsymbol{B}$ is the strain-displacement matrix and $\bar{\boldsymbol{Q}}$ is the constitutive stiffness matrix or reduced stiffness matrix in the global domain. $\boldsymbol{B}$ relates an element's strain to its nodal displacements, while $\bar{\boldsymbol{Q}}$ relates an element's stress to its strain, both of which are fundamental for calculating the elemental stiffness matrix.

Unlike isotropic materials, generally orthotropic lamina do not have their reduced stiffness matrix aligned in the global coordinate system, and thus a transformation from the local to the global domain is required. Figure 3.2 shows a generally orthotropic lamina with its local $(1-2)$ coordinate system positioned at an angle θ relative to the global $(x-y)$ coordinate system.

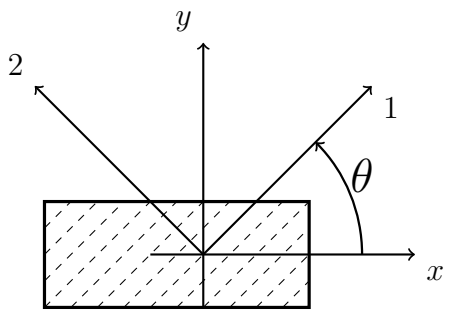

Figure 3.2: Generally Orthotropic Angled Lamina

Derivations related to generally orthotropic lamina, including the reduced stiffness matrix in both local and global coordinate systems, as well as the elemental stiffness matrix, are provided in Appendix A.

Having calculated $\boldsymbol{K_e}$, the global stiffness matrix, $\boldsymbol{K}$, can be assembled. $\boldsymbol{K}$ represents the stiffness of the entire domain, $\boldsymbol{\Omega}$, and describes how the nodal forces, $\boldsymbol{F}$, are related to the nodal displacements, $\boldsymbol{u}$.

Finally, the nodal displacements, $\boldsymbol{u}$, can be calculated by solving the following equation:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{F} \tag{3.2}$$

3.1.1 Incorporating the SIMP Method

In the SIMP method, $\boldsymbol{K}$ is a function of the design variables, $\boldsymbol{x}$ (containing the density of each element), and is computed by summing the elemental stiffness matrix $\boldsymbol{K_e}$, weighted by the design variables raised to a penalisation factor, p. As mentioned in Chapter 2, the penalisation factor is used to discourage intermediate element density values between 0 and 1. $\boldsymbol{K(x)}$ is represented in its mathematical form below.

$$\boldsymbol{K}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^p \boldsymbol{K_e} \tag{3.3}$$

where p is the penalisation factor and n is the number of elements in the design domain.

3.2 Topology Optimisation Procedure

The following sections outline the topology optimisation problem formuation, sensitivity analysis, a filtering technique, and the update procedure for the design variables.

3.2.1 Topology Optimisation Problem Definition

In the context of a compliance minimisation topology optimisation problem, the objective is to minimize the compliance of the structure, thereby maximizing its stiffness, subject to certain constraints. The optimisation problem takes the form of the following well-known formulation and is based on the SIMP method, popularised by Bendsøe and Kikuchi in 1989 and Rozvany and Zhou in 1992 [18, 19]:

$$\begin{aligned} \text{find} \quad & \boldsymbol{x} = \{x_1, x_2, ..., x_n\}^T \\ \text{minimize} \quad & C(\boldsymbol{x}) = \boldsymbol{F}^T \boldsymbol{u} \\ \text{subject to} \quad & \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{F} \\ & V(\boldsymbol{x}) \leq V_{\max} \\ & 0 \leq x_{min} \leq x_i \leq 1, \quad \forall i = 1, 2, ..., n, \end{aligned} \tag{3.4}$$

$C(\boldsymbol{x})$ is the compliance of the structure, and gives an indication of the structure's flexibility, as compliance is the inverse of stiffness. $V(\boldsymbol{x})$ is the volume of the structure, which is restricted to an upper bound $V_{\max}$. This represents the volume constraint and can be understood as the maximum permissible volume of material in the design domain. If a volume constraint of 0.5 is specified, the optimisation algorithm will converge to a design where the volume of material is such that half the design domain is solid material and the other half is void. $\boldsymbol{x}$ represents the vector of design variables, which hold the density values of each element and are limited to a lower bound of $x_{\min}$ and an upper bound of 1. $x_{\min}$ is typically set to some small non-zero value to prevent singularities during the assembly of the global stiffness matrix, $\boldsymbol{K(x)}$. Finally, n is the number of elements in the design domain.

By solving this optimisation problem, an optimised material distribution within the design domain, $\boldsymbol{\Omega}$, can be achieved, leading to a structure with minimized compliance and thus maximized stiffness.

3.2.2 Sensitivity Analysis

Having defined the objective function, $C(\boldsymbol{x})$, it is necessary to determine how a change in the design variables, $\boldsymbol{x}$, will impact this objective. Sensitivity analysis results in a derivative or gradient of the objective function with respect to the design variables. In broad terms, it provides an indication of the impact that increasing or decreasing an element's density has on the structure's overall compliance. Knowing this, the topology optimisation algorithm can appropriately update the design to minimise this objective and thus maximise the structure's stiffness.

The formula below shows how changing the design variable of a single element affects the overall compliance of the structure. A thorough derivation of this expression is provided in Section B.1 of Appendix B.

$$\frac{\partial C}{\partial x_i} = -p x_i^{p-1} C_i \tag{3.5}$$

where x_i is the density of the element and C_i is the element's contribution to the overall compliance of the structure. The negative sign in the equation above ensures that the optimisation process minimises the compliance of the structure.

C_i is calculated as follows:

$$C_i = \mathbf{u}_i^T \mathbf{f}_i \tag{3.6}$$

where $\mathbf{u}_i$ and $\mathbf{f}_i$ are the nodal displacement and force vectors of a single element.

3.2.3 Filtering Procedure

A common issue with the SIMP method is known as the checkerboarding effect, which is when the converged topology has portions of the design that have alternating solid and void elements, as shown in Figure 3.3. It occurs as a result of a numerical instability and not an "optimal" design [31].

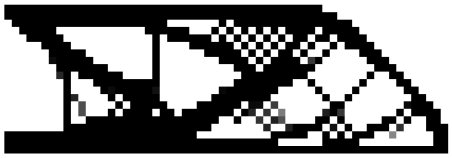

Figure 3.3: Checkerboarding Effect

To address this issue, many solutions have been proposed from an increased mesh size to the use of higher order finite elements [32]. In this work, a density-based sensitivity filtering technique is used to mitigate the checkerboarding effect [33]. In addition, the filtering procedure also serves to resolve the mesh-independency problem. This occurs when an increased mesh size, say from 60×20 elements to 600×200 elements converge to different topologies, instead of an improved resolution of the underlying optimal topology. For more details on the mesh-independency problem, refer to [32].

The filter modifies the sensitivity of an element of interest with the use of a weighted average of the element sensitivities neighbouring this element. To achieve this, the filter uses a weight factor, H_{ij}, which is expressed in the following piece-wise equation:

$$H_{ij} = \begin{cases} R_{\min} - \text{dist}(i,j) & \text{if } \text{dist}(i,j) \leq R_{\min}, \\ 0 & \text{if } \text{dist}(i,j) > R_{\min}. \end{cases} \tag{3.7}$$

where $R_{\min}$ is the filter radius, and $\text{dist}(i,j)$ is a function that calculates the distance between the centres of elements i and j. H_{ij} evaluates to zero for points outside this radius and reaches its maximum value at the centre of the kernel. Figure 3.4 shows a visual representation of the convolution kernel where the grayscale intensity represents the weight of each cell in the kernel.

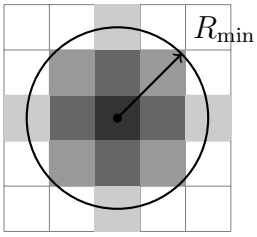

Figure 3.4: Convolution Kernel based on $R_{\min}$

In the figure above, darker cells correspond to higher weights, which are closer to the centre, while lighter cells have lower weights. These weights are normalized such that the sum of all elements in the kernel equals 1. This ensures that the filtered sensitivities preserve the scale of the original sensitivities, thereby preventing any unintended amplification or diminution of their magnitudes.

This kernel is convolved with the element sensitivities, weighted by their densities, to yield the filtered sensitivities. The convolution operation is performed using 'reflect' mode, which means the input is extended by mirroring its content around the edges, minimizing edge effects during convolution.

Finally, the filtered sensitivity of an element, x_i, in relation to another element, x_j, is given by [34]:

$$\frac{\partial \tilde{C}}{\partial x_i} = \frac{\sum_j H_{ij} \times \frac{\partial C}{\partial x_i}}{\sum_j H_{ij}}. \tag{3.8}$$

In the equation above, $\frac{\partial \tilde{C}}{\partial x_i}$ is the filtered sensitivity of element i, and $\frac{\partial C}{\partial x_i}$ is the unfiltered sensitivity. The summation index j ranges over all elements that fall within the filter radius $R_{\min}$ of an element i, in other words, all elements that are inside the circle in Figure 3.4.

3.2.4 Optimality Criteria (OC) Method

The design update scheme in this work uses a heuristic approach popularised by Bendsøe that was derived from the Karush-Kuhn-Tucker (KKT) optimality conditions [35]. This approach is known as the Optimality Criteria (OC) method and its mathematical formulation is provided below and describes how a design variable, x_i, is updated in each iteration.

$$x_i^{\text{new}} = \begin{cases} \check{x}_i, & \text{if } x_i B_i^{\eta} \leq \check{x}_i, \\ x_i B_i^{\eta}, & \text{if } \check{x}_i \leq x_i B_i^{\eta} \leq \hat{x}_i, \\ \hat{x}_i, & \text{if } x_i B_i^{\eta} \geq x_i. \end{cases} \tag{3.9}$$

where:

$$\check{x}_i = \max\left(x_i - \zeta, x_{min}\right) \tag{3.10}$$

$$\hat{x}_i = \min\left(x_i + \zeta, 1\right) \tag{3.11}$$

$\check{x}_i$ is the lowest bound that an element density can take, while $\hat{x}_i$ is the upper bound. Equation 3.10 ensures that the design variable does not go below x_{min} (the lowest permissible density for an element), and that the element's density does not decrease by more than ζ in a single iteration. Similarly, Equation 3.11 ensures that the design does not exceed the upper bound of 1 and that the element's density does not increase by more than ζ in a single iteration.

ζ is the move limit to update the density, while η is a numerical damping coefficient. For compliance minimisation problems, Bendsøe recommends values for ζ and η as 0.2 and 0.5, respectively [35]. These are found heuristically and are not based on any mathematical derivation. To speed up convergence, a higher value of ζ can be assumed, resulting in the design updating more aggressively, however, this can sometimes lead to numerical instabilities [32].

B_i is a scaling factor used to update the design and is derived from the KKT stationary point condition. It is provided below, however, a detailed derivation is provided in Section B.2 of Appendix B.

$$B_i = -\frac{\frac{\partial \widetilde{C}}{\partial x_i}}{\lambda \cdot \frac{V(\boldsymbol{x})}{V_{\max}}} \tag{3.12}$$

Where λ is the Lagrange multiplier, which can be found with a bisection method, wherein each iteration, the volume constraint described in Equation 3.4 is satisfied.

The optimisation algorithm is terminated when some convergence criteria are met. For example, when the maximum change in density of a single element is less than some tolerance (ϵ):

$$\max\left(|x_i^{\text{new}} - x_i|\right) < \epsilon \tag{3.13}$$

Chapter 4

THE SIMP METHOD
Numerical Implementation

"Talk is cheap. Show me the code."

- Linus Torvalds

This chapter serves to complement the methodology described in Chapter 3. The primary objective is to provide readers with information on the numerical implementation of the SIMP method within the framework of a `Python` program called `Toppie`, developed by the author of this document.

4.1 Program Architecture and Overview

`Toppie` is based on the SIMP approach as outlined in Chapter 3, and has been inspired by Sigmund's `top99` program [8]. `Toppie` is capable of solving 2D compliance minimisation topology optimisation problems for isotropic and orthotropic materials under plane stress.

While there are notable other topology optimisation programs with extensions far exceeding that of `top99` as mentioned in Chapter 2, since `Toppie` has been directly inspired by `top99`, it is worth making a comparison here.

`Toppie` differs from `top99` in many respects:

- `Toppie` is written in `Python` as opposed to `MATLAB`.
- `Toppie` is object-oriented, as opposed to procedural.
- `Toppie` has extended capability of modelling orthotropic materials, as opposed to isotropic materials only.

`Toppie` is comprised of several classes, all of which are described in detail in the following sections. Each section below describes the numerical implementation of the class. The coding sections related to each class are provided in Appendix C.

4.2 Material Class

The `Material` class is responsible for computing the reduced stiffness matrix, $\bar{\boldsymbol{Q}}$, for a specified isotropic or orthotropic material in plane stress, given the material properties. The most important aspect to consider for this section of code is to ensure that the principal axes of the material are aligned with the global axes of the design domain. This is already the case for isotropic materials whose principal axes are aligned with the global axes by definition.

4.2.1 Isotropic Materials

The method `Q_iso()` in the `Material` class computes the reduced stiffness matrix for an isotropic material in plane stress. It is equivalent to computing the following:

$$\bar{\boldsymbol{Q}} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{4.1}$$

where E is Young's modulus and ν is the Poisson's ratio for an isotropic material, used as inputs to the `Material` class.

4.2.2 Orthotropic Materials

For orthotropic materials, the principal axes are not aligned with the global axes as has been explained in Section 3.1. Therefore, first one has to compute the reduced stiffness matrix in the local coordinate domain, $\boldsymbol{Q}$, which is derived in Section A.1 and is provided below for convenience.

$$\boldsymbol{Q} = \begin{bmatrix} \frac{E_1}{1-\nu_{12}\nu_{21}} & \frac{\nu_{21}E_2}{1-\nu_{12}\nu_{21}} & 0 \\ \frac{\nu_{12}E_1}{1-\nu_{12}\nu_{21}} & \frac{E_2}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix} \tag{4.2}$$

$\boldsymbol{Q}$ is then transposed to the global domain by equivalently solving the following in code.

$$\bar{\boldsymbol{Q}} = \boldsymbol{T}^T \cdot \boldsymbol{Q} \cdot \boldsymbol{R} \cdot \boldsymbol{T} \cdot \boldsymbol{R}^T \tag{4.3}$$

where $\boldsymbol{T}$ is the transformation matrix from the local coordinate domain to the global coordinate domain, and $\boldsymbol{R}$ is the Rueter matrix. For further details on how the expression above is derived, please refer to Section A.2 in Appendix A. The method `Q_ortho()` in the `Material` class is responsible for the above computation.

`Toppie` requires a user to specify material properties such as Young's modulus. To improve numerical stability, these parameters are scaled down by a factor of 10^9. For example, to specify a Youngs modulus of 8.27 GPa, the user would specify `E=8.27` in `Toppie`.

The reduced stiffness matrix, $\bar{\boldsymbol{Q}}$, is an important parameter needed for the computation of the elemental stiffness matrix, which is discussed in the next section.

4.3 Element Class

The role of the `Element` class is to compute the elemental stiffness matrix, defined as `ke` in code. A thorough mathematical derivation has already been provided in Section A.3 of Appendix A.

`Toppie` uses a 2×2 Gaussian Quadrature scheme when computing `ke`, which means that there are four Gauss points per element, depicted as crosses in the figure below. Alongside are the values for the Gauss points and weights used in `Toppie`.

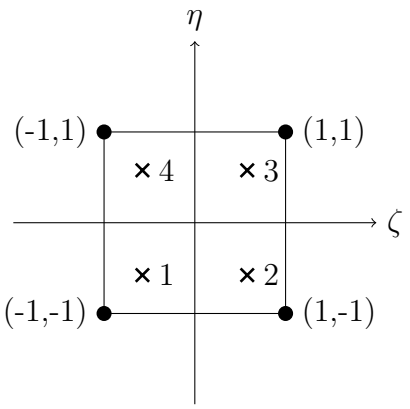

Figure 4.1: Square Element in Natural Coordinates (ζ, η) with Gauss Points

Gauss Point	ζ	η
1	-0.57735	-0.57735
2	0.57735	-0.57735
3	0.57735	0.57735
4	-0.57735	0.57735

Gauss Point	Weight
1	1
2	1
3	1
4	1

Table 4.1: Gauss Points and Weights

`Toppie`'s `elemental_stiffness_matrix` function, with the help of various private functions in the `Element` class, computes the elemental stiffness matrix by equivalently solving the following in code:

$$\boldsymbol{K_e} = t \sum_{i=1}^{2} \sum_{j=1}^{2} \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \det(\mathbf{J})\, w_i\, w_j \tag{4.4}$$

An algorithm is provided below that outlines the steps taken to compute the elemental stiffness matrix.

Algorithm 1 Computation of the Elemental Stiffness Matrix

1: **Input:** Constitutive stiffness matrix $\bar{\boldsymbol{Q}}$, thickness of element $t = 1$
2: **Initialize:** Elemental stiffness matrix $\boldsymbol{K_e} = \mathbf{0}$
3: **Constants:** Gauss Points, Gauss Weights, Natural Coordinate Nodes
4: **for** each Gauss Point (ζ, η) **do**
5: **Compute:** Shape function derivatives $\frac{\partial N}{\partial \zeta}$ and $\frac{\partial N}{\partial \eta}$
6: **Compute:** Jacobian $\mathbf{J}$ and its determinant $\det(\mathbf{J})$
7: **Compute:** Strain-Displacement matrix $\mathbf{B}$
8: **Update:** $\boldsymbol{K_e} +=$ Gauss Weight $\cdot \det(\mathbf{J}) \cdot t \cdot (\mathbf{B}^T \bar{\boldsymbol{Q}} \mathbf{B})$
9: **end for**
10: **Output:** Elemental stiffness matrix $\boldsymbol{K_e}$

Section C.2 of Appendix C contains the code associated with the `Element` class.

4.4 Problem Class

The `Problem` class defines the design domain and boundary conditions. `Toppie` can model several beam problems, including the half and full Messerschmitt-Bölkow-Blohm (MBB) beams, and two cantilever beam variants.

4.4.1 Element Degrees of Freedom

`Toppie` uses square elements similar to the one shown in the figure below.

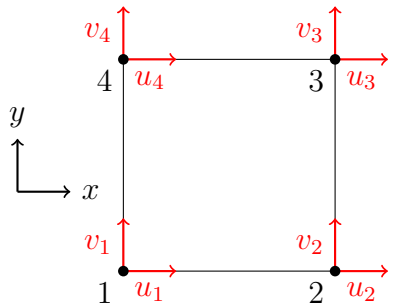

Figure 4.2: Square Element with Nodal Displacements

Each element has four nodes, with two degrees of freedom (DOF) per node, in the x and y directions respectively. These DOFs can be understood as local to the element, i.e. they describe the degrees of motion of the element itself. Many of these elements make up the entire mesh of a problem domain, similar to the depiction in Figure 4.3 below. Therefore, the local DOFs of each element need to be mapped to the global DOFs of the entire design domain. This implies that each DOF of the entire domain will have a unique index or identifier. This is crucial for the assembly of the global stiffness matrix, as will be explained in Section 4.5.1.

To visualise the numbering[1] of the DOFs within the design domain, consider the following figure. Figure 4.3 shows a grid of 4 elements in the x-direction (`X = 4`) and 3 elements in the y-direction (`Y = 3`), where `X` and `Y` are the variable names used by `Toppie`.

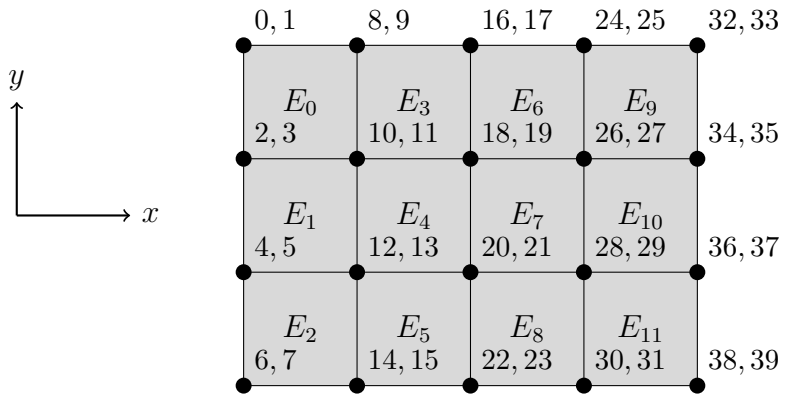

Figure 4.3: Degree of Freedom (DOF) Numbering

[1] Elements and nodes are numbered starting at 0, as is the convention in `Python`.

In the figure above, each node has two numbers alongside it. These numbers represent the DOFs associated with a particular node in the x and y directions respectively. Notice that each nodal DOF will have a unique number associated with it. The assignment of a unique number to each nodal DOF ensures that the system has a way of distinguishing each independent motion, whether in the x or y direction.

`Toppie` achieves this numbering through the use of a function called `initialise_elements`, which is part of the `Problem` class. `initialise_elements` takes the number of elements in the x and y directions as user-specified inputs and returns the element degrees of freedom matrix, `edofMat`. `edofMat` is a 2D array whose rows each represent one of the elements in the design domain and whose columns contain the associated DOFs for that element. It is thus an array that is `n_el` $\times$ 8 in size, where `n_el` is the total number of elements in the design domain, depicted below for clarity.

$$\texttt{edofMat} = \begin{bmatrix} \text{DOF}_{111} & \text{DOF}_{112} & \cdots & \text{DOF}_{141} & \text{DOF}_{142} \\ \text{DOF}_{211} & \text{DOF}_{212} & \cdots & \text{DOF}_{241} & \text{DOF}_{242} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \text{DOF}_{n11} & \text{DOF}_{n12} & \cdots & \text{DOF}_{n41} & \text{DOF}_{n42} \end{bmatrix}$$

In this representation, DOF_{ijk} is the degree of freedom (DOF) for the i-th element, j-th node, and k-th DOF (either 1 for x-direction or 2 for y-direction). The number of elements in the mesh is represented as n in the above representation, or equivalently `n_el` in code. Since DOFs are shared between neighbouring elements many entries in the `edofMat` array will be repeated.

Considering Figure 4.3 again, the `edofMat` array for this example is presented below. Notice that some entries are the same, indicating that the DOFs are shared between elements.

$$\texttt{edofMat}_{4\times 3} = \begin{bmatrix} 0 & 1 & 8 & 9 & 10 & 11 & 2 & 3 \\ 2 & 3 & 10 & 11 & 12 & 13 & 4 & 5 \\ 4 & 5 & 12 & 13 & 14 & 15 & 6 & 7 \\ 8 & 9 & 16 & 17 & 18 & 19 & 10 & 11 \\ 10 & 11 & 18 & 19 & 20 & 21 & 12 & 13 \\ 12 & 13 & 20 & 21 & 22 & 23 & 14 & 15 \\ 16 & 17 & 24 & 25 & 26 & 27 & 18 & 19 \\ 18 & 19 & 26 & 27 & 28 & 29 & 20 & 21 \\ 20 & 21 & 28 & 29 & 30 & 31 & 22 & 23 \\ 24 & 25 & 32 & 33 & 34 & 35 & 26 & 27 \\ 26 & 27 & 34 & 35 & 36 & 37 & 28 & 29 \\ 28 & 29 & 36 & 37 & 38 & 39 & 30 & 31 \end{bmatrix}$$

For more details on how the `edofMat` array is computed using `Toppie`, the reader is referred to Section C.3 of Appendix C.

4.4.2 Boundary Conditions and Loads

The definition of a problem domain in `Toppie` is best understood through the use of an example. Consider Figure 4.4, which represents a half MBB beam problem.

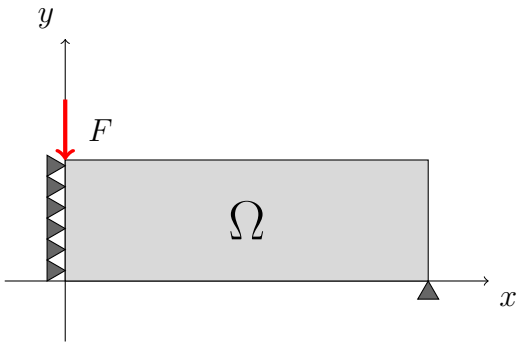

Figure 4.4: Half MBB Beam Example

To define this problem in `Toppie`, one has to fix the nodes on the far left of the design domain, as well as the bottom right node, represented as blue dots in Figure 4.5 below. To determine the DOFs that are free to move, first identify the fixed DOFs based on the desired problem domain. Once the nodal indices for these fixed DOFs are known, the free DOFs can be easily ascertained. The indices for the free DOFs are determined by taking the set difference between the complete set of all DOFs and the set of fixed DOFs. This is achieved through the use of a function called `compute_freedofs` contained in the `Problem` class.

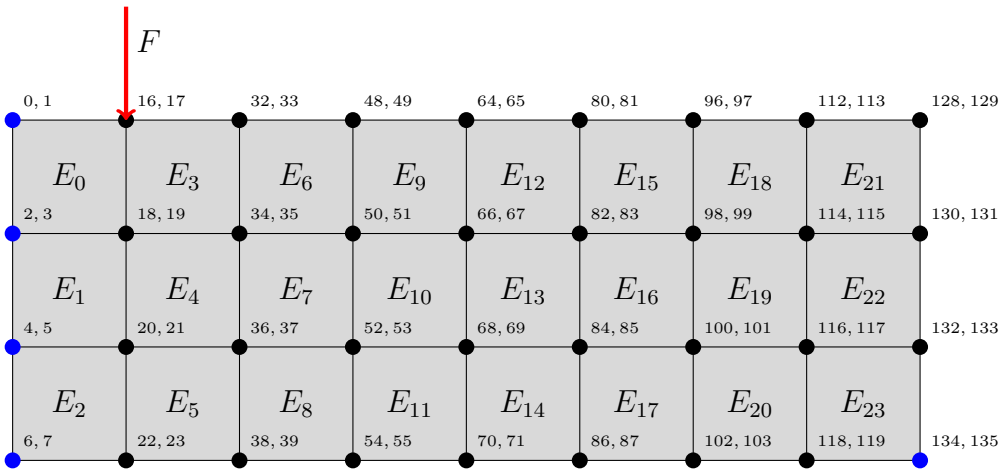

Figure 4.5: Half MBB Beam with Node Numbering

One should notice that in Figure 4.5 the force is applied at the node just to the right of the top left node, which differs from the problem domain in Figure 4.4. If it isn't clear to the reader why this is, it's important to understand that one can not logically say that a node is both immovable (due to constraint) and movable (due to applied force) at the same time. Therefore, the force is applied at the node just to the right of the top left node, which is the closest node that is free to move. Naturally, as the mesh size increases, the force will be applied closer to the top left node and the approximation of the problem domain will be improved.

4.5 FEAnalysis Class

The `FEAnalysis` class is responsible for assembling the global stiffness matrix, $\boldsymbol{K}$, and computing the nodal displacements, $\boldsymbol{u}$.

4.5.1 Assembly Global Stiffness Matrix

The elemental stiffness matrix, $\boldsymbol{K_e}$, is an important parameter needed to assemble the global stiffness matrix, $\boldsymbol{K}$. Therefore, it is necessary to consider the elemental stiffness matrix once more. $\boldsymbol{K_e}$ is computed using Gaussian Quadrature, as mentioned in Section 4.3, and results in an 2D array or equivalently a matrix of the following form:

$$\boldsymbol{K_e} = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{18} \\ k_{21} & k_{22} & \cdots & k_{28} \\ \vdots & \vdots & \ddots & \vdots \\ k_{81} & k_{82} & \cdots & k_{88} \end{bmatrix} \tag{4.5}$$

$\boldsymbol{K_e}$ is an 8×8 matrix where each row and column represent a unique nodal DOF of an element. For example, referring the element in Figure 4.2 from before, the DOF associated with node 1 in the horizontal direction, u_1, is represented by the first row and column of $\boldsymbol{K_e}$, i.e. k_{11}. Diagonal elements k_{ii} represent the stiffness of a DOF i against itself. For example, k_{11} would represent how much u_1 resists a unit displacement in the horizontal direction. The off-diagonal elements k_{ij} represent the coupling of displacements. If k_{ij} is non-zero, it would indicate a level of coupling between nodal displacements i and j.

From Section 4.4.1, the assemble of the global stiffness matrix was described by the following:

$$\boldsymbol{K}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^p \boldsymbol{K_e} \tag{4.6}$$

This is quite an abstract representation of the assembly process, so consider the following design domain that consists of only two elements, as a simplified example:

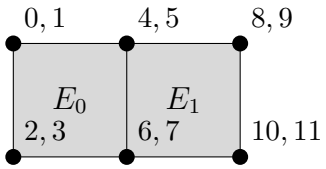

Figure 4.6: Design Domain with Two Elements

Here, the number of nodal displacements is 12 and not 16, because the elements are connected. The global stiffness matrix, $\boldsymbol{K}$, is thus a 12×12 matrix. Recall from Section 3.1 that the elemental stiffness matrix, $\boldsymbol{K_e}$, will be the same for both elements when considering homogeneous materials. Each element will have 8 local DOFs which need to be mapped to the corresponding global DOFs as described by the `edofMat` 2D array, in order to assemble the global stiffness matrix. The entries from the elemental stiffness matrix described by Expression 4.5 will be mapped to the global stiffness matrix as follows:

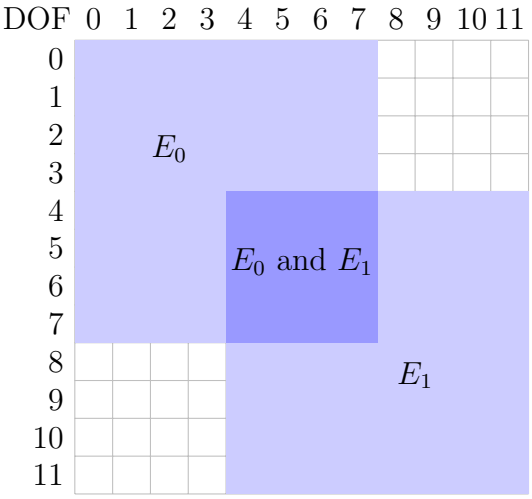

Figure 4.7: Global Stiffness Matrix Visual Representation

In Figure 4.7, one should notice that the entries associated with DOF 4,5,6 and 7 are shared between the elements. This shows where a coupling between the elements occurs. Th DOFs in the figure above start at 0, as is the convention in `Python`.

4.5.2 Compute Nodal Displacements

The nodal displacements, $\boldsymbol{u}$, are important parameters to compute as they are involved in the computation of the element sensitivities, which govern how the design updates. To solve for $\boldsymbol{u}$, the following equation is solved:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{F} \tag{4.7}$$

where $\boldsymbol{K}$ is the global stiffness matrix and $\boldsymbol{F}$ is the force vector. During the development of `Toppie`, several solvers were tested to compute the nodal displacements.

Table 4.2 compared the total computation time using different direct and iterative solvers for a half MBB beam of 60 x 20 elements. It is clear from this table that the iterative solvers are much slower than the direct solvers.

Table 4.2: Comparison of Solver Computational Times

Solver Type	Total Computation Time (s)
Iterative Solvers	
BiCGSTAB	84.78
QMR (Quasi-Minimal Residual)	74.02
LGMRES	67.53
CG (Conjugate Gradient)	51.89
Direct Solvers	
cholmod	1.28
SuperLU	1.27
SPSOLVE	1.04

To further investigate the performance of the direct solvers, the same comparison was done for a larger half MBB beam of 120 x 40 elements. Table 4.3 indicates that the `cholmod` solver was the fastest. Therefore, `cholmod` was chosen as the solver for `Toppie`.

Table 4.3: Direct Solver Comparison

Solver Type	Total Computation Time (s)
cholmod	6.66
SuperLU	17.36
SPSOLVE	16.70

For further details on the implementation of the `cholmod` solver in `Toppie`, the reader is referred to Section C.5 of Appendix C.

4.6 Optimisation Class

The `Optimisation` class is responsible for the sensitivity analysis, sensitivity filtering and design update steps used by `Toppie`

4.6.1 Sensitivity Analysis

In Section 3.2.2, the mathematical theory behind sensitivity analysis was provided. While the mathematical formulation is important to understand the sensitivity analysis function implemented in `Toppie`, it might be useful to the reader to consider the following example, which will hopefully provide the reader with a more intuitive understanding of sensitivity analysis.

Figures 4.8, 4.9 and 4.10 depict topology and sensitivity maps for a half MBB problem with a 240×80 mesh at the 10^{th}, 20^{th}, and 100^{th} iteration respectively.

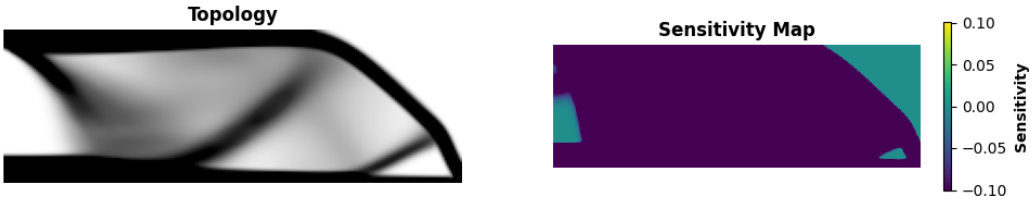

Figure 4.8: Topology and Sensitivity at the 10^{th} Iteration

Figure 4.9: Topology and Sensitivity at the 20^{th} Iteration

Figure 4.10: Topology and Sensitivity at the 100^{th} Iteration

In Figure 4.8, the sensitivity map has high sensitivities, depicted in purple, in the majority of the design domain. At this stage, the optimisation algorithm has identified numerous elements in the design domain where an increase or decrease in the density of these elements will minimize the objective function. This is evident in the accompanying topology plot, where areas with partial density are depicted in grey.

As the iterations increase, to 20 and 100 iterations, in Figure 4.9 and 4.10 respectively, the sensitivity map starts to resemble the material distribution in the topology plot more closely. The high-sensitivity regions (purple) in the sensitivity maps associated with these figures coincide with regions where material is distributed in the topology plots, while low-sensitivity regions (teal) coincide with areas where material has been removed.

The resemblance of the sensitivity map to the topology map is not surprising. After all, the sensitivities are directly influencing the removable and addition of material in the design domain. At higher iterations, like the plot in Figure 4.10, the sensitivity map and the topology map look very similar, indicating that the algorithm is less uncertain about where to add or remove material for the most optimal design.

4.6.2 Sensitivity Filtering

A mathematical formulation for the filtering technique was provided in Section 3.2.3. While this is important to understand if one wishes to implement a filtering function, it may not be entirely clear to the reader the purpose of filtering and why it is used. Therefore, consider the following two figures. Figure 4.11 shows the checkerboarding effect that occurs when the optimisation problem is solved without filtering:

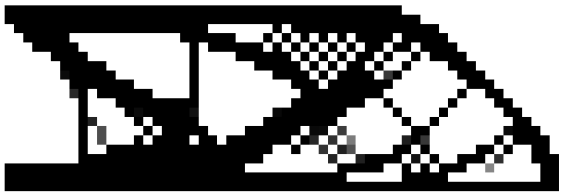

Figure 4.11: Checkerboarding Effect for Half MBB Beam

Figure 4.11 shows that the topology has regions of alternating solid and void elements. A filtering function called `filt` is used in `Toppie`, which eliminates this checkerboarding effect, as evidenced by Figure 4.12:

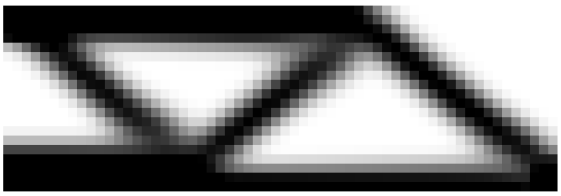

Figure 4.12: Filtered Topology for Half MBB Beam

In Section 3.2.3, the unfiltered sensitivity of one element, i, was denoted as $\frac{\partial C}{\partial x_i}$. In reality, every element in the design domain will have an associated sensitivity contribution. All these sensitivities are collectively stored in an array called `dc` in `Toppie`. `dc` is equivalent to the mathematical notation ∇C:

$$\nabla C = \left[\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_n}\right]$$

The algorithm below describes the implementation of this filtering technique in `Toppie`.

Algorithm 2 The `filt` function

1: **Input:** Density distribution $\boldsymbol{x}$, minimum radius $r_{\min}$, unfiltered sensitivities ∇C
2: **Create Kernel:** Generate a normalized convolution kernel, H_{ij}, based on $r_{\min}$
3: **Weighted Sensitivities:** Compute $\nabla C_{\boldsymbol{x}} = \nabla C \odot \boldsymbol{x}$
4: **Apply Convolution:** $\nabla \tilde{C}_{\boldsymbol{x}} = \nabla C_{\boldsymbol{x}} \circledast H_{ij}$
5: **Normalize:** Compute $\nabla \tilde{C} = \nabla \tilde{C}_{\boldsymbol{x}} / \boldsymbol{x}$
6: **Output:** Return $\nabla \tilde{C}$ as the filtered sensitivities

4.6.3 The OC Method and the Lagrange Multiplier

In Section 3.2.1, the volume constraint was introduced. To satisfy this constraint, while optimising the objective function, the optimisation problem was formulated as a Lagrangian problem with the use of a Lagrange multiplier, λ.

After the sensitivities have been computed and filtered, the design can be updated using the OC method, as described in Section 3.2.4. The OC method depends on the Lagrange multiplier: recall that the scaling factor, B_i, is dependent on λ. This implies that at each iteration an optimal value for λ needs to be found such that the objective is minimised while ensuring the constraint is as close to being satisfied as possible. In other words, the volume of the structure, $V(\boldsymbol{x})$, is as close to the maximum permittable volume, $V_{\max}$.

`Toppie` makes uses a simple bisection algorithm to find the values of λ (`lmid`). Figure 4.13 below shows how `lmid` approaches a minimum value as the iterations increase to convergence, for a half MBB problem with a 60×20 mesh.

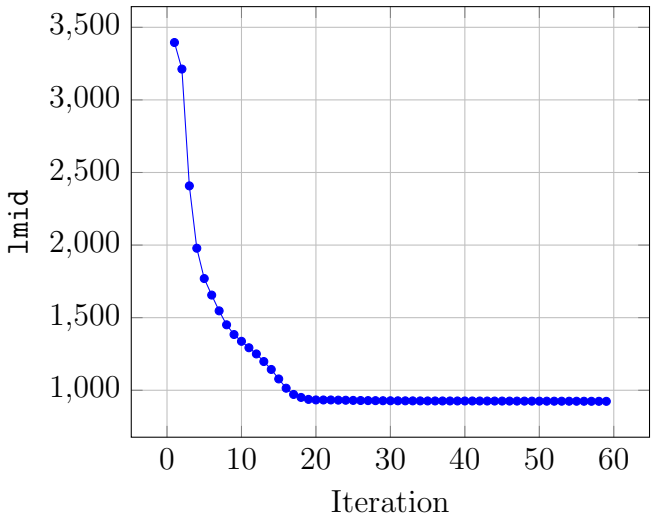

Figure 4.13: Convergence of `lmid` Value Across Iterations.

The figure above shows that as the optimiser approaches convergence, the value of `lmid` decreases and then stabilises. This indicates that the algorithm is less "conflicted" between the objective function and the volume constraint and could give a level of convergence for the model. In other words, the stabilised `lmid` value indicates that the model has found a design that is both structurally efficient (low compliance) and meets the volume constraint.

The algorithm below shows how `lmid` (λ) is used when updating the design in `Toppie`.

Algorithm 3 Design Update Algorithm

Input: Number of elements X, Y, density distribution $\boldsymbol{x}$, volume fraction f, filtered sensitivities $\tilde{\nabla C}$
Initialize: $\lambda_1 = 0$, $\lambda_2 = 100000$, $\eta = 0.5$
while $(\lambda_2 - \lambda_1) > 10^{-4}$ **do**
    $\lambda = 0.5 \cdot (\lambda_2 + \lambda_1)$
    Update $\boldsymbol{x}_{\text{new}}$ using the OC method as described in Section 3.2.4
    if $\sum \mathbf{x}_{\text{new}} - f \cdot X \cdot Y > 0$ **then**
        $\lambda_1 = \lambda$
    else
        $\lambda_2 = \lambda$
    end if
end while
Output: Return $\boldsymbol{x}_{\text{new}}$ as the new density distribution

Chapter 5

ON TOPOLOGY OPTIMISATION
Results and Discussion

"The separation between the natural and artificial worlds is as artificial as the values supporting this worldview."
- Neri Oxman

This chapter presents some results obtained from the testing of `Toppie`. Firstly, `Toppie` is validated with a known benchmark problem. Secondly, the full MBB beam problem is solved. Finally, the cantilever beam problem is solved for various fibre angles, investigating the effect that differing fibre angles have on the converged topology and compliance.

5.1 Validation with the Half MMB Benchmark Problem

To test the validity of `Toppie`'s implementation of the SIMP method, a standardised half MBB problem for an isotropic material is considered. An example from Morten Nobel-Jørgensen's thesis was used as a benchmark problem [36]. In this example, the half MBB beam is discretised into 120×40 elements, as shown in Figure 5.1.

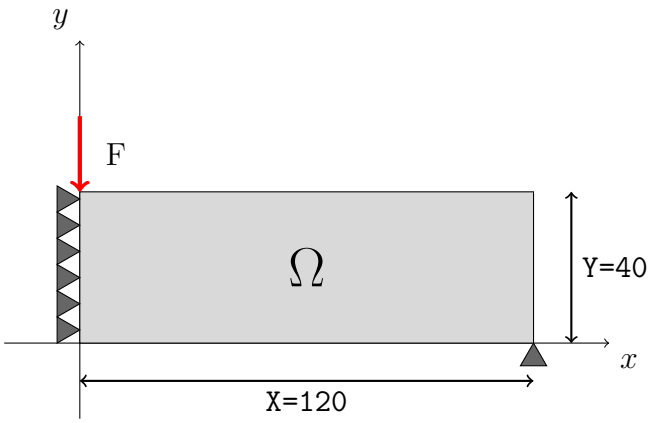

Figure 5.1: Half MBB Beam Benchmark Problem

A volume fraction (`f`), penalisation factor (`p`), and filter radius (`rmin`) of 0.5, 3, and 3 respectively were used. The material properties were defined as `E=1` and `v=0.3`. `Toppie` ran for 100 iterations and the results are shown in Figure 5.2 below.

Figure 5.2: Half MBB Beam Converged Topology

Figure 5.2 coincides with the results obtained by Nobel-Jørgensen in his thesis [36]. This confirms that `Toppie` has implemented the SIMP method for isotropic materials successfully.

5.2 The Full MBB Beam

`Toppie` is also capable of solving the full MBB beam problem. A mesh size of 400 by 100 elements was used, with a volume fraction of 0.5, a penalisation factor of 3, and a filter radius of 1.5. The material properties were defined as `E=1` and `v=0.3`. An example of the full MBB beam problem is shown in Figure 5.3 below.

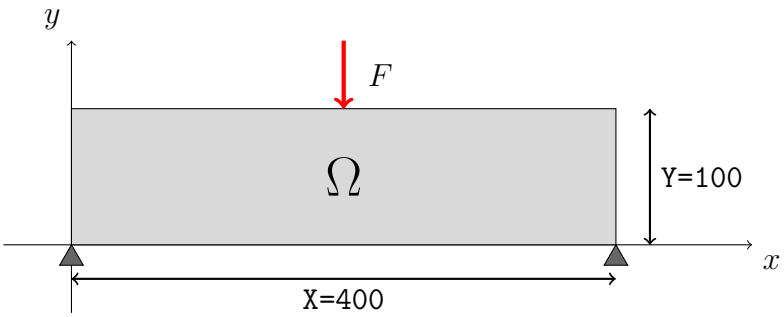

Figure 5.3: Full MBB Beam Problem

`Toppie` ran for 308 iterations with a maximum change in density of all design variables being less than 0.01 as the convergence criteria.

Figure 5.4: Full MBB Beam Converged Topology

In Figure 5.4, the material has been placed in the optimal locations that would maximise the stiffness of the beam, as is the objective. The location of the force is significant in that this is where the greatest bending stresses are experienced in the beam. In the figure, the material is placed at the location of this force and is distributed along paths that would help distribute the stresses induced by the point load. These paths are known as load paths and are investigated further in the following section.

5.3 Orthotropic Implementation - Cantilever Beam Problem

To investigate the effect that generally orthotropic laminae have on the converged topology at different fibre angles, `Toppie` has been tested using the material properties for a glass/epoxy composite material at several fibre angles. A cantilever beam problem was used as the problem domain, depicted in Figure 5.5:

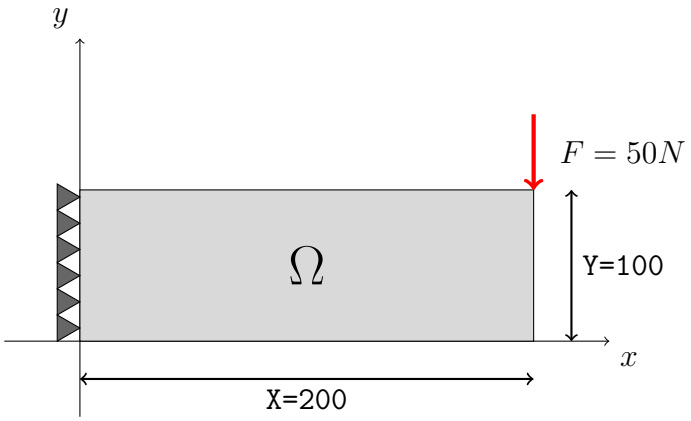

Figure 5.5: Cantilever Beam Problem

A 200×100 mesh grid was used, with a penalisation factor of 3, a volume fraction of 0.5, and a filter radius of 3. The glass/epoxy material properties that were used are taken from Table 2.1 of Autar K. Kaw's textbook, Mechanics of Composite Materials [26]. The material properties are as follows: a longitudinal elastic modulus $E_1 = 36$ GPa, a transverse elastic modulus $E_2 = 8.27$ GPa, a major Poisson's ratio $v_{12} = 0.26$, and a shear modulus $G_{12} = 4.14$ GPa. A point load of 50 N was applied downwards at the top right node of the design domain and `Toppie` ran until a maximum change in density of all design variables was less than 0.01, indicating convergence.

The converged topologies for a fibre angle of $\pm 30°$, $\pm 45°$, $0°$, and $90°$ are presented in the figures below.

(a) $-30°$ (b) $30°$

Figure 5.6: Optimal Material Distribution at $\pm 30°$ Fibre Angles

(a) $-45°$ (b) $45°$

Figure 5.7: Optimal Material Distribution at $\pm 45°$ Fibre Angles

(a) $0°$ (b) $90°$

Figure 5.8: Optimal Material Distribution at $0°$ and $90°$ Fibre Angles

From the figures above, it is evident that the "load-carrying strut", referring to the primary structural member supporting the majority of the applied load, is aligned with the fibre direction. For example, for negative angles in Figure 5.6a and 5.7a, the "load-carrying strut" is aligned with the $-30°$ and $-45°$ degree angles relative to the global x-axis.

In Figure 5.6b, the "load-carrying member" is roughly $30°$ relative to the x-axis of the global domain. Similarly, in Figure 5.7b, this strut appears to be $45°$ relative to the x-axis. In Figure 5.8, the topologies for a fibre angle of $0°$ and $90°$ are presented. In these cases, it is not as clear that the "load-carrying strut" is aligned with the fibre direction. There appears to be a compromise between the material distribution along the fibre direction and the distribution in other directions. The exploitation of the anisotropic behaviour of the orthotropic material is not as evident in these cases.

This phenomenon is consistent with composite theory in that unidirectional fibres provide the highest stiffness and strength along their primary axes, as highlighted in Section 2.4. Essentially, the optimisation algorithm exploits the anisotropic behaviour of the orthotropic material by disturbing material along these fibre angles, improving the distribution of the load along these paths, and thus maximising the stiffness.

5.4 Orthotropic Implementation - Compliance Trend

To investigate the effect that different fibre angles have on a structure's compliance at convergence, `Toppie` was tested with the same glass/epoxy composite material properties as before, except this time at a reduced mesh size of just 80×30 elements. Simulations were run for fibre angles from $-90°$ to $90°$, in $10°$ increments, until a convergence tolerance of 0.01 was achieved. The results are presented in the figure below.

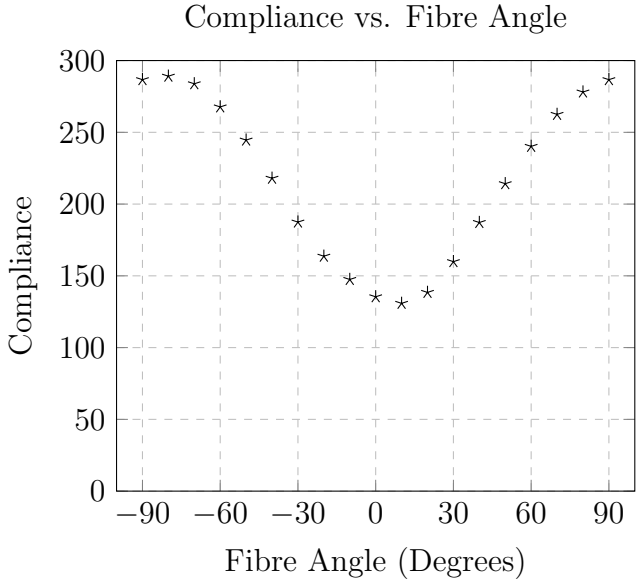

Figure 5.9: Relationship Between Fibre Angle and Compliance

The results in Figure 5.9 indicate that the lowest compliance occurred at a fibre angle of $10°$, meaning the fibres were almost parallel with the x-axis of the design domain. Compliance was greater when the fibres were more vertical relative to the x-axis, and achieved a maximum value when the fibres were aligned parallel to the y-axis of the design domain, as evidenced by the greatest compliance at fibre angles of $-90°$ and $90°$ respectively.

To interpret these results, consider the stresses in the cantilever beam, depicted in Figure 5.10 below. When a point load is applied at the right end of the beam, internal stresses are generated in the beam. These stresses are shear and bending stresses, where shear stress acts vertically downward across a slice or cross-section of the beam while bending stresses are a result of tensile and compressive forces acting laterally along the direction of the beam.

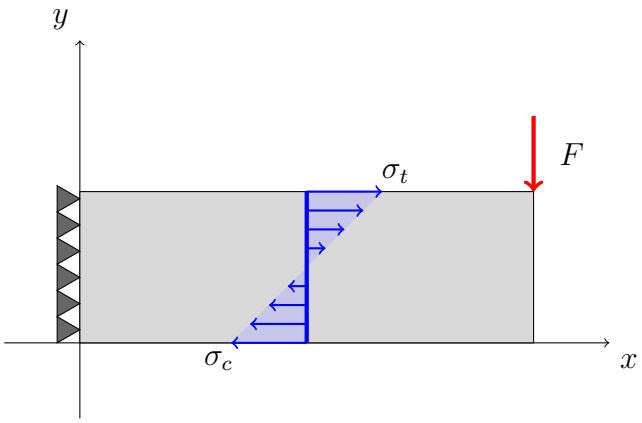

Figure 5.10: Cantilever Beam with Stress Distribution

Section 2.4 introduced the concept of composite materials composed of unidirectional fibres within a matrix. It was outlined that the fibres in a composite material are generally much stiffer and stronger than the matrix material. This becomes significant in the context of the cantilever beam problem as the alignment of fibres in the beam can significantly affect the compliance and thus stiffness of the beam, as evidenced by the results in Figure 5.9.

When the fibres are aligned parallel to the x-axis, they are aligned with the direction of the compressive (σ_c) and tensile (σ_c) stresses in the beam. As a result, the fibres are able to resist these stresses more effectively than when the fibres are aligned parallel to the y-axis. This is evidenced by the lowest compliance at a fibre angle of 10°.

Conversely, when the fibres are parallel to the y-axis, they are aligned with the direction of the shear stress but are not effective in counteracting the primary tensile and compressive stresses caused by the bending moment. As a result, the compliance is highest when the fibres are vertically aligned, resulting in a structure with lower relative stiffness.

Chapter 6

CONCLUDING REMARKS
A Conclusion

"What we learn with pleasure, we never forget."

- Alfred Mercier

This document serves as an educational resource for Mechanical Engineering students, extending the applicability of the SIMP method to orthotropic materials. It builds upon educational works like Sigmund's `top99` [8] and Andreassen's `top88` [9], by offering a `Python`-based implementation geared specifically towards orthotropic materials. Furthermore, this work aims to present topology optimisation theory in a manner accessible to undergraduate students.

This work has realised its aim as an education resource for the following reasons. Chapter 2 focused on the SIMP method within the broader context of topology optimisation, offering students the necessary background to understand the methodology chapters that followed. A comprehensive mathematical framework, covering sensitivity analysis, filtering, and the optimality criteria method, was presented in Chapter 3. Chapter 4 provided the numerical details, supporting the theoretical framework underpinning the SIMP method. A `Python`-based program, `Toppie`, was developed to support this objective and is fully contained in Appendix C.

The results in Chapter 5 validated the correct implementation of `Toppie`, as evidenced by its consistency with existing literature. Importantly, the work revealed the significant influence of fibre angles in orthotropic materials on material distribution and structural compliance. It demonstrated that the material is strategically placed along the direction of the fibre angles to mitigate stresses, exploiting the anisotropic properties of the material.

This work is not without its limitations. Only 2D compliance minimisation problems were considered. Chapter 2 mentioned numerous other topology optimisation objectives. An extension to 3D problems, and other objectives, is a natural progression of this work. For example, enhancing `Toppie` to handle objectives related to structural deformity at natural frequencies could provide valuable insights into earthquake-resistant building designs. Such an extension would offer students an interdisciplinary learning experience, spanning computer methods, structural mechanics, and mechanics of vibrations.

Bibliography

[1] W. Forst and D. Hoffmann, *Optimization—theory and practice*. Springer Science & Business Media, 2010.

[2] C. Wang, Z. Zhao, M. Zhou, O. Sigmund, and X. S. Zhang, "A comprehensive review of educational articles on structural and multidisciplinary optimization," *Structural and Multidisciplinary Optimization*, pp. 1–54, 2021.

[3] M. P. Bendsøe and N. Kikuchi, "Generating optimal topologies in structural design using a homogenization method," *Computer methods in applied mechanics and engineering*, vol. 71, no. 2, pp. 197–224, 1988.

[4] J. Alexandersen, "A detailed introduction to density-based topology optimisation of fluid flow problems with implementation in matlab," *Structural and Multidisciplinary Optimization*, vol. 66, no. 1, p. 12, 2023.

[5] N. P. Van Dijk, K. Maute, M. Langelaar, and F. Van Keulen, "Level-set methods for structural topology optimization: a review," *Structural and Multidisciplinary Optimization*, vol. 48, pp. 437–472, 2013.

[6] G.-C. Luh, C.-Y. Lin, and Y.-S. Lin, "A binary particle swarm optimization for continuum structural topology optimization," *Applied Soft Computing*, vol. 11, no. 2, pp. 2833–2844, 2011.

[7] X. Huang and Y. Xie, "Optimal design of periodic structures using evolutionary topology optimization," *Structural and Multidisciplinary Optimization*, vol. 36, pp. 597–606, 2008.

[8] O. Sigmund, "Sigmund, o.: A 99 line topology optimization code written in matlab. structural and multidisciplinary optimization 21, 120-127," *Structural and Multidisciplinary Optimization*, vol. 21, pp. 120–127, 04 2001.

[9] E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund, "Efficient topology optimization in matlab using 88 lines of code," *Structural and Multidisciplinary Optimization*, vol. 43, pp. 1–16, 2011.

[10] M. P. Salaimanimagudam, C. R. Suribabu, G. Murali, and S. R. Abid, "Impact response of hammerhead pier fibrous concrete beams designed with topology optimization," *Periodica Polytechnica Civil Engineering*, vol. 64, no. 4, pp. 1244–1258, 2020.

[11] X. Huang, Z. Zuo, and Y. Xie, "Evolutionary topological optimization of vibrating continuum structures for natural frequencies," *Computers & structures*, vol. 88, no. 5-6, pp. 357–364, 2010.

[12] B. Özmen and M. Topaç, "Effect of damping rate on fatigue failure tendency of a topology-optimised swing arm for a heavy commercial truck cab suspension," *Engineering Failure Analysis*, vol. 137, p. 106276, 2022.

[13] H.-L. Ye, W.-W. Wang, N. Chen, and Y.-K. Sui, "Plate/shell structure topology optimization of orthotropic material for buckling problem based on independent continuous topological variables," *Acta Mechanica Sinica*, vol. 33, pp. 899–911, 2017.

[14] A. Gersborg-Hansen, M. P. Bendsøe, and O. Sigmund, "Topology optimization of heat conduction problems using the finite volume method," *Structural and multidisciplinary optimization*, vol. 31, pp. 251–259, 2006.

[15] J. Deng, J. Yan, and G. Cheng, "Multi-objective concurrent topology optimization of thermoelastic structures composed of homogeneous porous material," *Structural and Multidisciplinary Optimization*, vol. 47, pp. 583–597, 2013.

[16] G. Vantyghem, V. Boel, M. Steeman, and W. De Corte, "Multi-material topology optimization involving simultaneous structural and thermal analyses," *Structural and Multidisciplinary Optimization*, vol. 59, pp. 731–743, 2019.

[17] T. Borrvall and J. Petersson, "Topology optimization of fluids in stokes flow," *International journal for numerical methods in fluids*, vol. 41, no. 1, pp. 77–107, 2003.

[18] M. P. Bendsøe, "Optimal shape design as a material distribution problem," *Structural optimization*, vol. 1, pp. 193–202, 1989.

[19] G. I. Rozvany, M. Zhou, and T. Birker, "Generalized shape optimization without homogenization," *Structural optimization*, vol. 4, pp. 250–252, 1992.

[20] G. Rozvany, M. Bendsoe, and U. Kirsch, "Layout optimization of structures," 1995.

[21] G. I. Rozvany, "Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics," *Structural and Multidisciplinary optimization*, vol. 21, pp. 90–108, 2001.

[22] M. P. Bendsøe and O. Sigmund, "Material interpolation schemes in topology optimization," *Archive of applied mechanics*, vol. 69, pp. 635–654, 1999.

[23] K. Liu and A. Tovar, "An efficient 3d topology optimization code written in matlab," *Structural and Multidisciplinary Optimization*, vol. 50, pp. 1175–1196, 2014.

[24] N. D. Lagaros, N. Vasileiou, and G. Kazakis, "Ac# code for solving 3d topology optimization problems using sap2000," *Optimization and Engineering*, vol. 20, pp. 1–35, 2019.

[25] T. Smit, N. Aage, S. J. Ferguson, and B. Helgason, "Topology optimization using petsc: a python wrapper and extended functionality," *Structural and Multidisciplinary Optimization*, vol. 64, pp. 4343–4353, 2021.

[26] A. K. Kaw, *Mechanics of composite materials*. CRC press, 2005.

[27] T. W. Clyne and D. Hull, *An introduction to composite materials*. Cambridge university press, 2019.

[28] S. Li and S. Atluri, "The mlpg mixed collocation method for material orientation and topology optimization of anisotropic solids and structures," *CMES: Computer Modeling in Engineering & Sciences*, vol. 30, no. 1, pp. 37–56, 2008.

[29] R. Hoglund and D. E. Smith, "Non-isotropic material distribution topology optimization for fused deposition modeling products," in *2015 International Solid Freeform Fabrication Symposium*. University of Texas at Austin, 2015.

[30] D. Harvey and P. Hubert, "3d topology optimization of sandwich structures with anisotropic shells," *Composite Structures*, vol. 285, p. 115237, 2022.

[31] A. Shukla, A. Misra, and S. Kumar, "Checkerboard problem in finite element based topology optimization," *International Journal of Advances in Engineering & Technology*, vol. 6, no. 4, p. 1769, 2013.

[32] O. Sigmund and J. Petersson, "Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima," *Structural optimization*, vol. 16, pp. 68–75, 1998.

[33] O. Sigmund, "Design of material structures using topology optimization," Ph.D. dissertation, Technical University of Denmark Lyngby, 1994.

[34] B. Bourdin, "Filters in topology optimization," *International journal for numerical methods in engineering*, vol. 50, no. 9, pp. 2143–2158, 2001.

[35] M. P. Bendsøe, *Optimization of structural topology, shape, and material*. Springer, 1995, vol. 414.

[36] M. Nobel-Jørgensen, "Interactive topology optimization," pp. 36–37, 2016.

[37] I. Koutromanos, *Fundamentals of finite element analysis: linear finite element analysis*. John Wiley & Sons, 2018.

Appendix A

Theory of Solid Mechanics

A.1 Derivation of the Local Stiffness Matrix

The following section outlines the methodology for the formation of the local stiffness matrix of an orthotropic material. The derivation is based on Hooke's Law as it relates to two-dimensional unidirectional lamina [26]:

The stress-strain relationship for an orthotropic material in a 3D local coordinate system and whose principle axes are 1,2 and 3 respectively is given by:

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{31} \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{13} & C_{23} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \gamma_{23} \\ \gamma_{21} \\ \gamma_{12} \end{bmatrix} \tag{A.1}$$

Under plane stress conditions, $\sigma_3 = \tau_{31} = \tau_{32} = 0$. The stress-strain relationship simplifies to:

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & 0 \\ C_{12} & C_{22} & 0 \\ 0 & 0 & C_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{bmatrix} \tag{A.2}$$

The terms C_{11}, C_{12}, C_{22}, and C_{66} can be related to the elastic moduli E_1, E_2, shear modulus G_{12}, and Poisson's ratios ν_{12}, ν_{21}. This relationship arrives by considering three different load cases: a pure tensile test with the load in direction 1, a pure tensile test with the load in direction 2 and finally applying a pure shear load in the 12 direction, while ensuring that the other elements in the stress vector remain zero, see [26] for more details. The following relationships are obtained:

$$
\begin{aligned}
\nu_{21} &= \frac{E_2}{E_1}\nu_{12} \\
C_{11} &= \frac{E_1}{1-\nu_{12}\nu_{21}} \\
C_{22} &= \frac{E_2}{1-\nu_{12}\nu_{21}} \\
C_{12} &= \frac{\nu_{12}E_1}{1-\nu_{12}\nu_{21}} = \frac{\nu_{21}E_2}{1-\nu_{12}\nu_{21}} \\
C_{66} &= G_{12}
\end{aligned}
$$

Substituting these into Equation A.2 yields:

$$
\boldsymbol{Q} = \begin{bmatrix} \frac{E_1}{1-\nu_{12}\nu_{21}} & \frac{\nu_{21}E_2}{1-\nu_{12}\nu_{21}} & 0 \\ \frac{\nu_{12}E_1}{1-\nu_{12}\nu_{21}} & \frac{E_2}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix} \tag{A.3}
$$

This represents the local stiffness matrix or reduced stiffness matrix for an orthotropic material. It is important to note that this is in the local elemental domain.

A.2 Derivation of the Global Stiffness Matrix

Having formulated the local stiffness matrix for a single orthotropic element, the next step is to transpose this element such that it aligns with the principle axes of the design domain. Consider the figure below of a generally orthotropic lamina, whose local (1-2) coordinate system is positioned at an angle θ with respect to the global (x-y) coordinate system.

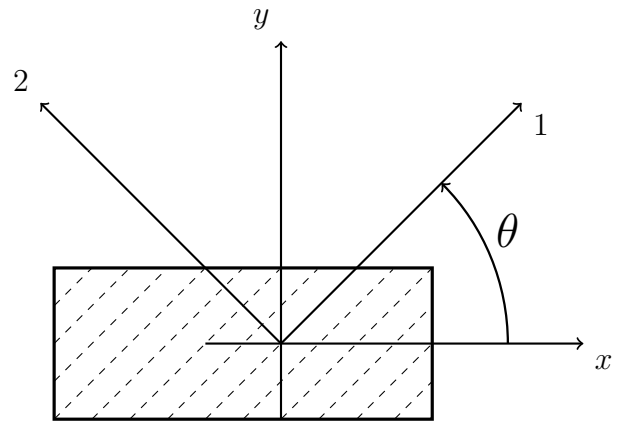

Figure A.1: Generally Orthotropic Angled Lamina

The transformation matrix $\boldsymbol{T}$ is derived based on the fibre orientation with respect to the global coordinate system. The plane stress transformation matrix is defined as:

$$\boldsymbol{T} = \begin{bmatrix} \cos^2(\theta) & \sin^2(\theta) & 2\sin(\theta)\cos(\theta) \\ \sin^2(\theta) & \cos^2(\theta) & -2\sin(\theta)\cos(\theta) \\ -\sin(\theta)\cos(\theta) & \sin(\theta)\cos(\theta) & \cos^2(\theta) - \sin^2(\theta) \end{bmatrix} \tag{A.4}$$

The Rueter matrix $\boldsymbol{R}$ is used to correctly account for engineering shear strain, which is twice the tensorial definition.[1] The Rueter matrix is defined as:

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

In "Mechanics of Composite Materials" by Kaw [26], the global stiffness matrix is defined using the transformation matrix and the Rueter matrix as follows:

$$\bar{\boldsymbol{Q}} = \boldsymbol{T}^T \cdot \boldsymbol{Q} \cdot \boldsymbol{R} \cdot \boldsymbol{T} \cdot \boldsymbol{R}^T$$

[1] In tensor notation, the shear strain is half that of the engineering strain. This means that under shear deformation, a rectangular element will deform into a parallelogram by an angle of $\gamma/2$. In topology optimization, the tensorial notation is not sufficient to represent the practical deformations experienced by materials in real-world applications. The engineering notation, which accounts for the full angular deformation of γ, provides a more accurate and practical representation of the material behaviour. Hence, the Rueter matrix accordingly accounts for this difference in shear-strain notation, ensuring that the derived stiffness matrix is consistent with real-world engineering scenarios.

A.3 Derivation of the Elemental Stiffness Matrix

The following derivation is based on the methodologies outlined in [37].

To find the elemental stiffness matrix $\boldsymbol{K_e}$, one has to perform a double integral with respect to the global coordinate system, as shown in the following equation:

$$\boldsymbol{K_e} = t \iint_e \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \, dx \, dy \tag{A.5}$$

where t is the thickness of the element, $\boldsymbol{B}$ is the strain-displacement matrix and $\bar{\boldsymbol{Q}}$ is the reduced stiffness matrix in the global domain.

This can be challenging because the shape of the element in the global coordinate system may not be a square and as such integrating over the element becomes difficult. To overcome this, one can perform the integration in the natural coordinate system, whose element is defined by the coordinates (-1,-1), (1,-1), (1,1), (-1,1). Integrating over these coordinates is much easier because the element is a square, and has integration limits that readily lend themselves to the use of Gaussian Quadrature, the numerical integration technique used in this project.

Consider the figure below, which shows a quadrilateral element in the global coordinate system and the corresponding square element in the natural coordinate system.[2] A mapping from the global coordinate system to the natural coordinate system needs to take place in order to exploit the integration limits of the natural coordinate system.

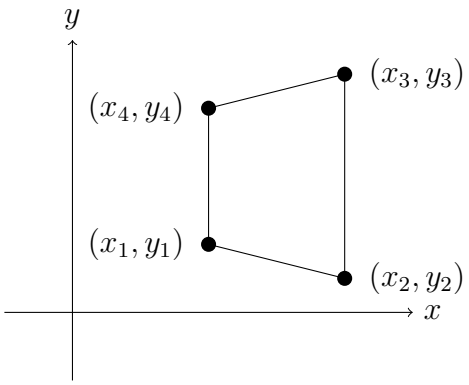

Figure A.2: Generalized Isoparametric Quadrilateral Element in the Global Coordinate System (x, y)

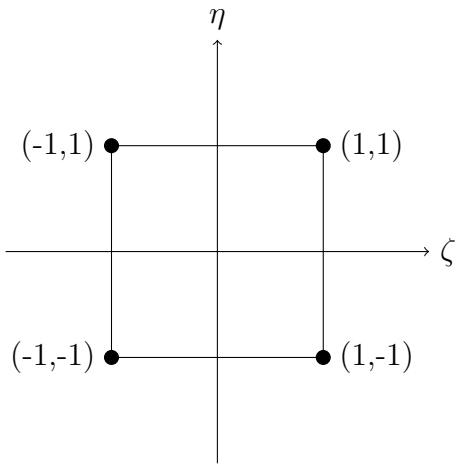

Figure A.3: Square Element in Natural Coordinates (ζ, η)

[2]The elements used in this project are square elements. However, the derivation of the elemental stiffness matrix is the same for all element types. For this derivation, a quadrilateral element is used.

To map between the global coordinate system and the natural coordinate system, one needs to derive a mapping function between these two systems. This mapping function can be expressed as $\begin{bmatrix} x \\ y \end{bmatrix}$ and is given by:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} N_1^{4Q} & 0 & N_2^{4Q} & 0 & N_3^{4Q} & 0 & N_4^{4Q} & 0 \\ 0 & N_1^{4Q} & 0 & N_2^{4Q} & 0 & N_3^{4Q} & 0 & N_4^{4Q} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} \tag{A.6}$$

where N_1, N_2, N_3 and N_4 are the shape functions associated with the four nodes of the element.

By definition, shape functions evaluate to 1 at the node they are associated with and 0 at all other nodes within an element. To illustrate this, consider the shape function for node 1, N_1, in the figure below, which is located in the natural coordinate system. It should be clear that N_1 evaluates to 1 at node 1 and has a smooth transition to 0 at all other nodes.

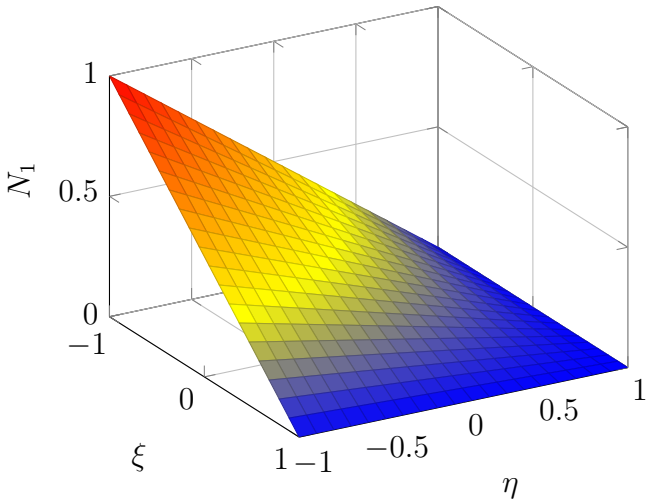

Figure A.4: Shape function behaviour of node 1 in the natural coordinate system.

The shape functions for all four nodes are given below. The superscript $4Q$ indicates that these shape functions are for a 4-node quadrilateral element.

$$N_1^{4Q}(\zeta, \eta) = \frac{1}{4}(1-\zeta)(1-\eta), \tag{A.7}$$

$$N_2^{4Q}(\zeta, \eta) = \frac{1}{4}(1+\zeta)(1-\eta), \tag{A.8}$$

$$N_3^{4Q}(\zeta, \eta) = \frac{1}{4}(1+\zeta)(1+\eta), \tag{A.9}$$

$$N_4^{4Q}(\zeta, \eta) = \frac{1}{4}(1-\zeta)(1+\eta). \tag{A.10}$$

Referring to Figure A.2 and A.3, it is clear that the shape of the element in the global coordinate system differs from that in the natural coordinate system. To account for the difference in shape, the Jacobian matrix, **J**, is employed:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \zeta} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \zeta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{A.11}$$

From the matrix mapping in Equation A.6, the partial derivatives of x and y with respect to ζ and η can be shown to be:

$$\begin{aligned}
\frac{\partial x}{\partial \zeta} &= \frac{\partial N_1^{4Q}}{\partial \zeta} x_1 + \frac{\partial N_2^{4Q}}{\partial \zeta} x_2 + \frac{\partial N_3^{4Q}}{\partial \zeta} x_3 + \frac{\partial N_4^{4Q}}{\partial \zeta} x_4, \\
\frac{\partial x}{\partial \eta} &= \frac{\partial N_1^{4Q}}{\partial \eta} x_1 + \frac{\partial N_2^{4Q}}{\partial \eta} x_2 + \frac{\partial N_3^{4Q}}{\partial \eta} x_3 + \frac{\partial N_4^{4Q}}{\partial \eta} x_4, \\
\frac{\partial y}{\partial \zeta} &= \frac{\partial N_1^{4Q}}{\partial \zeta} y_1 + \frac{\partial N_2^{4Q}}{\partial \zeta} y_2 + \frac{\partial N_3^{4Q}}{\partial \zeta} y_3 + \frac{\partial N_4^{4Q}}{\partial \zeta} y_4, \\
\frac{\partial y}{\partial \eta} &= \frac{\partial N_1^{4Q}}{\partial \eta} y_1 + \frac{\partial N_2^{4Q}}{\partial \eta} y_2 + \frac{\partial N_3^{4Q}}{\partial \eta} y_3 + \frac{\partial N_4^{4Q}}{\partial \eta} y_4.
\end{aligned}$$

Similarly, one can take the partial derivates of the shape functions with respect to the natural coordinates, which are given by:

$$\begin{aligned}
\frac{\partial N_1^{4Q}}{\partial \zeta} &= -\frac{1}{4}(1-\eta), & \frac{\partial N_1^{4Q}}{\partial \eta} &= -\frac{1}{4}(1-\zeta), \\
\frac{\partial N_2^{4Q}}{\partial \zeta} &= \frac{1}{4}(1-\eta), & \frac{\partial N_2^{4Q}}{\partial \eta} &= -\frac{1}{4}(1+\zeta), \\
\frac{\partial N_3^{4Q}}{\partial \zeta} &= \frac{1}{4}(1+\eta), & \frac{\partial N_3^{4Q}}{\partial \eta} &= \frac{1}{4}(1+\zeta), \\
\frac{\partial N_4^{4Q}}{\partial \zeta} &= -\frac{1}{4}(1+\eta), & \frac{\partial N_4^{4Q}}{\partial \eta} &= \frac{1}{4}(1-\zeta).
\end{aligned}$$

Using these partial derivatives, one can then populate the Jacobian matrix in A.11. This can be succinctly represented in the following way:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \zeta} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \zeta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} N^{4Q}(\zeta, \eta) \end{bmatrix} \begin{bmatrix} \{x\} & \{y\} \end{bmatrix} \tag{A.12}$$

where:

$$\begin{aligned}
N^{4Q}(\zeta, \eta) &= \begin{bmatrix} \frac{\partial N_1^{4Q}}{\partial \zeta} & \frac{\partial N_1^{4Q}}{\partial \eta} \\ \frac{\partial N_2^{4Q}}{\partial \zeta} & \frac{\partial N_2^{4Q}}{\partial \eta} \\ \frac{\partial N_3^{4Q}}{\partial \zeta} & \frac{\partial N_3^{4Q}}{\partial \eta} \\ \frac{\partial N_4^{4Q}}{\partial \zeta} & \frac{\partial N_4^{4Q}}{\partial \eta} \end{bmatrix}, \\
\{x\} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T, \\
\{y\} &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T
\end{aligned}$$

Conceptually $\mathbf{J}$ can be used to map points from the global coordinate system to the natural coordinate system in the following way:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \mathbf{J} \begin{bmatrix} \zeta_i \\ \eta_i \end{bmatrix} \tag{A.13}$$

From Expression A.12, the determinant of the jacobian is written as:

$$\det(\mathbf{J}) = \frac{\partial x}{\partial \zeta}\frac{\partial y}{\partial \eta} - \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \zeta} \tag{A.14}$$

Before a change of variables from the global to the natural coordinate system can take place through the use of the Jacobian, the strain-displacement matrix $\boldsymbol{B}$ needs to be defined. As the name suggests, $\boldsymbol{B}$ describes the relationship between the strains and the displacements experienced by an element.

The strain vector, $\boldsymbol{\varepsilon}$, is given by:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} \tag{A.15}$$

where ε_{xx} and ε_{yy} are the normal strains in the x and y directions respectively, and γ_{xy} is the shear strain. u and v are element nodal displacement vectors in the horizontal and vertical direction of the global $x - y$ domain. Equation A.15 can also be represented in matrix notation as follows:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} \tag{A.16}$$

Using the multivariable chain rule, the partial derivatives of the displacements are:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{A.17}$$

$$\frac{\partial u}{\partial y} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial y} \tag{A.18}$$

$$\frac{\partial v}{\partial x} = \frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{A.19}$$

$$\frac{\partial v}{\partial y} = \frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial y} \tag{A.20}$$

The above expressions can be succinctly represented in matrix notation:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & 0 & 0 \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & 0 & 0 \\ 0 & 0 & \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ 0 & 0 & \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{bmatrix} \tag{A.21}$$

The nodal displacements of the element in the natural coordinate system can be expressed as follows, where the shape functions are expressions of the natural coordinates. This is similar to the mapping from Equation A.6, however, it is important to note that unlike in Equation A.6, the following expression is in the natural coordinate system:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} N_1^{4Q} & 0 & N_2^{4Q} & 0 & N_3^{4Q} & 0 & N_4^{4Q} & 0 \\ 0 & N_1^{4Q} & 0 & N_2^{4Q} & 0 & N_3^{4Q} & 0 & N_4^{4Q} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} \tag{A.22}$$

Taking the derivative of the shape with respect to the natural coordinate system for the expression above:

$$\begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1^{4Q}}{\partial \xi} & 0 & \frac{\partial N_2^{4Q}}{\partial \xi} & 0 & \frac{\partial N_3^{4Q}}{\partial \xi} & 0 & \frac{\partial N_4^{4Q}}{\partial \xi} & 0 \\ \frac{\partial N_1^{4Q}}{\partial \eta} & 0 & \frac{\partial N_2^{4Q}}{\partial \eta} & 0 & \frac{\partial N_3^{4Q}}{\partial \eta} & 0 & \frac{\partial N_4^{4Q}}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1^{4Q}}{\partial \xi} & 0 & \frac{\partial N_2^{4Q}}{\partial \xi} & 0 & \frac{\partial N_3^{4Q}}{\partial \xi} & 0 & \frac{\partial N_4^{4Q}}{\partial \xi} \\ 0 & \frac{\partial N_1^{4Q}}{\partial \eta} & 0 & \frac{\partial N_2^{4Q}}{\partial \eta} & 0 & \frac{\partial N_3^{4Q}}{\partial \eta} & 0 & \frac{\partial N_4^{4Q}}{\partial \eta} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} \tag{A.23}$$

Substitute Equation A.23 into Equation A.16 to arrive at the follow expression which describes the relationship between strains and displacements in the natural coordinate system:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & 0 & 0 \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & 0 & 0 \\ 0 & 0 & \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ 0 & 0 & \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial N_1^{4Q}}{\partial \xi} & 0 & \frac{\partial N_2^{4Q}}{\partial \xi} & 0 & \frac{\partial N_3^{4Q}}{\partial \xi} & 0 & \frac{\partial N_4^{4Q}}{\partial \xi} & 0 \\ \frac{\partial N_1^{4Q}}{\partial \eta} & 0 & \frac{\partial N_2^{4Q}}{\partial \eta} & 0 & \frac{\partial N_3^{4Q}}{\partial \eta} & 0 & \frac{\partial N_4^{4Q}}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1^{4Q}}{\partial \xi} & 0 & \frac{\partial N_2^{4Q}}{\partial \xi} & 0 & \frac{\partial N_3^{4Q}}{\partial \xi} & 0 & \frac{\partial N_4^{4Q}}{\partial \xi} \\ 0 & \frac{\partial N_1^{4Q}}{\partial \eta} & 0 & \frac{\partial N_2^{4Q}}{\partial \eta} & 0 & \frac{\partial N_3^{4Q}}{\partial \eta} & 0 & \frac{\partial N_4^{4Q}}{\partial \eta} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix} \tag{A.24}$$

Finally, from this expression, one can derive the strain-displacement matrix $\boldsymbol{B}$ which is given by:

$$\boldsymbol{B} = \begin{bmatrix} \frac{\partial N_1^{4Q}}{\partial \xi} & 0 & \frac{\partial N_2^{4Q}}{\partial \xi} & 0 & \frac{\partial N_3^{4Q}}{\partial \xi} & 0 & \frac{\partial N_4^{4Q}}{\partial \xi} & 0 \\ 0 & \frac{\partial N_1^{4Q}}{\partial \eta} & 0 & \frac{\partial N_2^{4Q}}{\partial \eta} & 0 & \frac{\partial N_3^{4Q}}{\partial \eta} & 0 & \frac{\partial N_4^{4Q}}{\partial \eta} \\ \frac{\partial N_1^{4Q}}{\partial \eta} & \frac{\partial N_1^{4Q}}{\partial \xi} & \frac{\partial N_2^{4Q}}{\partial \eta} & \frac{\partial N_2^{4Q}}{\partial \xi} & \frac{\partial N_3^{4Q}}{\partial \eta} & \frac{\partial N_3^{4Q}}{\partial \xi} & \frac{\partial N_4^{4Q}}{\partial \eta} & \frac{\partial N_4^{4Q}}{\partial \xi} \end{bmatrix} \tag{A.25}$$

Finally, all the parameters are defined to compute the elemental stiffness matrix. Notice that the variables have been appropriately changed from Equation A.5, as now the integration is taking place in the natural coordinate domain:

$$\boldsymbol{K_e} = t \int_{-1}^{1} \int_{-1}^{1} \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \det(\mathbf{J})\, d\xi\, d\eta \tag{A.26}$$

A.3.1 Gaussian Quadrature

Gaussian Quadrature is employed to compute $\boldsymbol{K_e}$. Knowing the order of the matrices in the expression, it can be shown that the polynomial that is being integrated is a degree 3 polynomial. This would imply that two integration points are needed, however, since the expression has a double integral, 4 integration points are required. The points are given by:

$$\begin{bmatrix} \zeta_1 & \eta_1 \\ \zeta_2 & \eta_2 \\ \zeta_3 & \eta_3 \\ \zeta_4 & \eta_4 \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \tag{A.27}$$

The associated weights are:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{A.28}$$

Finally Equation A.26 can be expression in Gaussian notation:

$$\boldsymbol{K_e} = t \sum_{i=1}^{2} \sum_{j=1}^{2} \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \det(\mathbf{J})\, w_i\, w_j \tag{A.29}$$

Appendix B

Theory of Topology Optimisation

B.1 Compliance and Sensitivity Derivations

Consider figure B.1 below. It represents a single element in the design domain. At each of the element's nodes, nodal displacements in the vertical and horizontal directions have been depicted. [1]

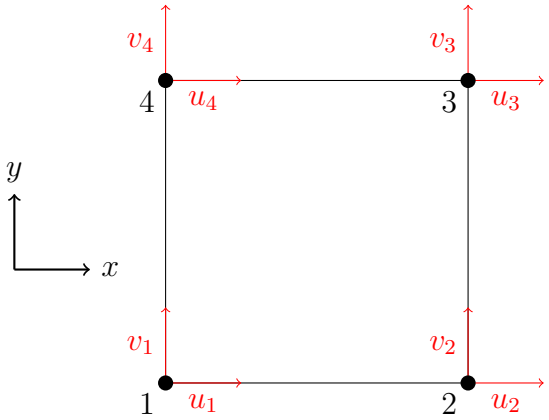

Figure B.1: Element with nodal displacements

The elemental nodal displacement vector and associated elemental force vectors can be represented as:

$$\mathbf{u_i}^T = \{u_1, v_1, \ldots, u_8, v_8\} \tag{B.1}$$

$$\mathbf{f_i}^T = \{f_{u_1}, f_{v_1}, \ldots, f_{u_8}, f_{v_8}\} \tag{B.2}$$

By Hooke's law, the elemental force vector $\mathbf{f}_i$ is related to the elemental displacement vector $\mathbf{u}_i$ through the use of the elemental stiffness matrix $\mathbf{K}_e$:

$$\mathbf{f}_i = \mathbf{K}_e \mathbf{u}_i \tag{B.3}$$

The elemental compliance C_i is defined as:

$$C_i = \mathbf{u}_i^T \mathbf{f}_i \tag{B.4}$$

[1] Notice that each node has 2 degrees of freedom, one in the x-direction and another in the y-direction. This implies that the total degrees of freedom for the square element is 8.

Referring to A.3, the elemental stiffness matrix for the "base" material, i.e., without any penalization, is given by:

$$\boldsymbol{K_e} = t \int_{-1}^{1} \int_{-1}^{1} \boldsymbol{B}^T \bar{\boldsymbol{Q}} \boldsymbol{B} \det(\mathbf{J}) \, d\xi \, d\eta \tag{B.5}$$

However, after the application of the SIMP method and incorporation of penalization, the effective elemental stiffness matrix becomes:

$$\mathbf{K}_{e_{eff}} = x_i^p \mathbf{K}_e \tag{B.6}$$

This equation means that the stiffness of any element can be varied between 0 and $\mathbf{K}_e$ based on the pseudo-density x_i, through the penalisation exponent, p. The elemental compliance considering the penalized material is:

$$C_i = \mathbf{u}_i^T \mathbf{K}_{e_{eff}} \mathbf{u}_i = \mathbf{u}_i^T x_i^p \mathbf{K}_e \mathbf{u}_i \tag{B.7}$$

The compliance of the complete structure is made up of the sum of the elemental compliances:

$$C = \sum_{i=1}^{n} C_i \tag{B.8}$$

substituting equation B.7 into the above expression yields:

$$C = \sum_{i=1}^{n} \mathbf{u}_i^T x_i^p \mathbf{K}_e \mathbf{u}_i \tag{B.9}$$

To determine how the global compliance changes with respect to the design variables, differentiate C with respect to each x_i:

$$\frac{\partial C}{\partial x_i} = \sum_{i=1}^{n} \frac{\partial (\mathbf{u}_i^T x_i^p \mathbf{K}_e \mathbf{u}_i)}{\partial x_i} \tag{B.10}$$

For a specific element i, this is:

$$\frac{\partial C}{\partial x_i} = \mathbf{u}_i^T p x_i^{p-1} \mathbf{K}_e \mathbf{u}_i \tag{B.11}$$

This expression effectively gives an indication of how a change in the design variable for element i affects the overall compliance of the entire structure. Here, it's crucial to highlight that both C_i and $\mathbf{K}_e$ are constants and are independent of the design variable x_i.

Substituting equation B.7 into the above expression yields:

$$\frac{\partial C}{\partial x_i} = p x_i^{p-1} C_i \tag{B.12}$$

The goal is to minimise the compliance of structure so to guide the optimiser, one has to take the negative of the expression above, which yields:

$$\frac{\partial C}{\partial x_i} = -p x_i^{p-1} C_i \tag{B.13}$$

B.1.1 A Clarification on Penalisation

When assembling the global stiffness matrix, penalization is applied to the elemental stiffness matrices. Having completed this, it's crucial to avoid re-penalizing these elemental stiffness matrices during the computation of the elemental compliance and sensitivity. This emphasis might appear counterintuitive, given the derivations presented above, however, the rationale is straightforward.

The displacements obtained from the global stiffness matrix, which has integrated the penalized elemental stiffness matrices, inherently account for the material property adjustments caused by penalization. One might conceptualize these displacements as "penalised displacements" since their computation contains the effects of the modified material properties through penalisation.

In the derivations above, C_i was introduced with the penalized stiffness matrices to ensure that the proof for the elemental compliance and sensitivity was exhaustive. However, if one has already incorporated the penalized matrices during the global stiffness matrix assembly, the elemental compliance, C_i, in the context of Equation B.13, should be defined as:

$$C_i = \mathbf{u}_i^T \mathbf{f}_i \tag{B.14}$$

This equation captures the inherent relationship between elemental displacements and forces, reflecting the effect of penalized stiffness.

B.2 On the Optimality Criteria (OC) Method

To understand the importance of the Lagrangian, $L(x, \lambda)$, and its use in the optimality criteria method presented below, it is useful to visualize the compliance minimisation optimisation problem. Unfortunately, this can be extremely difficult because the compliance function is a function of every design variable in the design domain. In other words, to accurately visualise the minimum compliance of the structure, under the volume constraint, one would have to plot each design variable on an independent "orthogonal" axis, with each of these axes themselves orthogonal to the compliance axis.

Visualising the optimisation problem in this form, in its hyperdimensionality, is extremely challenging. The figure below is a representation of the optimisation problem in 3D space which is much more accessible.

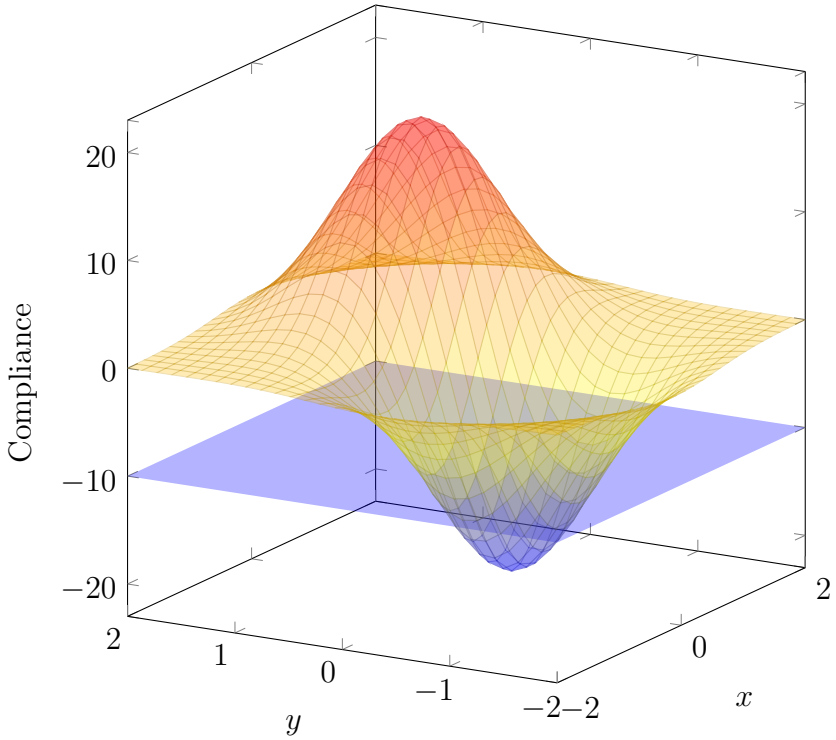

Figure B.2: An Approximation of the Compliance minimisation problem with a volume constraint.

The Lagrangian, $L(x, \lambda)$, is formed by combining the objective function (compliance) and the constraint (volume). By finding the local minima of the Lagrangian, one can identify the optimal design that minimizes compliance while satisfying the volume constraint. The visual representation above provides an intuitive understanding of this optimization problem, where the intersection of the compliance surface with the volume constraint plane indicates potential optimal designs. The lowest point within the bounds of this intersection is thus the most optimal design for a given set of design variables.

From the optimisation problem 3.4, the goal is to minimise the compliance function $C(x)$, while satisfying the volume constraint $V(x) \leq V_{\max}$. The Lagrangian $L(x, \lambda)$ for this problem can be expressed as:

$$L(x, \lambda) = C(x) + \lambda(V(x) - V_{\max}) \tag{B.15}$$

Where λ is the Lagrange multiplier associated with the volume constraint. V_{max} is the maximum volume of the structure and what defines the volume constraint, while $V(x)$ is the volume of the structure for a given set of design variables x. The goal of the optimiser is for the $V(x)$ to be as close to the V_{max} as possible, while minimizing the compliance $C(x)$.

Differentiating the Lagrangian with respect to a single design variable, x_i, will give an indication of how a change in that design variable will affect the Lagrangian. This is expressed as:

$$\frac{\partial L(x,\lambda)}{\partial x_i} = \frac{\partial C(x)}{\partial x_i} + \lambda \frac{dV(x)}{dx_i} \tag{B.16}$$

Considering the term $\frac{\partial V(x)}{\partial x_i}$, since x_i is the design variable for a specific element, the change in volume with respect to x_i is simply the volume of that element, which can be expressed as V_i. Hence:

$$\frac{\partial V(x)}{\partial x_i} = V_i \tag{B.17}$$

The aim is to find a design variable, x_i, that minimizes the Lagrangian. This is to seek a design variable that minimises the compliance of the structure and satisfies the volume constraint. Thus, Equation B.16 is set to zero:

$$\frac{\partial L(x,\lambda)}{\partial x_i} = 0 \tag{B.18}$$

Substituting in the expression for the derivatives, we get:

$$\frac{\partial C(x)}{\partial x_i} = -\lambda V_i \tag{B.19}$$

The expression above gives a relationship between the sensitivity and the volume constraint of an element, i. The Lagrange multiplier is then used to update the design using a heuristic approach, proposed by Bendsøe [35]:

$$x_i^{\text{new}} = \left(-\frac{\frac{\partial C(x)}{\partial x_i}}{\lambda V_i} \right)^{\eta} x_i^{\text{old}} \tag{B.20}$$

Where η is an exponent that controls how aggressively the design is updated.

From Equation B.20, a scaling factor can be defined as:

$$B_i = -\frac{\frac{\partial C}{\partial x_i}}{\lambda V_i} \tag{B.21}$$

Appendix C

Toppie

The following section contains the code framework for `Toppie`. The program contains several classes, each of which are contained in full below.

The following dependencies are required to run `Toppie`:

```
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
import math
from scipy.ndimage import convolve
from scipy.sparse.linalg import spsolve, splu, cg, gmres,
   bicgstab, lgmres, qmr
from scipy.sparse import csc_matrix, diags
from cvxopt import matrix, cholmod, spmatrix
```

Listing C.1: dependencies

C.1 Material Class

```
class Material:

    def __init__(self, material_type, E, v E1, E2, v12, G12,
   theta):

        self.E = E
        self.v = v
        self.E1 = E1
        self.E2 = E2
        self.v12 = v12
        self.G12 = G12
        self.theta = theta
        self.material_type = material_type

    def compute_material_matrix(self):
        """
        Compute the material matrix based on the given type
        """
        if self.material_type == "I":
```

```
            return self.Q_iso()
        elif self.material_type == "O":
            return self.Q_ortho()
        else:
            raise ValueError("Invalid material type. Choose 'I' for Isotropic and 'O' for Orthotropic.")

    def Q_iso(self):
        """
        Computes Reduced stiffness matrix for an isotropic element in plane-stress

        Parameters:
        E: Young's Modulus
        v: Poisson's Ratio

        """
        return (self.E/ (1 - self.v**2) * np.array([[1, self.v, 0], [self.v, 1, 0], [0, 0, (1 - self.v) / 2]]))

    def Q_ortho(self):
        """
        Computes Reduced stiffness matrix for an Orthotropic element in plane-stress

        Parameters:

        E1:     Young's Modulus in fibre-direction
        E2:     Young's Modulus in transverse direction
        v12:    Poisson's Ratio
        G12:    Shear Modulus
        theta:  fibre angle (degrees) relative to the global x-axis

        T: Transformation matrix
        Q: Reduced stiffness matrix in the material coordinate system
        R: Reuter matrix

        Returns:
        - np.ndarray: Reduced stiffness matrix for an Orthotropic element in plane-stress

        """
        v21 = self.v12 * self.E2 / self.E1
        theta_radians = -np.radians(self.theta)
        c = np.cos(theta_radians)
        s = np.sin(theta_radians)
        x = 1 - self.v12 * v21
        T = np.array([[c**2, s**2, 2*s*c],
                      [s**2, c**2, -2*s*c],
```

```
                        [-s*c, s*c, c**2 - s**2]])
        Q = np.array([[self.E1/x, v21*self.E1/x, 0],
                      [self.v12*self.E2/x, self.E2/x, 0],
                      [0, 0, self.G12]])
        R = np.array([[1, 0, 0],
                      [0, 1, 0],
                      [0, 0, 2]])

        return np.linalg.inv(T) @ Q @ R @ T @ np.linalg.inv(R)
```

Listing C.2: Material Class

C.2 Finite Element Class

```
class Element:

    GAUSS_POINTS_2D = np.array([(-0.5773502691896257, -0.5773502691896257),
                                (0.5773502691896257, -0.5773502691896257),
                                (0.5773502691896257, 0.5773502691896257),
                                (-0.5773502691896257, 0.5773502691896257)])

    GAUSS_WEIGHTS_2D    = np.ones(4)
    NATURAL_COORD_NODES = np.array([[-1, -1], [1, -1], [1, 1], [-1, 1]])

    def __init__(self, material_matrix, thickness=1.0):

        self.Q = material_matrix
        self.thickness = thickness
        self.KE = self.elemental_stiffness_matrix()

    def _shape_function_derivatives(self, zeta, eta):

        """
        Private function:

        Computes the derivatives of the shape functions with respect to the natural coordinates

        Parameters:
        zeta:       natural coordinate in the zeta direction
        eta:        natural coordinate in the eta direction

        Returns:

        Derivatives of the shape functions with respect to the natural coordinates
        """

        dN1_dzeta = -0.25 * (1 - eta)
        dN2_dzeta =  0.25 * (1 - eta)
        dN3_dzeta =  0.25 * (1 + eta)
        dN4_dzeta = -0.25 * (1 + eta)

        dN1_deta  = -0.25 * (1 - zeta)
        dN2_deta  = -0.25 * (1 + zeta)
        dN3_deta  =  0.25 * (1 + zeta)
        dN4_deta  =  0.25 * (1 - zeta)

```

```
        return (dN1_dzeta, dN2_dzeta, dN3_dzeta, dN4_dzeta), (dN1_deta, dN2_deta, dN3_deta, dN4_deta)

    def _compute_jacobian(self, dNdzeta, dNdeta, x_coords, y_coords):

        """
        Private function:

        Computes the Jacobian matrix and its determinant

        Parameters:
        dNdzeta:    derivatives of the shape functions with respect to the natural coordinate zeta
        dNdeta:     derivatives of the shape functions with respect to the natural coordinate eta
        x_coords:   x-coordinates of the nodes
        y_coords:   y-coordinates of the nodes

        returns:
        J:          Jacobian matrix
        detJ:       determinant of the Jacobian matrix

        """
        J11  = sum(np.array(dNdzeta) * x_coords)
        J12  = sum(np.array(dNdzeta) * y_coords)
        J21  = sum(np.array(dNdeta)  * x_coords)
        J22  = sum(np.array(dNdeta)  * y_coords)
        J    = np.array([[J11, J12], [J21, J22]])
        detJ = sc.linalg.det(J)

        return J, detJ

    def _compute_B_matrix(self, dNdzeta, dNdeta, J):

        """
        Private function:

        Computes the strain-displacement (B) matrix

        Parameters:
        dNdzeta:    derivatives of the shape functions with respect to the natural coordinate zeta
        dNdeta:     derivatives of the shape functions with respect to the natural coordinate eta
        J:          Jacobian matrix

        returns:
        B: strain-displacement matrix

        """
```

```
        invJ = sc.linalg.inv(J)
        dNdx = np.dot(invJ[0], [dNdzeta, dNdeta])
        dNdy = np.dot(invJ[1], [dNdzeta, dNdeta])
        B    = np.zeros((3,8))

        B[0, 0::2] = dNdx
        B[1, 1::2] = dNdy
        B[2, 0::2] = dNdy
        B[2, 1::2] = dNdx

        return B

    def elemental_stiffness_matrix(self):

        """
        Computes the elemental stiffness matrix using Gaussing Quadrature with helper functions
        _shape_function_derivatives, _compute_jacobian and _compute_B_matrix.

        Returns:

        ke: elemental stiffness matrix
        """

        x_coords = self.NATURAL_COORD_NODES[:, 0]           # fetch x-coordinates of the nodes from the NATURAL_COORD_NODES array
        y_coords = self.NATURAL_COORD_NODES[:, 1]           # fetch y-coordinates of the nodes from the NATURAL_COORD_NODES array

        ke = np.zeros((8, 8))                               # initialise the elemental stiffness matrix as a 8x8 matrix of zeros

        for i in range(4):

            zeta, eta    = self.GAUSS_POINTS_2D[i]
            dNdz, dNdeta = self._shape_function_derivatives(zeta, eta)
            J, detJ      = self._compute_jacobian(dNdz, dNdeta, x_coords, y_coords)
            B            = self._compute_B_matrix(dNdz, dNdeta, J)
            ke          += self.GAUSS_WEIGHTS_2D[i] * detJ * self.thickness * (B.T @ self.Q @ B)

        return ke
```

Listing C.3: Finite Element Class

C.3 Problem Class

```
class Problem:

    def __init__(self, X, Y):

        self.n_dof = 2 * (X + 1) * (Y + 1)
          # number of degrees of freedom
        self.X = X
          # number of elements in the x-direction
        self.Y = Y
          # number of elements in the y-direction
        self.F = np.zeros((self.n_dof, 1))
          # Initialise the force vector as a column vector of
    zeros (considering only y direction)
        self.alldofs = np.arange(self.n_dof)
          # Numpy array containing indices representing all
    degrees of freedom in the system.

    def half_MMB(self):

        node_of_interest = 0 + (self.Y + 1)                    #
    node just to the right of the top left node
        direction = 1                                          # 0
    for x-direction and 1 for y-direction
        self.F[2*node_of_interest + direction] = -1
        left_edge_dofs    = np.arange(0, 2*(self.Y + 1), 2)  #
    degrees of freedom on the left edge of the domain
        bottom_right_node = np.array([self.n_dof - 1])
        fixeddofs = np.union1d(left_edge_dofs, bottom_right_node)

        return self.F, fixeddofs, self.compute_freedofs(fixeddofs
    )

    def full_MMB(self):

        node_of_interest          = 0 + (self.X//2)*(self.Y + 1)
        direction                 = 1
                        # 0 for x-direction and 1 for y-direction
        self.F[2*node_of_interest + direction] = -1
        bottom_left_node          = np.array([2*self.Y, 2*self.Y
    +1])
        bottom_right_node         = np.array([self.n_dof - 1])
        fixeddofs                 = np.union1d(bottom_left_node,
    bottom_right_node)

        return self.F, fixeddofs, self.compute_freedofs(fixeddofs
    )

    def cantilever(self):

```

```
        node_of_interest = self.Y//2 + self.X * (self.Y + 1)
        direction = 1
        self.F[2*node_of_interest + direction] = -1
        left_edge_dofs_x = np.arange(0, 2*(self.Y + 1), 2)   # x-direction degrees of freedom on the left edge
        left_edge_dofs_y = np.arange(1, 2*(self.Y + 1), 2)   # y-direction degrees of freedom on the left edge
        fixeddofs = np.union1d(left_edge_dofs_x, left_edge_dofs_y)

        return self.F, fixeddofs, self.compute_freedofs(fixeddofs)

    def compute_freedofs(self, fixeddofs):
        """
        computes the difference between alldofs and fixeddofs to get the free degrees of freedom
        """
        return np.setdiff1d(self.alldofs, fixeddofs)
```

Listing C.4: Problem Class

C.4 Optimisation Class

```
class Optimisation:

    def __init__(self, X, Y, move):
        self.X = X
        self.T = Y
        self.move = move

    def initialisation(self, X, Y, f: float, C):

        """
        Initialisation of the optimisation problem

        Parameters:
        X: number of elements in the x-direction
        Y: number of elements in the y-direction
        f: volume fraction
        C: reduced stiffness matrix in global domain

        Returns:
        tuple: Containing density distribution, loop counter,
   error, penalisation factor,
        compliance derivative and elemental stiffness matrix.
        """

        x     = np.ones((Y, X)) * f
        loop  = 0
        error = 1.0
        c     = 0.0
        dc    = np.zeros((Y, X))
        inst_element = Element(material_matrix=C, thickness=1.0)
        ke    = inst_element.elemental_stiffness_matrix()

        return x, loop, error, c, dc, ke

    @staticmethod
    def comp(x, u, ke, P: int, edof):

        """
        Computes the compliance and its derivative (sensitivity).

        Parameters:
        x (numpy.ndarray):      Density distribution.
        u (numpy.ndarray):      Displacement vector.
        ke (numpy.ndarray):     Elemental stiffness matrix.
        P (int):                Penalisation factor.
        edof :                  Element degrees of freedom.

        Returns:
        tuple: Containing compliance and its derivative.
```

```
        """

        x  = x.T.flatten() # density array transposed and flattened to 1-d array
        u  = u[edof]       # extract the displacement variables corresponding to each element DOF

        f  = np.dot(ke, u.reshape((X*Y, 8, 1)))            # compute the element forces for each element
        C_e  = np.sum(u.squeeze() * f.squeeze().T, axis=1) # compute the element compliance for each element
        dc   = -P * (x ** (P-1)) * C_e

        C    = np.sum(C_e * x.T**P)
      # total compliance of the structure
        dc   = dc.reshape((X, Y)).T

        return C, dc

    def filt(self, x, rmin, dc):

        """
        Filters sensitivies to mitigate checkerboarding effect. [TopOpt]

        Parameters:
        x (numpy.ndarray):  density distribution
        rmin (float):       minimum radius
        dc (numpy.ndarray): compliance derivative (sensitivity)

        Returns:
        numpy.ndarray: filtered sensitivities
        """

        rminf   = math.floor(rmin)

        size    = rminf*2+1
        kernel  = np.array([[max(0, rmin - np.sqrt((rminf-i)**2 + (rminf-j)**2)) for j in range(size)] for i in range(size)])
        kernel /= kernel.sum()                                  # kernel is normalised such that its elements sum to 1

        xdc     = dc * x                                        # sensitivies (dc) are weighed by the densisties, x (element-wise multiplication)
        xdcn    = convolve(xdc, kernel, mode='reflect')     # convolve the sensitivities with the kernel
        dcn     = xdcn / x                                      # normalised convutioned sensitivities [care for near zero densities]

```

```
87          return dcn
88
89      @staticmethod
90      def OC(X, Y, x, f, dc):
91
92          """
93          Optimality criteria
94
95          Parameters:
96          X:      number of elements in the x-direction
97          Y:      number of elements in the y-direction
98          x:      density distribution
99          f:      volume fraction
100         dc:     compliance derivative (sensitivity)
101
102         Returns:
103         xnew: new density distribution
104         """
105
106         l1    = 0
107         l2    = 100000
108         eta   = 0.5
109         x_min = 0.001 # ensures that minimum density is not 0,
    which would causse numerical instabilities.
110
111         while l2 - l1 > 1e-4:
112
113             lmid = 0.5 * (l2 + l1)
114             xnew = np.maximum(x_min,
115                               np.maximum(x - move,
116                                          np.minimum(1,
117                                                     np.minimum(x
    + move,
118                                                                x
    * (-dc / ((f/1)*lmid))**(eta)))))
119             # check if the volume constraint is satisfied
120             if np.sum(xnew) - f * X * Y > 0:
121                 l1 = lmid
122             else:
123                 l2 = lmid
124
125         return xnew
```

Listing C.5: Optimisation Class

C.5 FEAnalysis Class

```
class FEAnalysis:

    ELEMENT_SIZE = 64
    NODES_PER_ELEMENT = 8
    DOF_MULTIPLIER = 2

    def __init__(self, X, Y, E0):
        self.X     = X
        self.Y     = Y
        self.n_el  = X * Y
        self.n_dof = self.DOF_MULTIPLIER * (X + 1) * (Y + 1)
        self.elx, self.ely, self.edofMat = self.
    initialize_elements()
        self.E0 = E0

    def initialize_elements(self):
        """
        Initializes the finite element analysis setup by
    preparing element degrees of freedom,
        and allocating space for the global stiffness matrix data
     structure.

        Returns:
        tuple: Contains arrays representing the x-indices, y-
    indices of each element in the grid and the element degrees of
     freedom matrix.
        """
        elx = np.repeat(np.arange(self.X), self.Y).reshape((self.
    n_el, 1))    # column vector where the index of each element
    along the X-axis is repeated 'Y' times
        ely = np.tile(np.arange(self.Y), self.X).reshape((self.
    n_el, 1))      # column vector where the indices of the
    elements along the Y-axis are repeated for each column in the
    grid.

        # np.repeat and np.tile are used to construct these
    arrays (elx and ely)
        # such that each pair (elx[i], ely[i]) represents the
    coordinates of the
        # i-th element in the grid.

        # calculate the node numbers of the four corner nodes of
    each element:

        n1 = (self.Y+1)*elx + ely                    # bottom-left
     node of the element
        n2 = (self.Y+1)*(elx + 1) + ely              # bottom-
    right node of the element
        n3 = (self.Y+1)*(elx + 1) + (ely + 1)        # top-right
```

```
node of the element
        n4 = (self.Y+1)*elx + (ely + 1)             # top-left
node of the element

        # Define the DOF of each node:
        # For each node index n, 2*n and 2*n+1 represent the
degrees of freedom
        # in the x and y directions, respectively;

        # The element degrees of freedom matrix (edofMat)
contains the
        # degrres of freedom associated with each node of each
element.
        # The size of edofMat is:
        # number of elements * number of nodes per element *
number of degrees of freedom per node.
        edofMat = np.hstack([2*n1, 2*n1+1,
                             2*n2, 2*n2+1,
                             2*n3, 2*n3+1,
                             2*n4, 2*n4+1])

        FEAnalysis.edofMat = np.hstack([2*n1, 2*n1+1, 2*n2, 2*n2
+1, 2*n3, 2*n3+1, 2*n4, 2*n4+1])

        return elx, ely, edofMat

    @staticmethod
    def csc_to_cvxopt(K_free):

        data = K_free.data.tolist()
        row  = K_free.indices.tolist()
        col  = np.repeat(np.arange(K_free.shape[1]), np.diff(
K_free.indptr)).tolist()

        return spmatrix(data, row, col, size=K_free.shape)

    def compute_displacements(self, x, P, ke, F, fixeddofs,
freedofs, E0):
        """
        This method performs the finite element analysis to find
the displacement vector.

        Parameters:
        x (np.ndarray): Density distribution.
        P (int): Penalisation factor.
        ke (np.ndarray): Elemental stiffness matrix with shape
(8, 8).
        F (np.ndarray): Force vector with shape (n_dof, 1).
        fixeddofs (np.ndarray): Array containing indices
representing the fixed degrees of freedom in the system.
        freedofs (np.ndarray): Array containing indices
```

```
    representing the free degrees of freedom in the system.
74
75          Returns:
76          np.ndarray: Displacement vector with shape (n_dof, 1).
77          """
78
79          K          = self._assemble_stiffness_matrix(x, P, ke, E0
    )       # Assemble the global stiffness matrix
80          K_free     = K[np.ix_(freedofs, freedofs)]
        # Extract the free part of the stiffness matrix
81          K_free_cvx = self.csc_to_cvxopt(K_free)
        # Convert the free part of the stiffness matrix to a
    cvxopt matrix
82          F_free_cvx = matrix(F[freedofs])
        # Convert the free part of the force vector to a cvxopt
    matrix
83          U          = np.zeros((self.n_dof, ))
        # Initialise the displacement vector
84
85          cholmod.linsolve(K_free_cvx, F_free_cvx)
        # Solve the linear system of equations Ku = F for the free
     part of the displacement vector
86
87          U[freedofs]  = np.array(F_free_cvx).flatten()
        # Extract the free part of the displacement vector from
    the cvxopt matrix
88          U[fixeddofs] = 0
        # Set the fixed part of the displacement vector to zero
89
90          return U.reshape(-1, 1)
91
92      def _assemble_stiffness_matrix(self, x, P, ke, E0):
93          """
94          This method assembles the global stiffness matrix.
95
96          Parameters:
97          x (np.ndarray): Density distribution
98          P (int): Penalisation factor.
99          ke (np.ndarray): Elemental stiffness matrix with shape
    (8, 8).
100
101         Returns:
102         tuple: Contains arrays representing the row indices,
    column indices, and data for the global stiffness matrix.
103         """
104         Emin = 10e-6
105
106         ke_flat = ke.flatten()
107         # xe = x[self.ely, self.elx] ** P               # not the
     most accurate penalisation but appears sound
108         xe = (Emin + x[self.ely, self.elx] ** P * (E0 - Emin))
```

```
109
110         edofMat_repeated = np.repeat(self.edofMat[:, :, np.
    newaxis], self.NODES_PER_ELEMENT, axis=2)
111         edofMat_tiled    = np.tile(self.edofMat[:, np.newaxis,
    :], (1, self.NODES_PER_ELEMENT, 1))
112
113         assembled_rows = edofMat_repeated.reshape(self.n_el, -1).
    flatten()
114         assembled_cols = edofMat_tiled.reshape(self.n_el, -1).
    flatten()
115         assembled_data = (xe[:, np.newaxis] * ke_flat[np.newaxis,
     :]).reshape(self.n_el, -1).flatten()
116
117         K = csc_matrix((assembled_data, (assembled_rows,
    assembled_cols)), shape=(self.n_dof, self.n_dof))
118
119         return K
```

Listing C.6: Finite Element Class

C.6 Main Function

```
def main(X, Y, f, P, rmin, material_type, E, v, E1, E2, v12, G12,
    theta, problem_type, move, solver_type, force, max_iterations
   ):

    if material_type == "I":
        E0  = E
    elif material_type == "O":
        E0  = E2

    # Instantiate classes
    inst_problem       = Problem(X, Y, force)
    elx, ely, edofMat  = inst_problem.initialize_elements()
    inst_material      = Material(material_type, E, v, E1, E2,
   v12, G12, theta)
    inst_optimisation  = Optimisation(X, Y, move)
    inst_FEAnalysis    = FEAnalysis(X, Y, E0, elx, ely, edofMat)
    inst_visualization = Visualization()

    # Problem initialisation
    if problem_type == "h":
        F, fixeddofs, freedofs = inst_problem.half_MBB()
    elif problem_type == "f":
        F, fixeddofs, freedofs = inst_problem.full_MBB()
    elif problem_type == "c":
        F, fixeddofs, freedofs = inst_problem.cantilever()
    elif problem_type == "c2":
        F, fixeddofs, freedofs = inst_problem.cantileverV2()
    else:
        raise ValueError("Unknown problem type: " + problem_type)


    Q = inst_material.compute_material_matrix()
    x, loop, error, C, dc, ke, = inst_optimisation.initialisation
   (X, Y, f, Q)

    x_values = []
    current_iteration = 0

    while error > 0.01 and current_iteration < max_iterations:


        current_iteration += 1
        loop += 1
        xold  = x

        U     = inst_FEAnalysis.compute_displacements(x, P, ke, F
   , fixeddofs, freedofs, E0, solver_type)
        C, dc = inst_optimisation.comp(x, U, ke, P, edofMat)
        dc    = inst_optimisation.filt(x, rmin, dc)
```

```
        x     = inst_optimisation.OC(X, Y, x, f, dc)
        error = np.max(np.abs(x - xold))

        x_values.append(x.copy())

        inst_visualization.display_results(loop, C, x, X, Y,
    error, problem_type, material_type, theta)

    plt.show()


if __name__ == '__main__':

    material_type = "I"
    problem_type  = "h"

    # Isotropic Material Properties:
    E  = 1          # GPa
    v  = 0.36

    # Orthotropic Material Properties (Glass/epoxy)

    E1    = 36      # GPa
    E2    = 8.27    # GPa
    v12   = 0.26
    G12   = 4.14    # GPa
    theta = -30     # degrees

    X    = 60       # number of elements in the x-direction
    Y    = 20       # number of elements in the y-direction
    f    = 0.5      # volume fraction
    P    = 3        # penalisation factor
    rmin = 2.5      # filter radius
    move = 0.2      # move limit


    force = 50      # N

    max_iterations = 200

    # cg
    # spsolve
    # bicgstab
    # cvxopt
    # lgmres
    # qmr
    # SuperLU
    # UMFPACK

    solver_type = "cvxopt"

```

```
95      main(X, Y, f, P, rmin, material_type, E, v, E1, E2, v12, G12,
        theta, problem_type, move, solver_type, force, max_iterations
    )
```

Listing C.7: Main function