applied sciences

MDPI

Review

Stress-Constrained Topology Optimization for Commercial Software: A Python Implementation for ABAQUS®

Pedro Fernandes [1,*], Àlex Ferrer [2], Paulo Gonçalves [1], Marco Parente [1,3], Ricardo Pinto [1] and Nuno Correia [1]

[1] INEGI—Institute for Science and Innovation in Mechanical and Industrial Engineering, 4200-465 Porto, Portugal; prgoncalves@inegi.up.pt (P.G.)
[2] CIMNE—International Center for Numerical Methods in Engineering, Campus Nord UPC, S/N 08034 Barcelona, Spain
[3] FEUP—Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
* Correspondence: pnfernandes@inegi.up.pt

Abstract: Topology optimization has evidenced its capacity to provide new optimal designs in many different disciplines. However, most novel methods are difficult to apply in commercial software, limiting their use in the academic field and hindering their application in the industry. This article presents a new open methodology for solving geometrically complex non-self-adjoint topology optimization problems, including stress-constrained and stress minimization formulations, using validated FEM commercial software. The methodology was validated by comparing the sensitivity analysis with the results obtained through finite differences and solving two benchmark problems with the following optimizers: Optimality Criteria, Method of Moving Asymptotes, Sequential Least-Squares Quadratic Programming (SLSQP), and Trust-constr optimization algorithms. The SLSQP and Trust-constr optimization algorithms obtained better results in stress-minimization problem statements than the methodology available in ABAQUS®. A Python implementation of this methodology is proposed, working in conjunction with the commercial software ABAQUS® 2023 to allow a straightforward application to new problems while benefiting from a graphic user interface and validated finite element solver.

Keywords: topology optimization; stress constraints; Python; ABAQUS; educational

Citation: Fernandes, P.; Ferrer, À.; Gonçalves, P.; Parente, M.; Pinto, R.; Correia, N. Stress-Constrained Topology Optimization for Commercial Software: A Python Implementation for ABAQUS®. *Appl. Sci.* **2023**, *13*, 12916. https://doi.org/10.3390/app132312916

Academic Editors: Aniello Riccio and Angela Russo

Received: 27 October 2023
Revised: 26 November 2023
Accepted: 29 November 2023
Published: 2 December 2023

1. Introduction

Originally described by Bendsøe and Kikuchi in 1988 [1], topology optimization is one of the three sub-fields of structural optimization, amongst size and shape optimization. Usually applied in the early stage of structural design, its purpose is to find the optimal distribution of material when certain load conditions are applied [2,3]. The design region is usually represented with density based or level-set functions, which are optimized according to a certain objective function and constraints, defining which regions should have material and which should not [4].

The popularity of topology optimization has led to the publishing of several educational articles [5]. These include the MATLAB codes written by Ole Sigmund [6], later revisited by Andreassen et al. [7], and an equivalent implementation for 3D problems by Liu [8], all of them using the Solid Isotropic Material with Penalization (SIMP) method [9,10] to define the material properties and the Optimality Criteria (OCs) [11] to determine the optimal design variables. Other publications include the MATLAB implementation of the level-set method [12,13] by Challis [14] and the implementations of the Bi-directional Evolutionary Structural Optimization (BESO) method [15] in MATLAB and Python by Huang and Xie [16] and Zuo and Xie [17], respectively. Table 1 compiles the publications on the development and implementation of topology optimization methods, sorted by the problem statement addressed, highlighting:

- The optimization algorithm used;
- If the publication provides the code implementation;
- If the method considers more constraints than just a mass or volume limit;
- If the method is suitable for complex geometries, allowing an easy application to case studies with intricate design spaces or the interaction with CAD models;
- If the method is based on commercial solvers.

Table 1. Literature review summary table, including scientific articles referred to in the ABAQUS® and ANSYS® documentation, sorted by scope. The black dots indicate an affirmative answer to the questions posed in each column.

Reference Source	Scope of the Reference	Author	Algorithm	Provides Code?	Consider Non-Material Constraint?	Suitable for Complex Geometries?	Based on Commercial Solver?
Literature	Review article or complementary method	Sigmund [18]	-				
		Zhu et al. [19]	-				
		Sigmund and Maute [20]	OC	•			
		Wang et al. [5]	-				
		Harzheim and Graf [21]	-				
	Compliance minimization	Sigmund [6]	OC	•			
		Andreassen et al. [7]	OC	•			
		Challis [14]	Level-set method	•			
		Zuo and Xie [17]	OC	•			•
		Suresh [22]	Pareto-optimal tracing	•			
		Talischi et al. [23]	OC	•			
		Smith and Norato [24]	MMA and fmincon	•			
		Sanders et al. [25]	Modified OC	•			
		Otomori et al. [26]	Level-set method	•			
		Gao et al. [27]	OC	•			
		Xia and Breitkopf [28]	OC	•			
		Liu and Tovar [8]	OC, SLSQP, and MMA	•			
		Aage et al. [29]	MMA	•		•	
		Liang and Cheng [30]	OC	•			

Table 1. *Cont.*

Reference Source	Scope of the Reference	Author	Algorithm	Provides Code?	Consider Non-Material Constraint?	Suitable for Complex Geometries?	Based on Commercial Solver?
Literature	Compliance minimization	Picelli et al. [31]	OC	•			
		Ferrari and Sigmund [32]	OC	•			
		Du et al. [33]	Level-set method	•			
		Chen et al. [34]	MMA	•			•
		Sotiropoulos et al. [35]	OC, MMA				
	Stress-constrained compliance minimization	Londoño and Paulino [36]	MMA	•	•		
		Yang et al. [37]	MMA		•		
		Bruggi and Venini [38]	MMA		•		
		Lee et al. [39]	SQP		•		
		Cai and Zhang [40]	Level-set method		•		
		Troya and Tortorelli [41]	MMA		•		
		Suresh and Takalloozadeh [42]	Level-set method		•	•	
		Chu et al. [43]	OC		•	•	
		Oh et al. [44]	Gradient-based phase field		•		
		Luo et al. [45]	MMA		•		
		Biyikli and To [46]	PTO	•	•		
		Amir [47]	MMA	•	•		
		París et al. [48]	SLP and steepest descent		•		
		Holmberg et al. [49]	MMA		•		
		Norato et al. [50]	MMA		•		

Table 1. *Cont.*

Reference Source	Scope of the Reference	Author	Algorithm	Provides Code?	Consider Non-Material Constraint?	Suitable for Complex Geometries?	Based on Commercial Solver?
Literature	Stress-constrained compliance minimization	Burger and Stainko [51]	Interior point method		•		
		De Leon et al. [52]	MMA		•		
		Holmberg et al. [53]	MMA		•		
		Pastore et al. [54]	SLP, MMA, and OC		•		
		París et al. [55]	SLP and steepest descent		•		
		Deng and Suresh [56]	Level-set method		•		
		Senhora et al. [57]	MMA		•		
		Saadlaoui et al. [58]	Commercial software (black box)		•	•	•
		Holmberg et al. [59]	MMA		•		
		Amir and Lazarov [60]	MMA		•		
		Granlund et al. [61]	MMA		•		
		Amstutz et al. [62]	Level-set method		•		
		Cheng et al. [63]	MMA		•		
		Deng et al. [64]	MMA	•	•		
		Mirzendehdel and Suresh [65]	Level-set method		•	•	
	Stress- and compliance-constrained mass minimization	Collet et al. [66]	MMA		•		
		Bruggi and Duysinx [67]	MMA		•		
	Stress-constrained mass minimization	Londoño et al. [68]	MMA		•		
		París et al. [69]	SLP		•		

Table 1. *Cont.*

Reference Source	Scope of the Reference	Author	Algorithm	Provides Code?	Consider Non-Material Constraint?	Suitable for Complex Geometries?	Based on Commercial Solver?
Literature	Stress-constrained mass minimization	Long et al. [70]	SQP and MMA		•		
		Duysinx [71]	MMA		•		
		Silva et al. [72]	Level-set method		•	•	•
	Stress-constrained compliance maximization	Miyajima et al. [73]	Level-set method		•		
		Emmendoerfer et al. [74]	Level-set method		•		
ABAQUS® documentation	Review article or complementary method	BendsØe and Sigmund [75]	OC, MMA, SLP, and level-set method	•	•		
	Compliance minimization	Bakhtiary et al. [76]	OC			•	•
ANSYS® documentation	Review article or complementary method	Mlejnek [77]	-				
		Stolpe and Svanberg [78]	-				
		Pedersen and Allinger [79]	-				
		BendsØe and Sigmund [80]	-				
		Clausen and Pedersen [81]	-				
		Svanberg [82]	-				
	Compliance minimization	Kabus and Pedersen [83]	Commercial software (black box)		•	•	•
	Minimization of maximum pressure	Søndergaard and Pedersen [84]	Commercial software (black box)		•	•	•
	Eigenvector minimization	Hansen [85]	MMA				
		Olhoff and Du [86]	MMA			•	•
	Compliance and eigenfrequency minimization	Jog [87]	OC				
		Sigmund and Jensen [88]	MMA		•		
	Maximization of the magnitude of steady-state vibrations	Tcherniak [89]	MMA		•		

Table 1 includes several publications with open access implementations of topology optimization methods suitable for compliance-minimization problems [6,7,14,17,22–32,34]. However, only four publications provide implementations of topology optimization methods compatible with stress-constrained compliance-minimization problems [36,46,47,64]. Regarding these three publications, Biyikli et al. [46] proposed a non-sensitivity method and Amir [47] proposed a method based on inexact sensitivities, leaving the MATLAB implementations proposed by Londoño and Paulino [36] and by Deng et al. [64] as the only publications providing a non-approximated solution to the stress-constrained compliance-minimization problem. On top of the reduced number of publications providing open access solutions with non-approximated methods to this problem statement, there are additional points of improvement that could be addressed to promote the research and application of topology optimization methods in the industry: including the compatibility with complex design spaces and geometries, as well as the interaction with commercially available and validated solvers. Finally, it is also relevant to consider that the topology optimization methods available in commercial software, such as ABAQUS® 2023 and ANSYS® 2023, operate on an opaque black-box system, limiting the access to relevant information and hindering their extension to a new feature of interest, such as reliability-based topology optimization [90–92] and robust topology optimization considering uncertainties [93].

To address this issue, the present article contributes with a methodology capable of solving non-self-adjoint problems in commercial software, including stress-constrained compliance minimization and stress-minimization problem statements, defined in a complex geometrical space. This methodology has been validated, evaluating the sensitivity analysis with finite differences and solving two benchmark problems with the following optimizers: Optimality Criteria, Method of Moving Asymptotes, Sequential Least-Squares Quadratic Programming, and Trust-constr optimization algorithms. In compliance-minimization problems and stress-constrained compliance-minimization problems, the methodology herein implemented achieved a performance equal to ABAQUS® in terms of the objective function. In stress-minimization problems, the methodology herein implemented allowed a stress reduction 35% superior to the one obtained with ABAQUS®.

To allow straightforward use even in an industrial field, a Python implementation is provided as Supplementary Material. This implementation works in conjunction with ABAQUS®, allowing a direct application to new problems, providing a graphic user interface and access to a commercially validated finite-element solver. The implementation can also be downloaded in digital format from the following Dataset [94] or repository: https://github.com/pnfernandes/Python-Code-for-Stress-Constrained-Topology-Optimization-in-ABAQUS.

The following sections of the article are organized as follows. Sections 2 and 3 formulate the topology optimization problem statements and deduce the necessary function derivatives in both continuous and discrete formulations, respectively. Section 4 describes the filter technique to void numerical instabilities. Section 5 overviews the optimization algorithms available in the provided code. Section 6 summarizes the most-relevant aspects of the Python code implementation, while in Section 7, the benchmark problems considered are presented. Section 8 demonstrates the suitability of the code implementation to solve the problems proposed. The results of the topology optimization processes are presented in Section 9. Finally, Section 10 summarizes the main conclusions of the present work.

2. Continuous Formulation of Topology Optimization Problem Statements and Sensitivities

In this section, the theoretical part of the topology optimization problem statement and the sensitivity analysis are defined in their continuous versions. Its practical counterpart, the discrete formulation, is presented in Section 3, which may allow an easier understanding of both the theoretical and implementation fundamentals of the topology optimization method.

2.1. Topology Optimization Problem

Consider the formulation of a generic topology optimization problem, defined as finding $\rho \in L^\infty(\Omega)$ such that:

$$\min_{\rho} : J(\rho, u(\rho)) \tag{1}$$

subject to:

$$\rho_{min} \leq \rho \leq 1 \tag{2}$$

$$\int_\Omega \rho \, dx \leq V^* \tag{3}$$

where Ω is the domain of the topology optimization problem, $L^\infty(\Omega)$ is the space of bounded functions, and the density ρ represents the design variables, which can vary within the interval $[\rho_{min}, 1]$. J represents a generic objective or cost function; $\int \rho \, dx$ represents a generic volume constraint function, where V^* is its maximum allowable value; $u(\rho)$ are the displacement solutions of the standard equilibrium equation presented as finding $u \in H_0^1(\Omega)$ such that:

$$a(\rho, u, v) = l(\rho, v), \; \forall \, v \in H_0^1(\Omega) \tag{4}$$

where $a(\rho, u, v)$ is the constitutive function of the equilibrium equation, $l(\rho, v)$ are the external forces, and $H_0^1(\Omega)$ is the space of functions with square integrable derivatives and homogeneous values on the boundary and domain $\Omega \in R^D$, with $D \in [2, 3]$. The bilinear form is:

$$a(\rho, u, v) = \int_\Omega \bigtriangledown^s u \mathbb{C}(\rho) \bigtriangledown^s v \, dx \tag{5}$$

with the external forces $l(\rho, v)$ written as:

$$l(\rho, v) = \int_\Omega f(\rho) \, v \, dx + \int_{\partial\Omega} t \cdot nv \, ds. \tag{6}$$

Here, $\bigtriangledown^s$ represents a symmetric gradient and $\mathbb{C}(\rho)$ is the constitutive tensor, and thus, the strains $\xi(u) = \bigtriangledown^s u$ are in accordance with linear elasticity. v is the corresponding test function; $f(\rho)$ is a generic volumetric load function; t indicates the applied boundary forces, if they exist; n is the normal direction pointing outwards of the boundary $\partial\Omega$ of v. It is important to note that this material constraint can be applied to either the volume or mass of the structure, following the same expressions. The volume constraint is preferred and used in this article to conform with the most-used term in the literature.

When considering a maximum stress constraint, the problem statement is also subject to:

$$\int_\Omega \sigma_a^{VM}(\rho, u(\rho)) \, dx \leq \sigma^* \tag{7}$$

where $\int_\Omega \sigma_a^{VM}(\rho, u(\rho)) \, dx$ represents the stress constraint with a maximum allowable value σ^*. In particular, σ_a^{VM} is the penalized von Mises stress, as described in the following subsection.

2.2. Regularization and Penalization

Regarding this particular work and implementation, it is important to note two adopted considerations. The first one is using two penalization factors applied to the material stiffness and stress. These factors aim at making intermediate design solutions uncompetitive and, in turn, promoting black-and-white solutions.

The material stiffness is penalized by the SIMP penalization parameter $P = 3.0$, in accordance with [80], when using continuous design variables (Equation (8)). In this implementation, this factor is also applied to the other material properties, as:

$$\mathbb{C}(\rho) = \rho^P \mathbb{C}_0 \tag{8}$$

where $\mathbb{C}_0$ is the material stiffness of a fully solid element. Note that $\mathbb{C}'(\rho) = P\rho^{P-1}\mathbb{C}_0$ for this type of stiffness penalization.

The stress is also penalized according to a factor equal to ρ^β, with $\beta = \frac{1}{2}$, as adopted in [53], following the initial proposal by Bruggi et al. [95] with the exponent suggested in [96]. This penalization leads to a non-physical stress for intermediate design densities, but not for black-and-white solutions, and tends towards 0 when the design density decreases, avoiding singularity problems [53]. Thus, σ_a is defined as the amplified stress, described by the following expression:

$$\sigma_a(\rho) = \rho^\beta \hat{\sigma}(\rho) \tag{9}$$

with:

$$\hat{\sigma}(\rho) = \mathbb{C}_0 \bigtriangledown^s u(\rho) \tag{10}$$

where $\hat{\sigma}(\rho)$ is the stress vector, written in Voigt notation. Therefore, the von Mises amplified stress norm is $\sigma_a^{VM} = (\sigma_a M \sigma_a)^{\frac{1}{2}}$, with M being the von Mises matrix operator.

The second consideration is the use of a regularization approach, where removed or void elements will maintain a minimum design density, $\rho_{min} = 0.01$ by default, which contrasts with the complete element removal. In the literature, these regularization techniques may be referred to as "soft-kill" and "hard-kill" approaches [17]. The regularization approach was introduced in the code implementation to avoid the stiffness matrix becoming singular, as well as allowing a constant mesh in the finite-element model during the topology optimization loop [17,53].

2.3. Sensitivity Analysis

The derivative of the cost function in the direction $\tilde{\rho} \in L^\infty(\Omega)$ is then defined as:

$$DJ(\rho, u(\rho))\tilde{\rho} = D_\rho J(\rho, u(\rho))\tilde{\rho} + D_u J(\rho, u(\rho))\ D_\rho u(\rho)\tilde{\rho}, \quad \forall \quad \tilde{\rho} \in L^\infty(\Omega) \tag{11}$$

Taking the derivative in the equilibrium equation, in order to find $D_u J(\rho, u(\rho))\ D_\rho u(\rho)\tilde{\rho}$, leads to:

$$\begin{aligned} &[D_\rho a(\rho, u(\rho), v) - D_\rho l(\rho, v)]\tilde{\rho} + D_u a(\rho, u(\rho), v)\ D_\rho u(\rho)\tilde{\rho} = 0 \\ &\quad \forall\, v \in H_0^1(\Omega),\ \forall\, \tilde{\rho} \in L^\infty(\Omega). \end{aligned} \tag{12}$$

Since $a(\rho, u(\rho), v)$ is linear, it can be rewritten as:

$$D_u a(\rho, u(\rho), v)\ D_\rho u(\rho)\tilde{\rho} = a(\rho, D_\rho u(\rho)\tilde{\rho}, v); \tag{13}$$

therefore, Equation (12) can be rewritten as follows:

$$-a(\rho, D_\rho u(\rho)\tilde{\rho}, v) = [D_\rho a(\rho, u(\rho), v) - D_\rho l(\rho, v)]\tilde{\rho}, \ \ \forall\, v, \tilde{\rho}. \tag{14}$$

Solving Equation (14) for all values of $\tilde{\rho}$ would be too expensive. For this reason, the use of the adjoint method is preferred. To do so, the adjoint variable λ is defined, the solution of:

$$\begin{aligned} &a(\rho, \lambda, w) = -D_u J(\rho, u(\rho))w \ \ \forall\, w \in H_0^1(\Omega) \Leftrightarrow \\ &\Leftrightarrow -a(\rho, D_\rho u(\rho)\tilde{\rho}, \lambda) = D_u J(\rho, u(\rho))\ D_\rho u(\rho)\tilde{\rho}, \ \ \forall\, \tilde{\rho}. \end{aligned} \tag{15}$$

Then, taking $w = D_\rho u(\rho)\tilde{\rho}$ in Equation (15):

$$\begin{aligned} D_u J(\rho, u(\rho))\, D_\rho u(\rho)\tilde{\rho} &= -a(\rho, \lambda, D_\rho u(\rho)\tilde{\rho}) = \\ &= -a(\rho, D_\rho u(\rho)\tilde{\rho}, \lambda) = \\ &= [D_\rho a(\rho, u(\rho), \lambda) - D_\rho l(\rho, v)]\tilde{\rho} \end{aligned} \tag{16}$$

where the self-adjoint property of $a(\rho, \lambda, w)$ and Equation (12) are applied.

The generic optimization process, which can be defined in four main steps, is proposed as follows:

- Find $u(\rho) \in H_0^1(\Omega)$, the solution of: $a(\rho, u, v) = l(\rho, v)$, $\forall\, v \in H_0^1(\Omega)$;
- Find λ, the solution of: $a(\rho, \lambda, w) = -D_u J(\rho, u(\rho))w$, $\forall\, w$;
- Compute the derivative as $DJ(\rho, u(\rho))\tilde{\rho} = [D_\rho J(\rho, u(\rho)) + a_\rho(\rho, u(\rho), \lambda) - l_\rho(\rho, \lambda)]\tilde{\rho}$;
- Update the design variables using the gradient information.

The first two steps correspond to solving the state and adjoint problems, respectively. The third step is the gradient calculation, which is defined in terms of the state and adjoint variables. The fourth and final step consists of using the gradient to determine the next value of the design variables, which can be performed by any suitable optimization algorithm (such as the algorithms described in Section 5).

2.4. Compliance Functional

In the particular problem statement of compliance minimization, the objective function can be rewritten as:

$$J(\rho, u(\rho)) = \int_\Omega f u \, dx \tag{17}$$

and its derivative in the direction $w \in H_0^1(\Omega)$ is then:

$$D_u J(\rho, u(\rho))w = \int_\Omega f w \, dx = l(w). \tag{18}$$

Notice that, in the particular case of compliance-minimization problem statements, it is commonly assumed in the literature that the external loads do not depend on ρ. Therefore:

$$a(\rho, u, v) = l(v). \tag{19}$$

In this particular case, the adjoint variable leads to the same expression shown in Equation (15), and introducing Equation (4), it is implied that:

$$\begin{aligned} &a(\rho, \lambda, w) = -D_u J(\rho, u(\rho))w, \quad \forall\, w \\ &\Leftrightarrow -a(\rho, u, w) = -D_u J(\rho, u(\rho)) = -l(w). \end{aligned} \tag{20}$$

Note that Equation (20), which describes the adjoint problem of the compliance-minimization problem statement, returns the definition of the state problem, setting them equal to each other. Functionals that are "self-adjoint" lead to clear computational benefits since Equations (4) and (15) can be solved with the same computation, a single FEA in the case of the code provided.

Since $l(v)$ and $l(w)$ do not depend on ρ, the terms $D_\rho J(\rho, u(\rho))\tilde{\rho} = D_\rho l(w)\tilde{\rho} = D_\rho l(v)\tilde{\rho} = 0$, and thus, the derivative is just:

$$DJ(\rho, u(\rho))\tilde{\rho} = -D_\rho a(\rho, u(\rho), u(\rho))\tilde{\rho}. \tag{21}$$

Following the definition of the compliance, the derivative of the bilinear form is:

$$D_\rho a(\rho, u(\rho), v) = \int_\Omega \bigtriangledown^s v \mathbb{C}'(\rho) \bigtriangledown^s u\tilde{\rho} \, dx \tag{22}$$

Finally, considering the stiffness penalization used, Equation (22) can be introduced into (21), obtaining:

$$DJ(\rho, u(\rho))\tilde{\rho} = \int_\Omega \bigtriangledown^s u(P\rho^{P-1}\mathbb{C}_0) \bigtriangledown^s u\tilde{\rho}\, dx = \int_\Omega g\tilde{\rho}\, dx \tag{23}$$

where g is the gradient of the compliance objective function, thus obtained as $g = \bigtriangledown^s u (P\rho^{P-1}\mathbb{C}_0) \bigtriangledown^s u$. For this particular case, the first three optimization steps can be rewritten as:

- Find $u(\rho) \in H_0^1(\Omega)$, the solution of: $a(\rho, u(\rho), v) = l(v),\ \forall\, v \in H_0^1(\Omega)$;
- Take $\lambda = -u$, since $a(\rho, \lambda, w) = -D_u J(\rho, u(\rho))w = -l(v),\ \forall\, w$ is exactly the same problem as the first step;
- Compute: $DJ(\rho, u(\rho))\tilde{\rho} = -a_\rho(\rho, u(\rho), u(\rho))\tilde{\rho} = \int_\Omega \bigtriangledown^s u(P\rho^{P-1}\mathbb{C}_0) \bigtriangledown^s u\tilde{\rho}\, dx$.

2.5. Stress Functional

In the stress-minimization or stress-constrained compliance-minimization problem statements, the derivative of the stress norm functional should also be taken. Here, the maximum function is approximated by means of a modified p-norm function. This approximation is necessary to provide a derivable function that approximates the maximum function, which is non-differentiable. Adopting the modified p-norm approximation proposed in [53], the maximum function represented in Equation (7) can be redefined as:

$$J(\rho, u(\rho)) = \left(\int_\Omega \left(\sigma_a^{VM}(\rho, u(\rho)) \right)^q dx \right)^{\frac{1}{q}} \tag{24}$$

where q is the exponential factor, and the von Mises amplified stress norm is $\sigma_a^{VM} = (\sigma_a M \sigma_a)^{\frac{1}{2}}$, with M being the von Mises matrix operator. Note that, although unconventional, this manuscript adopted q as the exponential factor of the p-norm approximation to improve the readability of the equation and to avoid confusion with the design density symbol ρ and the SIMP exponential factor P. Furthermore, it is relevant to highlight that, by adopting the use of this approximation, a global approach is selected, instead of local enforcement at each integration point. This option was preferred for providing better scaling with the mesh refinement, reducing the computational cost, and for its simplicity. However, the use of augmented Lagrangian methods could be seen as an alternative [57].

The term $D_\rho J(\rho, u(\rho))(\tilde{\rho})$ is defined as:

$$\begin{aligned} &D_\rho J(\rho, u(\rho))(\tilde{\rho}) = \\ &= \left(\int_\Omega \left(\sigma_a^{VM}(\rho, u(\rho)) \right)^q dx \right)^{\frac{1}{q}-1} \times q \int_\Omega \sigma_a^{VM}(\rho, u(\rho))^{q-1} D_{\sigma_a}\sigma_a^{VM}(D_\rho \sigma_a(\tilde{\rho}))\, dx \end{aligned} \tag{25}$$

where:

$$D_\rho \sigma_a(\tilde{\rho}) = \beta\rho^{\beta-1}\mathbb{C}_0 \bigtriangledown^s u\tilde{\rho}. \tag{26}$$

The derivative in the direction w, which allows us to define the adjoint problem, is:

$$\begin{aligned} &D_u J(\rho, u(\rho))(w) = \\ &= \left(\int_\Omega \left(\sigma_a^{VM}(\rho, u(\rho)) \right)^q dx \right)^{\frac{1}{q}-1} \times \int_\Omega \sigma_a^{VM}(\rho, u(\rho))^{q-1} D_{\sigma_a}\sigma_a^{VM}(D_u \sigma_a(w))\, dx \end{aligned} \tag{27}$$

where:

$$D_{\sigma_a}\sigma_a^{VM}(D_u\sigma_a(w)) = (\sigma_a M \sigma_a)^{-\frac{1}{2}} \sigma_a M D_u \sigma_a(w) \tag{28}$$

with:

$$D_u\sigma_a(w) = \rho^\beta \mathbb{C}_0 \bigtriangledown^s w. \tag{29}$$

Finally, before determining the value of $DJ(\rho, u(\rho))\tilde{\rho}$, it is required to find $D_\rho a(\rho, u(\rho), \lambda)$ as follows:

$$D_\rho a(\rho, u(\rho), \lambda)\tilde{\rho} = \int_\Omega \bigtriangledown^s \lambda \mathbb{C}'(\rho) \bigtriangledown^s u\tilde{\rho}. \tag{30}$$

As stated previously, note that $\bigtriangledown^s \lambda$ and $\bigtriangledown^s u$ represent the strains of the adjoint and state problems, respectively. Therefore, in the provided code implementation, these terms are obtained directly from the ABAQUS® FEA. Also, note that, for the stress functional, $D_\rho l(\lambda) = 0$. Finally, the first three optimization steps can then be rewritten as:

- Find u, the solution of $a(\rho, u(\rho)v) = l(\rho, v)$;
- Using Equation (27), find λ such that $a(\rho, \lambda, w) = -D_u J(\rho, u(\rho))w, \ \forall \ w$. Note that this functional is not self-adjoint, leading to a different procedure than the one shown in the previous section;
- Compute: $DJ(\rho, u(\rho))\tilde{\rho} = [D_\rho J(\rho, u(\rho)) \ + \ D_\rho a(\rho, u(\rho), \lambda)]\tilde{\rho}$.

3. Discrete Formulation of Topology Optimization Problem Statements and Sensitivities

In this section, the topology optimization problem statement and sensitivity analysis defined in Section 2 are now presented in their discretized versions. This information is included here to bridge the gap between the continuous framework shown in Section 2 and the implementation used in the code provided with this work, allowing an easier understanding of both the theoretical and implementation fundamentals of a topology optimization method.

3.1. Topology Optimization Problem

The discrete version of Problem (1) results in:

$$\min_\rho : J(\rho, u(\rho)) \tag{31}$$

subject to:

$$\rho_{min} \leq \rho_e \leq 1 \ \forall \ e = 1, \ldots, N \tag{32}$$

$$V(\rho) = \sum_\rho \rho_e v_e \leq V^* \tag{33}$$

where, as previously defined, $J(\rho)$ is the cost function, the density ρ represents the design variables, which can vary within the interval $[\rho_{min}, 1]$, $V(\rho)$ is the total volume, with v_e representing the volume of element e when the design density $\rho = 1$, V^* is the maximum value of the volume constraint, and $u(\rho)$ is the solution displacement vector of the state problem:

$$F = K(\rho)u. \tag{34}$$

The stiffness matrix $K(\rho)$ and force vector F are defined as, respectively:

$$K(\rho) = \int_\Omega B_a^T \mathbb{C} B_a dx \tag{35}$$

$$F = \int_\Omega N \ f \ dx + \int_{\partial\Omega} N \ t \cdot n \ ds \tag{36}$$

where B_a is the strain–displacement matrix in the evaluation point a, N a linear shape function, and f a generic load function, while t indicates the applied boundary forces, if they exist, and n is the normal direction pointing outwards of the boundary $\partial\Omega$.

When considering a maximum stress constraint, the problem statement is also subject to:

$$\sigma^{PN}(\rho) \leq \sigma^* \tag{37}$$

where $\sigma^{PN}(\rho)$ represents the maximum stress and σ^* its the maximum allowable value.

Note that the penalization factors and regularization process described in Section 2.2 are not changed.

3.2. Sensitivity Analysis

The gradient of the cost function can be defined as:

$$\bigtriangledown_\rho J = \frac{\partial J}{\partial \rho} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \rho}. \tag{38}$$

In order to find $\frac{\partial J}{\partial u}$, it is possible to reorganize the state equation, multiply it for a generic vector v, and derive the expression, leading to:

$$v^T\left[\frac{\partial K(\rho)}{\partial \rho}u(\rho) - \frac{\partial F(\rho)}{\partial \rho} + K(\rho)\frac{\partial u(\rho)}{\partial \rho}\right] = 0; \tag{39}$$

therefore:

$$-\left(\frac{\partial u(\rho)}{\partial \rho}\right)^T K^T(\rho)\, v = v^t\left[\frac{\partial K(\rho)}{\partial \rho}u - \frac{\partial F(\rho)}{\partial \rho}\right]. \tag{40}$$

As stated in Section 2, solving Equation (40) for all values of ρ would be too expensive, motivating the use of the adjoint method. Defining the adjoint variable as $\lambda = v$ such that $K^T(\rho)\lambda = -\frac{\partial J}{\partial u}$ leads to:

$$-\left(\frac{\partial u(\rho)}{\partial \rho}\right)^T K^T(\rho)\lambda = \left(\frac{\partial u}{\partial \rho}\right)^T \frac{\partial J}{\partial u} = \lambda^T\left[\frac{\partial K(\rho)}{\partial \rho}u - \frac{\partial F(\rho)}{\partial \rho}\right]. \tag{41}$$

Thus, the gradient can finally be rewritten as:

$$\bigtriangledown_\rho J = \frac{\partial J}{\partial \rho} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \rho} = \frac{\partial J}{\partial \rho} + \lambda^T\left[\frac{\partial F(\rho)}{\partial \rho} - \frac{\partial K(\rho)}{\partial \rho}u\right]. \tag{42}$$

With this information, the equivalent three-step process described in Section 2.3 can be rewritten as:

- Find u such that: $K(\rho)u = F(\rho)$;
- Find λ such that: $K(\rho)\lambda = -\frac{\partial J}{\partial u}$;
- Compute: $\bigtriangledown_\rho J = \frac{\partial J}{\partial \rho} + \lambda^T\left[\frac{\partial F(\rho)}{\partial \rho} - \frac{\partial K(\rho)}{\partial \rho}u\right]$.

3.3. Compliance Functional

The compliance can be defined as $C(\rho) = F\,u(\rho)$, and its sensitivity can be determined as follows [97–99]:

$$\frac{\partial C}{\partial \rho_e} = F\frac{\partial u}{\partial \rho_e}; \tag{43}$$

since $u(\rho) = K^{-1}(\rho)F$ and $\frac{\partial u}{\partial \rho_e} = -K^{-1}\frac{\partial K}{\partial \rho_e}K^{-1}F$, Equation (43) becomes:

$$\frac{\partial C}{\partial \rho_e} = -FK^{-1}\frac{\partial K}{\partial \rho_e}K^{-1}F = -u\frac{\partial K}{\partial \rho}u. \tag{44}$$

Considering that $K = A\int_{\Omega_e} B_a^T \mathbb{C} B_a \, dx$, with A being the assembly operator, and that ρ is constant in Ω_e, $\frac{\partial K}{\partial \rho}$ can be rewritten as follows:

$$\frac{\partial K}{\partial \rho} = \int_{\Omega_e} B_a^T \mathbb{C}' B_a \, dx = \int_{\Omega_e} B_a^T P \rho^{P-1} \mathbb{C}_0 B_a \, dx = \int_{\Omega_e} \frac{P}{\rho} B_a^T \mathbb{C} B_a \, dx = \frac{P}{\rho} K. \tag{45}$$

Introducing Equation (45) into (44) leads to:

$$\frac{\partial C}{\partial \rho_e} = -\frac{P}{\rho} \rho^P u_e^T K_0 u_e = -P \frac{E_e}{\rho} \tag{46}$$

where u_e and K_0 are the elemental displacement vector and stiffness matrix of a fully solid element (i.e., $\rho = 1.0$). The term $\rho^P u_e^T K_0 u_e$ is the strain energy (E_e), missing only the $\frac{1}{2}$ constant. However, because this constant is applied to all elements, it can be neglected, and set the term $\rho^P u_e^T K_0 u_e$ equal to the strain energy (E_e) automatically calculated in ABAQUS®.

In this particular case, the first three steps of the generic optimization process can be simplified and rewritten as follows:

- Find u such that: $K(\rho)u = F(\rho)$. This procedure can be performed using an ABAQUS® FEA;
- Take $\lambda = -u$, since the problem is self-adjoint;
- Compute: $\frac{\partial C}{\partial \rho_e} = -P\frac{E_e}{\rho}$. Note that E_e can be obtained from the ABAQUS® FEA executed in the first step.

This information is included to allow an easier understanding of the code implementation and its correlation with the formal mathematical formulation.

3.4. Stress Functional

The maximum function is approximated by means of a modified p-norm function. This approximation is necessary to provide a derivable function that approximates the maximum function, which is non-differentiable. This implementation adopts the modified p-norm approximation proposed in [53]:

$$\sigma^{PN}(\rho) = \left(\frac{1}{N_i} \sum_{\Omega} \left(\sigma_a^{vM}(\rho) \right)^q \right)^{\frac{1}{q}} \tag{47}$$

where q is the exponential factor, Ω is the set of stress evaluation points in the topology optimization problem, and σ_a^{vM} is the value of the amplified von Mises stress in point a. Note that, although unconventional, this manuscript adopted q as the exponential factor of the p-norm approximation to improve the readability of the equation and avoid confusion with the design density symbol ρ and the SIMP exponential factor P. Furthermore, it is relevant to highlight that, by adopting the use of this approximation, a global approach is selected, instead of local enforcement at each integration point. This option was preferred for providing better scaling with the mesh refinement, reducing the computational cost, and for its simplicity. However, the use of augmented Lagrangian methods could be seen as an alternative [57].

The derivative of the p-norm approximation with respect to the design density of an element can be obtained by the chain-rule, multiplying three intermediate terms. The first term is the derivative of the p-norm approximation with respect to the amplified von Mises stress:

$$\frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}} = \left(\frac{1}{N_i} \sum_{\alpha \in \Omega} \left(\sigma_a^{vM}(\rho) \right)^q \right)^{\frac{1}{q}-1} \times \frac{1}{N_i} \left(\sigma_a^{vM}(\rho) \right)^{q-1}. \tag{48}$$

The second term is the derivative of σ_a^{vM} with respect to the stress vector in point a. Since σ_a^{vM} can be written in matrix form as $\sigma_a^{vM} = (\sigma_a M \sigma_a)^{\frac{1}{2}}$, its derivative becomes:

$$\frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a(\rho)} = (\sigma_a(\rho) M \sigma_a(\rho))^{-\frac{1}{2}} \times \sigma_a(\rho) M \frac{\partial \sigma_a(\rho)}{\partial \rho}. \tag{49}$$

The third and last terms are the derivative of the stress vector σ_a with respect to the design density. Considering Equations (9) and (10):

$$\frac{\partial \sigma_a(\rho)}{\partial \rho} = \beta \rho^{\beta-1} \mathbb{C}_0 B_a u(\rho) + \rho^{\beta} \mathbb{C}_0 B_a \frac{\partial u(\rho)}{\partial \rho}. \tag{50}$$

The term $\frac{\partial u(\rho)}{\partial \rho}$ is obtained from the state Equation (34):

$$\begin{aligned} \frac{\partial K(\rho)}{\partial \rho} u(\rho) + K(\rho) \frac{\partial u(\rho)}{\partial \rho} = \frac{\partial F(\rho)}{\partial \rho} = 0 \Leftrightarrow \\ \Leftrightarrow \frac{\partial u(\rho)}{\partial \rho} = -K^{-1}(\rho) \frac{\partial K(\rho)}{\partial \rho} u(\rho) \end{aligned} \tag{51}$$

and, therefore, with $\beta = \frac{1}{2}$:

$$\frac{\partial \sigma_a(\rho)}{\partial \rho} = \frac{1}{2} \rho^{-\frac{1}{2}} \mathbb{C}_0 B_a u(\rho) - \rho^{\frac{1}{2}} \mathbb{C}_0 B_a K^{-1}(\rho) \frac{\partial K(\rho)}{\partial\, \rho} u(\rho). \tag{52}$$

Note that it was assumed that $\frac{\partial F(\rho)}{\partial \rho} = 0$. This assumption is valid for static-load-driven problems, where the load is considered to be constant and independent of the material distribution. For displacement-driven problems, this assumption is not valid, as the forces resulting from the displacement applied will change depending on the material distribution.

With Equations (48) through (51) and applying the chain rule, it is possible to define $\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}$ as:

$$\begin{aligned} \frac{\partial \sigma^{PN}(\rho)}{\partial \rho} = \frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}} \frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a} \frac{\partial \sigma_a(\rho)}{\partial \rho} = \\ = \frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}} \frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a} \left(\frac{1}{2} \rho^{-\frac{1}{2}} \mathbb{C}_0 B_a u(\rho) \right. \\ \left. - \rho^{\frac{1}{2}} \mathbb{C}_0 B_a K^{-1}(\rho) \frac{\partial K(\rho)}{\partial \rho} u(\rho) \right). \end{aligned} \tag{53}$$

Renaming these two terms as $\partial \sigma_{spf}^{PN}(\rho)$ (Equation (54)), referring to the component of the derivative that is dependent on the stress penalization factor, and $\partial \sigma_u^{PN}(\rho)$ (Equation (55)), referring to the component of the derivative that is dependent on the nodal displacement:

$$\partial \sigma_{spf}^{PN}(\rho) = \frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}} \frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a} \left(\frac{1}{2} \rho^{-\frac{1}{2}} \mathbb{C}_0 B_a u(\rho) \right) \tag{54}$$

$$\partial \sigma_u^{PN}(\rho) = \frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}} \frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a} \times \left(-\rho^{\frac{1}{2}} \mathbb{C}_0 B_a K^{-1}(\rho) \frac{\partial K(\rho)}{\partial\, \rho} u(\rho) \right). \tag{55}$$

Notice that $\partial \sigma_{spf}^{PN}(\rho)$ can be easily determined, since the only information required is the stress vector ($\hat{\sigma}_a(\rho) = \mathbb{C}_0 B_a u(\rho)$) and the design densities. $\frac{\partial \sigma^{PN}(\rho)}{\partial \sigma_a^{vM}}$ can be determined with a simple summation and $\frac{\partial \sigma_a^{vM}(\rho)}{\partial \sigma_a}$ through the product of two vectors and a matrix.

The term $\partial\sigma_u^{PN}(\rho)$, on the other hand, requires the explicit definition of the matrices B_a, $K^{-1}(\rho)$, and $\frac{\partial K(\rho)}{\partial \rho}$, which are dependent on the element formulation used in the numerical model. Furthermore, for topology optimization problems where the number of design variables is larger than the number of constraints, the most-efficient way to determine $\partial\sigma_u^{PN}(\rho)$ is using an adjoint model, defining the adjoint variable as:

$$\begin{aligned} K(\rho)\lambda &= \frac{\partial\sigma^{PN}(\rho)}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}(\rho)}{\partial\sigma_a}\mathbb{C}_0 B_a \Leftrightarrow \\ \Leftrightarrow \lambda &= \frac{\partial\sigma^{PN}(\rho)}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}(\rho)}{\partial\sigma_a}\mathbb{C}_0 B_a K^{-1}(\rho). \end{aligned} \tag{56}$$

Therefore, the adjoint variable can be extracted from a finite element analysis, where the load applied on each node is equal to $\frac{\partial\sigma^{PN}(\rho)}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}(\rho)}{\partial\sigma_a}\mathbb{C}_0 B_a$, and λ is equal to the node displacement. Introducing the adjoint variable in Equation (54) then leads to:

$$\partial\sigma_u^{PN}(\rho) = \frac{\partial\sigma^{PN}(\rho)}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}(\rho)}{\partial\sigma_a}\left(-\rho^{\frac{1}{2}}\lambda\frac{\partial K(\rho)}{\partial\,\rho}u(\rho)\right). \tag{57}$$

Finally, since K can be defined as $K = AB_a^T\mathbb{C}(\rho)B_a$, since $B_a\lambda$ and $B_a u$ are equal to the deformation vectors of the adjoint and state models (ξ_{adj} and ξ_s, respectively), Equation (57) can be simplified to:

$$\partial\sigma_u^{PN}(\rho) = \frac{\partial\sigma^{PN}}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}}{\partial\sigma_a}\left(-\rho^{\frac{1}{2}}\xi_{adj}^T(\rho)\mathbb{C}'(\rho)\xi_s(\rho)\right). \tag{58}$$

With this information, the first three steps of the generic optimization process can be rewritten as follows:

- Find u such that: $K(\rho)u = F(\rho)$. This procedure can be performed through an ABAQUS® FEA;
- Find λ such that: $K(\rho)\lambda = \frac{\partial\sigma^{PN}(\rho)}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}(\rho)}{\partial\sigma_a}\mathbb{C}_0 B_a$;
- Compute: $\partial\sigma_u^{PN}(\rho) = \frac{\partial\sigma^{PN}}{\partial\sigma_a^{vM}}\frac{\partial\sigma_a^{vM}}{\partial\sigma_a}\left(-\rho^{\frac{1}{2}}\xi_{adj}^T(\rho)\mathbb{C}'(\rho)\xi_s(\rho)\right)$.

As demonstrated in Equation (55) through (58), determining $\partial\sigma_u^{PN}(\rho)$ is significantly more complex and computationally expensive, due to the necessity of using the adjoint model. Some researchers [100] have proposed approximating this term as $\partial\sigma_u^{PN}(\rho) = 0$, in an attempt to reduce the complexity and computational cost of stress-dependent topology optimization problems. While doing so does have evident advantages, the results obtained for each element of the benchmark problem described in Section 7.2 show that, on average, $|\partial\sigma_u^{PN}(\rho)| > |\partial\sigma_{spf}^{PN}(\rho)|$. As a result, assuming that $\partial\sigma_u^{PN}(\rho) = 0$ would lead to an average error greater than 50% in the calculation of the maximum stress sensitivity, at least for the particular case of the benchmark problem described in Section 7.2. The error in this approximation is aggravated by the fact that $\partial\sigma_u^{PN}(\rho)$ has a negative sign from Equation (53), and as a consequence, ignoring this term could potentially point the optimization algorithm in the opposite direction of the gradient. Therefore, the simplification proposed in [100] was not adopted in this work.

3.5. Volume Constraint

The value of the sensitivity of the volume constraint to changes in the design density of each element is equal to the value of its volume, as defined in Equation (59). Note that the equivalent sensitivity can be obtained for a mass constraint, replacing the volume of the element with its mass.

$$\frac{\partial V(\rho)}{\partial\rho_e} = V_e. \tag{59}$$

4. Mesh Dependency and Data Filtering

In order to obtain a mesh-independent solution and avoid the "checkerboard" instability, the raw sensitivities and/or design densities are processed with a blurring filter [6,101]. This article adopted the filter scheme used in [17], which is a simplification of the scheme proposed by Huang and Xie [102], described as follows:

$$\rho_e = \sum_j \left(\frac{w(r_{ej})}{\sum_j w(r_{ej})} \rho_j \right) \tag{60}$$

where the value of $w(r_{ej})$ is equal to the difference between the maximum range of the filter (r_{max}) and the actual distance between the central element e and the j elements in its neighborhood (r_{ej}), defined as follows:

$$w(r_{ej}) = max(0, r_{max} - r_{ej}). \tag{61}$$

This parameter can be interpreted as a pondered measurement of how close the two elements are. Note that the code implementation provided allows the user to select if the blurring filter should be applied to the sensitivities (excluding the volume constraint sensitivities, or equivalent), the design densities, or both.

It is relevant to note that, although not implemented in the methodology herein proposed, it is common to couple filtering techniques with projection methods [103] in order to obtain a better geometry definition in the final solution. Doing so may eliminate residual grey areas, which is particularly relevant in stress-constrained problems, where a smoother geometry may avoid the creation of stress concentration points.

5. Optimization Algorithms

5.1. Optimality Criteria

The OC method implemented in this article follows the approach proposed by Bendsøe [99] and implemented by Sigmund in [6]. It is usually applied to problems of compliance minimization with a volume constraint. According to this procedure, the design variables can be updated according to the following expression:

$$\rho = \begin{cases} \rho_{lower}, & \text{if } \rho B_e^{\eta} \leq \rho_{lower} \\ \rho B_e^{\eta}, & \text{if } \rho_{lower} \leq \rho B_e^{\eta} \leq \rho_{upper} \\ \rho_{upper}, & \text{if } \rho_{upper} \leq \rho B_e^{\eta} \end{cases} \tag{62}$$

$$\rho_{lower} = max(\rho_{min}, \rho - \delta\rho) \tag{63}$$

$$\rho_{upper} = min(1.0, \rho + \delta\rho) \tag{64}$$

where ρ_{lower} and ρ_{upper} represent the design densities' move limits imposed by the parameter $\delta\rho$. η is a numerical damping coefficient set equal to 0.5 in accordance with [6]. B_e is a parameter obtained from the optimality condition, described as the ratio between the gradient of the objective and constraint functions divided by the Lagrangian multiplier (ψ), as follows:

$$B_e = \frac{-\frac{\partial C}{\partial \rho}}{\psi \frac{\partial V}{\partial \rho}} = \frac{-\frac{\partial C}{\partial \rho}}{\psi V_e}. \tag{65}$$

The Lagrangian multiplier ψ can be found using a bisection method, updating its value until the volume of the solution meets the imposed constraint.

This method can also be simplified in order to obtain an equivalent version suitable for discrete design variables. This simplification corresponds to the approach used by Zuo and Xie in [17]. For the current iteration, the code determines the sensitivities of all elements. Then, a bisection method is used to determine a threshold sensitivity. All elements with a sensitivity greater than the threshold become solid (i.e., $\rho = 1.0$), while the other elements are removed according to the "soft-kill" approach (i.e., $\rho = \rho_{min}$). Similar to

the continuous version, the value of the sensitivity threshold is updated until the mass or volume of the structure meets the volume constraint imposed.

When using the OC method, especially the discrete version, it is usual to consider a fully solid design for the initial iteration and gradually reduce the volume constraint in each iteration until the target value is reached. This procedure is adopted to obtain a convergent solution and can be defined through the following equation:

$$V^{k+1} = max(V^*, V^k(1.0 - e_{vol})) \tag{66}$$

where k is the number of the current iteration and e_{vol} is the ratio at which the volume constraint decreases in each iteration, until the target volume constraint (V^*) is reached.

Nonetheless, it is important to understand that this discrete version should be regarded as a heuristic, which was observed to work well when solving compliance-minimization problems. However, this procedure can be criticized from an optimization perspective, as there is no mathematical guarantee that the objective function will decrease during each iteration.

5.2. Method of Moving Asymptotes

The Method of Moving Asymptotes (MMA) is a nonlinear programming method that can be applied to structural optimization problems. Its functioning is based on an iterative process, generating a series of strictly convex approximating sub-problems. As an input to generate the sub-problem approximation, the MMA takes the value of the objective function, the value of the constraints, and the respective gradients as the inputs. Then, using dual-methods such as [104,105], the sub-problem is solved and the solution is taken as the value of the design variables in the next iteration of the main optimization problem [82].

The popularity of the application of the MMA to topology optimization problems is justified by several factors. From a computational point of view, the use of an approximation sub-problem and gradient information provides a means to reduce the number of structural problems solved, which are usually expensive. Furthermore, the versatility of the method and capability of handling multiple types of constraints make it suitable and applicable to a wide range of cases, from simple compliance-minimization problems with a single volume constraint, to cases where multiple constraints with different natures are applied. Its stability is also a positive characteristic, even for initial non-fully solid designs. Due to its extension and complexity, the interested reader is referred to [82] for a detailed explanation of this method.

The present code establishes a connection between ABAQUS® and the implementation of the MMA created by Deetman [106] ("mmasub" and "subsolve" functions of the code provided), which is a Python version of the MATLAB code created by Svanberg [107].

5.3. Sequential Least-Squares Programming

Sequential Least-Squares Programming (SLSQP) is a Sequential Least-Squares Programming algorithm that utilizes the Quasi-Newton method to obtain an optimal solution. The strategy behind this algorithm consists of approximating the objective function with a quadratic equation and using the minimum of the approximation function as a possible solution in the next iteration. To improve the efficiency, the SLSQP can use Quasi-Newton methods to approximate the objective Hessian function.

The code provided uses the SLSQP implementation available in the SciPy module [108], which is a library of numerical routines for the Python programming language, which provides the fundamental building blocks for modeling and solving scientific problems.

5.4. Trust-Constr

Trust-constr is a trust region algorithm for constrained optimization, available in the SciPy module [108]. Both line search methods and trust region methods generate steps with the help of a quadratic model of the objective function. However, while line search

methods use a step length along a given search direction, trust region methods define a region around the current iteration, within which they trust that the quadratic model is an accurate approximation [109].

This algorithm utilizes the trust region interior point method described in [110] to solve the inequality constraints imposed in the topology optimization problem statement. The strategy behind this interior point method [110] consists of solving inequality constraints by introducing slack variables. Then, the process is repeated for a sequence of equality-constrained barrier problems with progressively smaller values of the barrier parameter.

6. Python Implementation and Usage

The following subsections present a brief guide on how to use the Python code provided with the following Dataset [94] or repository: https://github.com/pnfernandes/Python-Code-for-Stress-Constrained-Topology-Optimization-in-ABAQUS. A brief overview of the most-important details of the main code classes and functions is also included in these subsections. The detailed information of each class and function can be found in their respective doc-string. The sequence of operations performed by this code implementation is described in Figure 1.

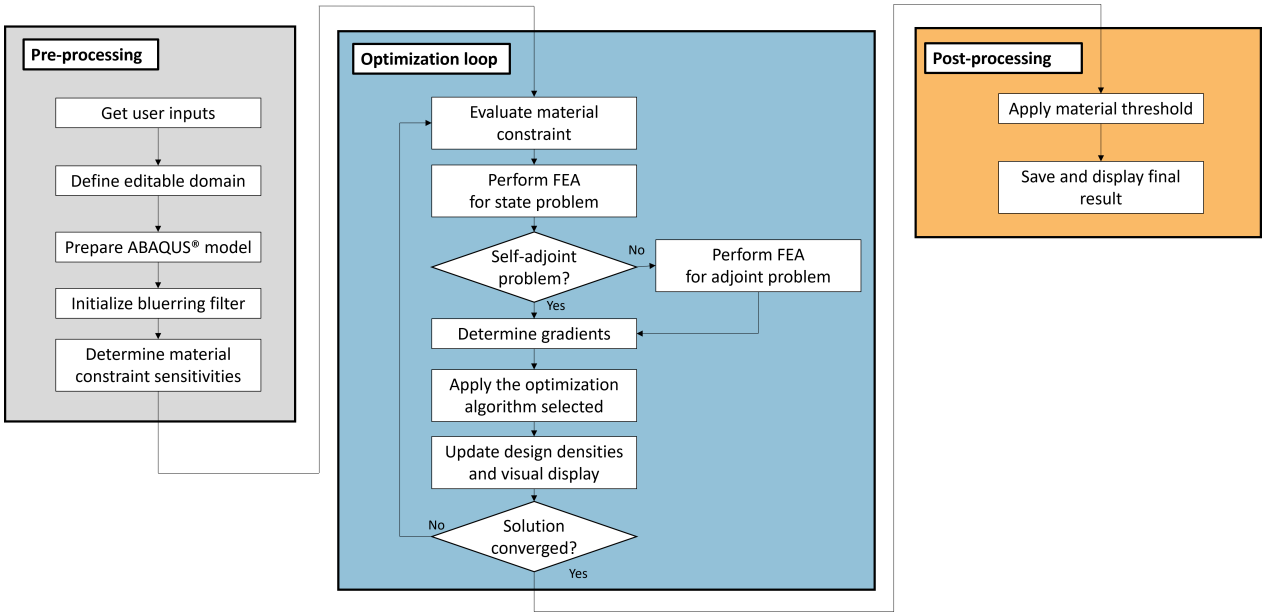

Figure 1. Sequence of operations performed by the topology optimization code implemented in this manuscript.

This code can be called in ABAQUS® using the *Run Script* command in the *File* tab or by copying it into the command line. Note that the Run Script command loads the code faster, but automatically assumes that the user intends to solve the topology optimization in its totality. Copying it into the command line leads to a slower input, but allows a line-by-line execution, which may be useful for understanding the functioning of the code implementation.

The PEP8 [111] Python style guide was followed as closely as possible. However, the interested reader is warned that some exceptions exist, caused by compatibility constraints with the data structures and pre-defined variables existing in ABAQUS®. An example of these exceptions includes the naming of constant material properties without all capital letters, since these names are already used by ABAQUS® in a different data structure.

6.1. Code Usage

Upon executing the code, ABAQUS® will create a series of prompt dialogue boxes. This allows the user to select the model database file (.cae), structural part, material, problem

statement, optimization algorithm, and internal parameters to be considered during the topology optimization.

Besides being fully functional, the ABAQUS® model does not require any specific formatting, except for the following considerations:

- Within the same part, the user can define specific elements to be optimized, dividing the part into two regions: one with editable elements and another with elements that should not be included in the optimization process (i.e., passive elements, from this point onward referred to as "frozen region" or "frozen elements"). To do so, the user should create a set named "editable_elements", containing the elements to be edited. If this set does not exist, the whole part will be optimized. Note that the name "editable_elements" is case sensitive. This is performed in order to promote the optimization only of the targeted area and also to reduce the computational cost.
- The region to be optimized should not have a material section assigned to it, as the code will automatically assign the material properties and sections directly to the elements. Therefore, if the user defined a set with the editable elements, only the elements not included (frozen elements) should have a material section assigned to them.
- It is possible to allow the frozen elements to be considered during the filtering operations. To do so, the user should create a second element set with the case-sensitive name "neighbouring_region".
- If multiple copies of the part to be optimized are included in the model assembly, only the first copy will be optimized.

6.2. Model Formatting, Job Submission, and Sensitivities (Lines 28–3452)

The *ModelPreparation* class modifies the ABAQUS® model file (.cae) in order to allow the assignment of material properties and material sections to each individual element, as well as the property update during each iteration. Furthermore, this class requests the output of the strain energies, external work, and node strains if solving a stress-dependent problem. In ABAQUS®, the strain energies of geometrically nonlinear problems are stored in the variables "SENER" and "PENER", respectively, for the elastic and plastic strain components. Note that, according to the literature [14,17,98,102,112], the sensitivity of each element in a geometrically nonlinear compliance-minimization problem is given by the sum of the elastic strain (E_e^e) and plastic strain components (E_e^p), as described in Equation (67). However, for geometrically linear problems, the plastic strain component is zero, and ABAQUS® stores the strain energies in the variable "ESEDEN". The external work is only requested for a critical analysis of the result, as it should be equal to the objective function considering no energy dissipation.

$$\frac{\partial C}{\partial \rho} = E_e^e + E_e^p. \tag{67}$$

The submission of the FEA simulations for the state and adjoint models is managed by the classes *ABAQUSFEA* and *AdjointModel*, respectively. *ABAQUSFEA* is also responsible for determining the sensitivity of the compliance objective function $\left(\frac{\partial C}{\partial \rho}\right)$, which is stored as the class attribute "ae". On the other hand, the *AdjointModel* class also applies the adjoint loads and determines the stress sensitivity. The stress sensitivity $\left(\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}\right)$, and its most-relevant intermediate terms ($\partial \sigma_{spf}^{PN}(\rho)$ and $\partial \sigma_u^{PN}(\rho)$) are stored as attributes of this class, along with the stress and strain vectors. In particular:

- "elmt_stress_sensitivity_continuous" stores the values of $\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}$ in a mesh-independent and mesh-independent form, making the comparison of the value of this derivative more suitable for the results obtained through finite differences.

- "elmt_stress_sensitivity_discrete" stores the values of $\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}$ in a mesh-dependent form, which is more suitable to be used as an input to the optimization algorithms, since they generally do not have access to the information of the mesh used in the FEA model. The difference between discrete and continuous element stress sensitivity is the multiplication by the determinant of the Jacobian matrix.
- "d_pnorm_spf" stores the values of $\partial \sigma_{spf}^{PN}(\rho)$.
- "d_pnorm_displacement" stores the values of $\partial \sigma_{u}^{PN}(\rho)$.
- "stress_vector" and "stress_vector_int" store the stress vectors determined at the element nodes and integration points.
- "deformation_vector" and "deformation_vector_int" store the strain vectors determined at the element nodes and integration points.

The sensitivity of the material constraint is determined by the *material_constraint_sensitivity* function, accounting for the possibility of having a non-uniform mesh with elements of different sizes and the use of either a volume or mass constraint.

6.3. Material and Stress Constraints (Lines 3453–3607)

The values of the material and maximum stress constraints are determined by the *MaterialConstraint* class and the *stress_constraint_evaluation* function, respectively. The maximum stress value is approximated with the modified p-norm function described in Equation (47), which is determined by the *p_norm_approximation* function.

6.4. Data Filtering (Lines 3608–3846)

The blurring filter is defined by the *DataFilter* class, which determines how each element is influenced by its neighborhood. The neighborhood of each element is defined by the elements that are fully within a maximum search radius defined by the user. The code Lines 3760–3762, intentionally left commented to reduce computational cost, generate element sets that allow an easy visual interpretation of the neighborhood of each element.

6.5. Optimization Algorithms: Optimality Criteria, Method of Moving Asymptotes, Sequential Least-Squares Quadratic Programming, and Trust-Constr (Lines 3847–5622)

The present code provides five optimization algorithms, described in Section 5, which can be used to solve the topology optimization problem.

The discrete and continuous versions of the OC algorithm, implemented in the functions *oc_discrete* and *oc_continuous*, are only suitable for compliance-minimization problems. The remaining optimization algorithms are suitable for all problem statements included.

The *mma* function is a decorator that links ABAQUS® to the *mmasub* and *subsolv* functions implemented by Arjen Deetman [106]. Therefore, the main purpose of the *mma* function is the processing of the inputs necessary to apply the MMA [107] as a function of the problem statement selected by the user.

The functioning of the *oc_discrete*, *oc_continuous*, and *mma* functions is managed by the main program loop, described in Section 6.11. However, this is not the case for the SLSQP and Trust-constr algorithms available in this code, as they belong to the SciPy module [108]. To be fully functional, the SLSQP and Trust-constr algorithms require the additional freedom to decide when and how many times to call the objective, sensitivity, and constraint functions. Furthermore, the problem statement must be defined using unique data structures. For these reasons, class *SciPyOptimizer* is responsible for:

- Calling the SLSQP and Trust-constr methods from the SciPy module [108].
- Re-defining the problem statement with the correct data structure required by the algorithm selected.
- Providing the freedom for these algorithms to call the objective, sensitivity, and constraint functions as many times and in any given order necessary.

- Trying to record the data of each iteration as accurately as possible. However, it is to be expected that the data recorded may include information from intermediate evaluation points.

Finally, there are two details that should be considered before using the SLSQP and Trust-constr algorithms. One is that both have internal convergence criteria. To change them, the user may be required to edit the code provided with this article. The second detail is that the initial solution must be feasible; otherwise, the optimization process will stop. This is particularly relevant when defining the initial material distribution and/or when using a p-norm continuation approach (described in Section 6.11). The OC and MMA methods implemented do not have these restrictions for the initial guess.

6.6. Display Definition (Lines 5623–6207)

The *SetDisplay* class is capable of changing the color codes of the element sets. In compliance-minimization problems, this allows the user to plot a gray-scale representation of the material distribution. In stress-dependent problems, the user has the additional options of plotting the distribution of the element stress or element-amplified stress, both of which can be raised to the p-norm exponential factor. The element stress plotted is equal to the average of the stresses determined at the integration points of the element.

The graphic representation(s) plotted at each iteration are automatically saved to a .png file.

To allow an easier interpretation of the 3D material distributions, the *SetDisplay* class has a built-in method (function of a class) named *hide_elements*, whose input is a design variable threshold value. This method will hide all elements whose design variable is below the threshold input.

6.7. Data Recording (Lines 6208–6349)

At the end of each iteration, the *save_data* function will create a text file that records the values of all variables necessary to describe the current solution, as well as the internal parameters selected by the user.

This text file can also be used to restart the topology optimization process, starting from any iteration. However, as mentioned in Section 6.5, note that the algorithms SLSQP and Trust-constr may require the current iteration to be feasible with respect to the constraints imposed.

At the end of the optimization process, *save_mdb* will create a separate ABAQUS® .cae file with the final solution obtained and a record of the objective function and material constraint values.

6.8. Element Formulation and Stiffness Matrix (Lines 6350–7283)

To solve stress-dependent problems, it is necessary to have access to information that depends on the element formulation (as shown in Section 3.4), such as B_a, K, $\mathbb{C}$, and the Jacobian matrix, as well as the definition of the element shape functions and their derivatives. The code provided includes the information required to use one 2D element with 4 nodes and one 3D element with 8 nodes. The 2D element is referred in ABAQUS® by the code 2DQ4 and, more specifically, as CPS4 or CPE4 for the plane–stress and plane–strain cases, respectively. The 3D element is referred in ABAQUS® by the code C3D8. The information of these elements is included in the *ElementFormulation* class, while the stiffness matrix $\mathbb{C}$ is created by the *c_matrix_function* function.

Although not used in the present article, the *ElementFormulation* class also includes the formulation of a shell element (referenced by the code S4 in ABAQUS®). This information was included in order to promote the development of topology optimization methodologies suitable to this particular type of element.

6.9. Parameter Input Request, Domain Definition, and Variable Generation (Lines 7284–8854)

The *ParameterInput* class generates the prompt boxes that appear when running the code. The information introduced by the user is then used to create the global variables required for the code. A detailed list of these variables and their purpose can be found in Lines 8366–8461.

This information is then used by the *EditableDomain* class, to define the design space of the topology optimization problem, and by the *VariableGenerator* class, to create the lists and dictionaries that record the relevant data gathered during the topology optimization process. For a detailed description of these variables, please refer to the code Lines 8721–8827.

6.10. Auxiliary Functions (Lines 8855–9013)

Finally, there are four auxiliary functions to be briefly mentioned:

- *average_ae* outputs the average of the compliance sensitivity of each element over the last three iterations. This leads to an easier convergence of the OC algorithms, especially when considering discrete variables.
- The *update_past_info* function updates the variables that record the design variables and compliance sensitivity used in the previous two iterations.
- The *evaluate_change* function creates a ratio that describes how the objective function has changed over the last 10 iterations, which is used as convergence criteria in the implementation provided.
- The *remove_files* function will allow an automatic removal of the temporary files created by ABAQUS® after each FEA.

6.11. Main Program (Lines 9014–9307)

The main program is divided in two phases. Between Lines 9014 and 9110, the code creates the classes required for the problem statement selected and prepares the ABAQUS® model accordingly. The optimization process and the convergence criteria are defined in Lines 9111–9307.

The optimization process considers two loops. The first is applicable for stress-dependent problems, allowing a continuous increase of the p-norm exponential factor between Qi and QF. This functionality, here referred to as the "p-norm continuation approach", allows the stress constraint to be applied gradually and may be useful to improve the convergence of the algorithm. The continuation approach contrasts with the constant approach, which only considers one constant value for the p-normfactor. Note that for stress-independent problems (where Qi is not used, but set to 1.0) and when using the constant approach, $Qi = QF$, leading to a single loop.

The second loop represents the convergence criteria. The code assumes that the algorithm has converged if the objective function has not changed more than 0.1 % over the last 10 iterations. It was selected as the default convergence criteria to allow an easy and relatively fast recreation of the results detailed in this article, as well as for being similar to the criteria adopted in [17]. The interested user is reminded that these criteria do not apply to the SLSQP and Trust-constr algorithms (as mentioned in Section 6.5) and is encouraged to create convergence criteria adapted to the topology optimization problem analyzed.

7. Benchmark Problems and Case Study

7.1. Cantilever Beam

The purpose of including this problem was to validate the functioning of the compliance objective function and volume constraint implemented in the code provided, before addressing more-complex stress-dependent problems, such as the L-bracket detailed in Section 7.2.

This problem was modeled using a regular mesh with an element size of 5.0 mm and the element type CPS4. The load F was equal to 100.0 N, applied to a single node, in the corner represented in Figure 2. The problem was solved using an implicit analysis and

considering a material with a 70.000 MPa Young's modulus, a 0.33 Poisson's ratio, and a material density of 2.7×10^{-9} t/mm^3.

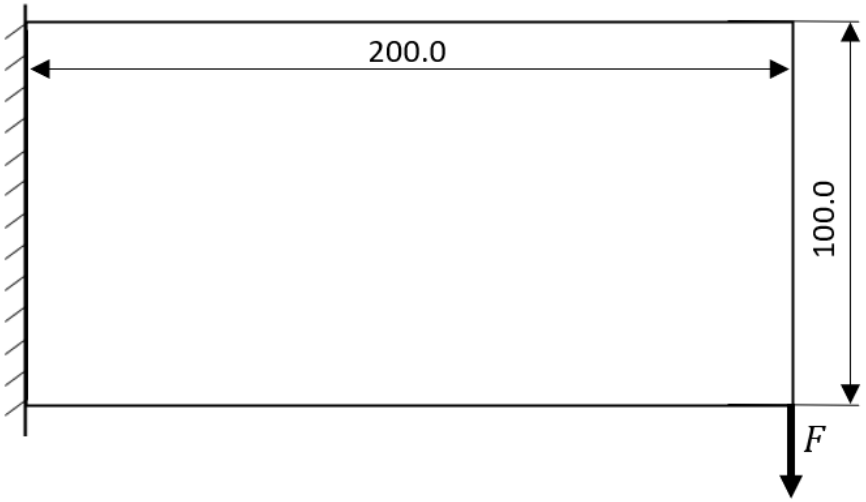

Figure 2. Dimensions and boundary conditions of the Cantilever beam numerical model.

The ABAQUS® models necessary to reproduce all the results described in this work are available in the Dataset [94].

7.2. L-Bracket

The L-bracket was chosen as a more-challenging benchmark problem [53,113,114], characterized by its initial geometry containing a stress-concentration point in the internal corner. Unconstrained compliance-minimization problems tend to ignore the stress concentration point, leading to an angular shape. On the other hand, stress-minimization or stress-constrained compliance-minimization problems tend to create rounded shapes that avoid the stress concentration. For these reasons, the L-bracket problem will be used in this research to validate and demonstrate the functioning of the code implemented when solving stress-dependent topology optimization problems. Furthermore, the strain and stress state generated by this L-bracket geometry is used in Section 8 to validate the correct implementation of the element formulations and maximum stress differentiation process.

The dimensions and boundary conditions considered are represented in Figure 3. Unless stated otherwise, this problem was modeled using a regular mesh with an element size of 3.0 mm and the element type CPS4. The load F was equal to 1500.0 N, with this value being distributed over the nodes included in the 12.0×12.0 corner represented in Figure 3. The elements within the 12.0×12.0 corner are not editable during the optimization process. The problem was solved using an implicit analysis and considering a material with a 70.000 MPa Young's modulus, a 0.33 Poisson's ratio, and a material density of 2.7×10^{-9} ton/mm^3.

7.3. Bonded Support

The problem described in Figure 4 was selected to provide a three-dimensional case study that is more representative of the complex geometries that can be found in engineering design problems.

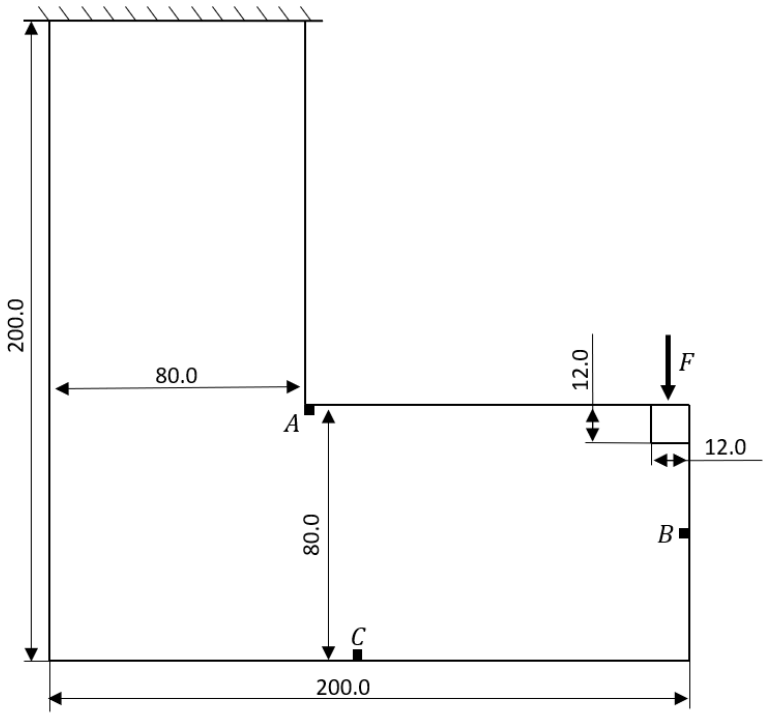

Figure 3. Dimensions and boundary conditions of the L-bracket numerical model.

The design space was modeled using an irregular mesh with an average element size of 1.25 mm and hexahedral C3D10, leading to an approximate total of 370.000 elements. A unit load was applied at a reference point attached to the connection region of the structure, while the surface of the adhesive joint was considered fixed. To minimize the computational cost, two planes of symmetry were considered along the XY and YZ planes, reducing the total number of elements. The problem was solved using an implicit analysis considering a material with: a 70.000 MPa Young's modulus, a 0.33 Poisson's ratio, and a material density of 2.7×10^{-9} ton/mm^3. The problem also considered a simplified representation of an adhesive bond, accounting only for the elastic properties (a 1.500 MPa Young's modulus and a 0.3 Poisson's ratio). Due to the complexity of the design, the interested reader is referred to the Mendeley Dataset [94] for the complete description of the geometry considered in this case study and shown in Figure 4.

The material elements representing the adhesive bond are identified as "frozen elements". This excludes the possibility of editing the material distribution in this region, which would influence the resistance of the joint due to the reduction of the bonded area and excludes the contribution of these elements during the use of the blurring filter, which would be physically unreasonable as the adhesive and substrate are made of different materials. On the other hand, the first layer of elements of the support in contact with the adhesive was identified as a "neighboring region", which does not allow changes in the material distribution of these elements, but allows the consideration of their influence in the determination of the gradients. However, unlike the previous case, the elements in this region were considered during the filtering process, as the material is the same as the one assigned to the editable region of the topology optimization process. The "frozen elements" and "neighboring region" are shown in Figure 4 in red and blue, respectively.

The ABAQUS® models necessary to reproduce all the results described in this work are available in the Dataset [94].

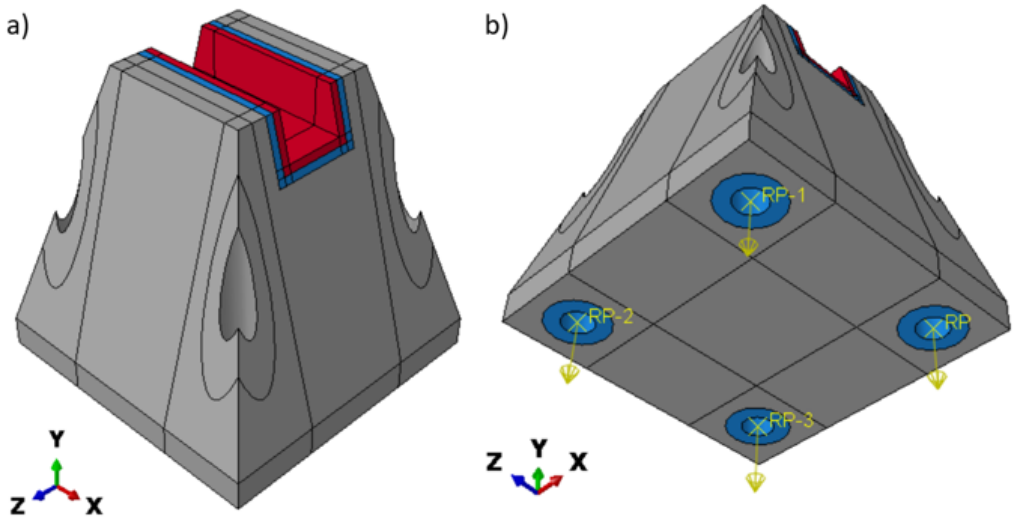

Figure 4. Complete three-dimensional representation of the geometry considered for the bonded support and its boundary conditions. The exterior surface of the region shown in red is considered fixed. The complete geometry is provided in the Mendeley Dataset [94]. (**a**) Detailed view of the bonded region. (**b**) Detailed view of the loads applied in the lower surface.

8. Code Validation

The data necessary to solve compliance-minimization problems can be directly obtained from ABAQUS®, which is a commercially certified software. However, stress-dependent problems require the explicit programming of the element formulation and of the derivation process of the maximum stress function. The purpose of the following subsections is to present a brief validation of the code implementation. Section 8.1 validates the element formulations implemented, comparing the strains and stresses determined by the code and by ABAQUS® at each integration point. Section 8.2 compares the derivative of the maximum stress determined by the code with the derivative obtained through finite differences.

In this section, the L-bracket benchmark problem described in Section 7.2 is used. The reason behind this choice is that the L-bracket problem leads to a complex strain and stress state, allowing a more-challenging and -generic testing of elements over a wider variety of loading conditions than the Cantilever beam problem.

8.1. Validation of the Element Formulation

The analysis of the element formulation is based on the measurement of a pondered error, described as follows:

$$E_{rr}^{i} = \delta\gamma_i \frac{\gamma i}{|\gamma|}. \tag{68}$$

where E_{rr}^{i} is the pondered error determined for the component i of a generic vector γ and $\delta\gamma_i$ is the difference between the component determined by ABAQUS® and its code counterpart. E_{rr}^{i} is determined as the product of $\delta\gamma_i$ by the ratio between the component γ_i and the magnitude of the vector γ.

This metric is preferred over the direct comparison of $\delta\gamma_i$, since the latter is highly influenced by floating point errors. One of the reasons for this influence is that, while ABAQUS® operates internally with double-precision, the output received by the code is in a single-precision format. This difference reduces the number of decimal places considered in each value and could lead to disproportional artificial errors in the vector components that contribute the least to the magnitude of the vector or in vectors whose magnitude tends towards zero. The second reason is that $\delta\gamma_i$ tends to estimate large error values for

vector components that have a significantly smaller magnitude when compared to the magnitude of the vector.

Table 2 summarizes the average pondered error observed for each integration point of four variants of the L-bracket numerical model described in Section 7.2, each with a different element type. For the particular case of using a 3D element (element type C3D8 in ABAQUS®), a thickness of 1.0 mm was considered for the L-bracket geometry. Since the largest average pondered error observed is equal to 2.0%, it can be concluded that the element formulations included in the code provided were implemented successfully for the case selected. Furthermore, for each element type, the table indicates the percentage of integration points with a pondered error less than 1.0%.

Table 2. Average pondered error between the elements implemented in the code and the ABAQUS® output. The dimensions are represented in millimeters.

	CPS4		**CPE4**		**S4**		**C3D8**	
	E^i_{rr} **(%)**	**Int. Points** E^i_{rr} **< 1% (%)**	E^i_{rr} **(%)**	**Int. Points** E^i_{rr} **< 1% (%)**	E^i_{rr} **(%)**	**Int. Points** E^i_{rr} **< 1% (%)**	E^i_{rr} **(%)**	**Int. Points** E^i_{rr} **< 1% (%)**
ξ_{11}	0.05	99.65	0.90	63.39	0.49	97.29	0.44	96.37
ξ_{22}	0.15	99.55	0.97	63.31	0.44	97.60	0.44	96.37
ξ_{33}	-	-	-	-	-	-	0.44	96.37
ξ_{12}	0.04	99.51	0.04	99.51	1.95	32.45	<0.01	100.0
ξ_{13}	-	-	-	-	-	-	<0.01	100.0
ξ_{23}	-	-	-	-	-	-	<0.01	100.0
σ_{11}	0.00	99.88	1.85	31.74	0.61	95.89	1.45	44.81
σ_{22}	0.15	99.65	2.00	63.41	0.61	97.29	1.45	96.36
σ_{33}	-	-	-	-	-	-	1.45	44.82
σ_{12}	0.01	99.55	0.01	99.55	0.77	77.44	<0.01	100.0
σ_{13}	-	-	-	-	-	-	<0.01	100.0
σ_{23}	-	-	-	-	-	-	<0.01	100.0

Although not included in the present work due to its extension, the interested reader is informed that the Dataset [94] associated with this research includes a detailed comparison of every integration point, considering both the pondered and regular error measurements.

8.2. Validation of the Maximum Stress Derivative

To validate the stress norm derivative of the code provided, its result was compared with the derivative obtained through finite differences. To do so, the design density of the elements located at points A, B, and C (shown in Figure 3) were changed individually, and the value of the maximum stress approximation was used to apply the finite differences. This procedure was repeated for the three elements, considering three different mesh sizes of 3.0 mm, 1.0 mm, and 0.5 mm, each increasing the order of magnitude of the total number of elements by 1, and different design density changes of 0.1, and 0.01. These density changes were chosen to conform with the maximum stiffness element difference allowable by ABAQUS®. The results obtained are summarized in Table 3.

It can be observed that the results obtained through finite differences are in good agreement with the derivative obtained by the Python code implemented, especially when the mesh size and the design density decrease.

Table 4 indicates the average values of $\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}$ and its components $\partial \sigma^{PN}_{spf}(\rho)$ and $\partial \sigma^{PN}_{u}(\rho)$ observed in the L-bracket model as a function of the element mesh size and design density. These data indicated that the approximation proposed in [100] (described at the end of Section 3.4) should not be adopted for the present problem.

Table 3. Analysis of the continuous maximum stress derivative determined by the Python code and the derivative obtained through finite differences.

Mesh Size	Number of Elements	p-Norm (MPa) @ ρ = 1.0	Element Location	Element Label	$\Delta\rho$	p-Norm (MPa) @ $\rho = 1 - \Delta\rho$	Finite Differences	Continuous Derivative
0.5	102,400	314.27741	A	69,184	0.1	335.12679	−208.4939	−129.6819
					0.01	315.23590	−95.8493	
			B	102,332	0.1	314.27740	0.0000	0.0001
					0.01	314.27741	0.0000	
			C	63,841	0.1	314.28104	−0.0363	−0.0301
					0.01	314.27784	−0.0438	
1	25,600	272.96255	A	17,269	0.1	272.87736	0.8519	0.4426
					0.01	272.95502	0.7534	
			B	25,560	0.1	272.96249	0.0006	0.0004
					0.01	272.96255	0.0000	
			C	23,901	0.1	272.97007	−0.0752	−0.0685
					0.01	272.96326	−0.0713	
3	2889	221.08948	A	1953	0.1	232.36352	−112.7404	−65.2413
					0.01	221.48312	−39.3637	
			B	2875	0.1	221.08923	0.0025	0.0029
					0.01	221.08941	0.0066	
			C	2683	0.1	221.13180	−0.4232	−0.4054
					0.01	221.09332	−0.3842	

Table 4. Average value of the stress derivative, and its components, observed in the L-bracket model at different design density values, considering three different mesh sizes.

Mesh Size	ρ	$\frac{\partial \sigma^{PN}(\rho)}{\partial \rho}$	$\partial \sigma^{PN}_{spf}(\rho)$	$\partial \sigma^{PN}_{u}(\rho)$	$\frac{\lvert\partial \sigma^{PN}_{u}(\rho)\rvert}{\lvert\partial \sigma^{PN}_{spf}(\rho)\rvert}$
0.5	1	−0.0088	0.0004	−0.0092	24.037
	0.5	−2304.3703	0.5426	−2304.9129	4248.295
	0.1	−90.9833	1.2132	−92.1965	75.996
1	1	−0.0305	0.0013	−0.0318	23.864
	0.5	−7949.4335	1.8849	−7951.3184	4218.445
	0.1	−313.8380	4.2147	−318.0527	75.462
3	1	−0.2140	0.0096	−0.2236	23.289
	0.5	−3.4697	0.1086	−3.5784	32.936
	0.1	−2206.1706	30.3676	−2236.5382	73.649

9. Topology Optimization Results

Figure 5 summarizes the results obtained for the compliance minimization of the Cantilever beam. Figure 6 summarizes the results obtained for the topology optimization of the L-bracket considering the different problem statements, plotting the objective function at each iteration and illustrating one of the final geometries obtained. Finally, Figure 7 shows the topology-optimized geometry of the bonded support. The color code adopted here displays elements with $\rho = 0$ in white, intermediate densities in grey, and elements with $\rho = 1.0$ in black. These results were not adimensionalized.

The results obtained for each problem statement and any particular choice of parameters are discussed in the following subsections. In common, these examples share the volume constraint $V^* = 0.5$. In the particular case of the solution "OC—Discrete and decreasing", the initial density was set to 1.0 and gradually reduced by a factor of

$e_{vol} = 0.05$, while the remaining cases considered an initial density equal to 0.5. The maximum move limit was equal to 0.2 for the OC or MMA algorithms and 1.0 for the SciPy algorithms SLSQP or Trust-constr, corresponding to their pre-default values. The blurring filter was applied to both the sensitivity and design densities with a maximum search radius of 8.0 mm for the L-bracket, 12.0 mm for the Cantilever beam, and 3.5 mm for the bonded support, except for the discrete version of the OC, which only applied the blurring filter to the sensitivity. The influence of the frozen elements was also considered during the application of the blurring filter. For stress-dependent problem statements, a p-norm exponential factor $Qi = 8.0$ was used.

Overall, the Trust-constr and SLSQP algorithms tended to require a larger number of iterations to reach convergence. The reason for this difference was justified by their internal convergence criteria, as described in Section 6.11.

In the results shown, the existence of few elements with intermediate design densities is, generally, to be expected for two reasons. First, applying a blurring filter will cause a gradual transition between the solid and void regions, avoiding a full "black-and-white" solution. Second, as described in Section 6.11, the convergence criterion depends only on the values of the objective function and does not impose a strict restriction on the final solution being constituted only by solid elements. These two factors justify the apparent better performance of the discrete version of the OC algorithm, since it only uses solid elements and does not have a blurring filter applied to the element densities. However, it is most likely that this optimization method converged to a better local minimum.

It is also relevant to note that the mesh size had an influence on the use of elements with intermediate design densities. To completely avoid the existence of residual grey areas, it is recommended to implement projection methods [103], to define the final contour of the geometry, or to apply a final threshold filter to the design densities of each element. However, despite the potential influence of internal parameters, such as the filtering approach, mesh refinement, and the choice of the optimization algorithm and its hyperparameters, the results presented in both case studies are in line with the information reported in previous publications [53].

Regarding the computational efficiency of the implementation proposed here, it was seen that it requires an average of 50 s per iteration when executed with an Intel® Core™ i7-8750H CPU @ 2.20 GHz. However, the use of external packages, to apply the SLSQP and Trust-constr algorithms available in Python, led to an increase of this value to an average of 380 s per iteration. These values include the whole process required to perform an iteration, including: the update of the material properties, the preparation of the ABAQUS® input file, model execution, data extraction from the FEA solution, the calculation of gradients, the application of the optimization process to update the design densities, and the plot of the new material distribution.

Finally, the Cantilever beam and L-bracket problems solved in these sections could also be replicated in 3D space. This is possible as the code provided includes the implementation of a 3D element (C3D8 in ABAQUS®) and as the problem solving process does not require any particular adaptation when being converted to 3D space.

The interested reader is informed that the Dataset [94] contains all the solutions obtained by each optimization algorithm, at each iteration. Due to its large extension, the information presented in this section is limited to the most-relevant solutions.

9.1. Cantilever Beam: Compliance Minimization Results

The objective functions obtained by each of the different optimization algorithms were in good agreement, converging towards an approximately equal value of compliance. However, a different trend can be observed for the "OC—discrete and decreasing" case. In this particular case, the initial solution considers a fully solid design that must gradually reduce its mass to meet the volume constraint, causing a consequent reduction of the structural stiffness and an increase of the compliance value.

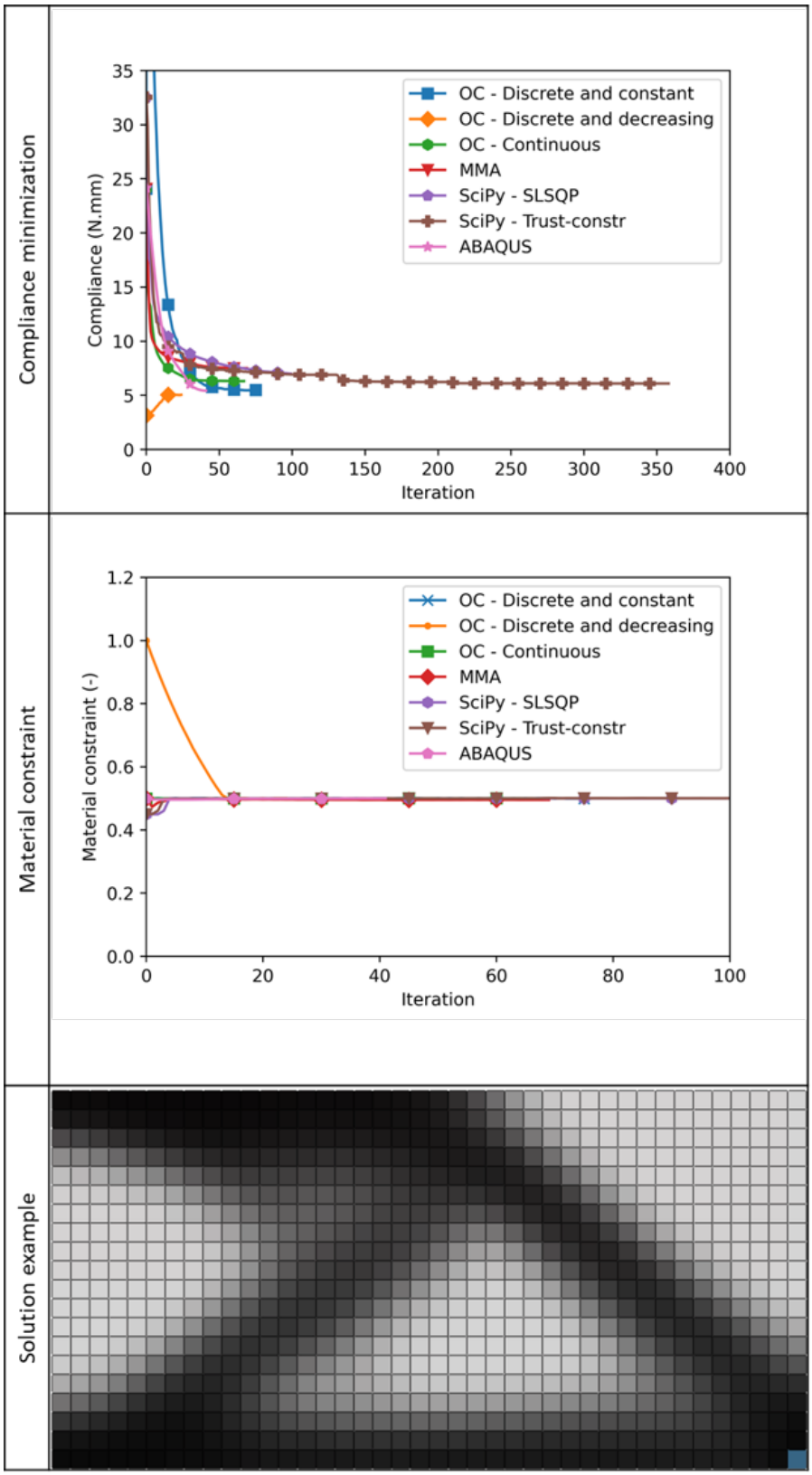

Figure 5. Graphic representation of the objective function and volume constraint obtained for the compliance minimization of the Cantilever beam. The geometry displayed represents the final solution obtained by the MMA algorithm.

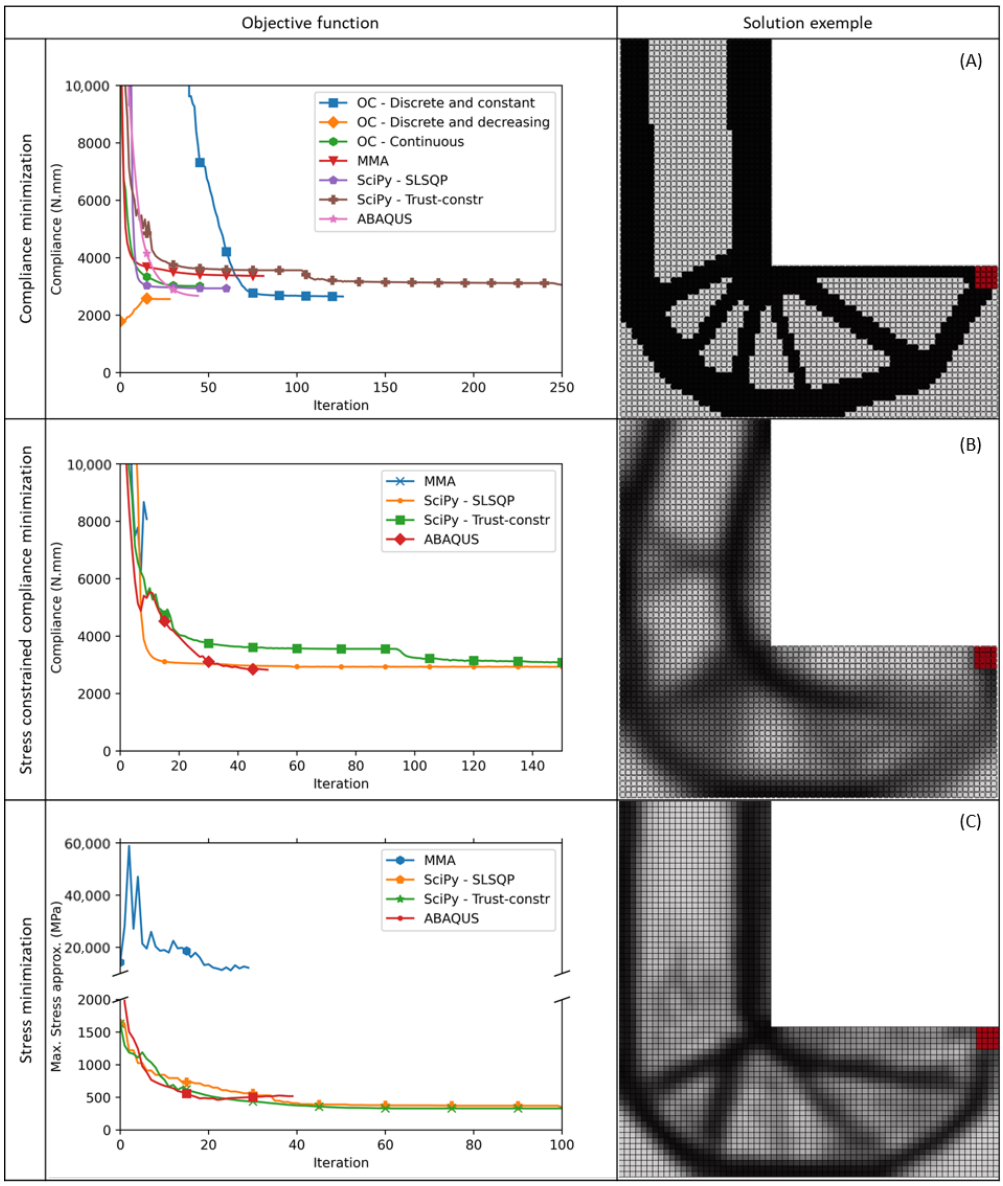

Figure 6. Graphic representation of the objective function obtained for each optimization algorithm and problem statement. The geometries displayed represent the final solution obtained by the algorithms: (**A**) OC—Continuous, (**B**) MMA, (**C**) SciPy SLSQP.

This observation is also in line with the volume constraint graphic, which shows a gradual decrease until the value 0.5 is reached. On the other hand, the remaining algorithms maintained the volume constraint within a feasible domain.

The final Cantilever geometry obtained was similar across all cases considered and illustrated, in Figure 5, with the solution found by the MMA. In this particular case, the elements with intermediate densities may also be the result of using a coarser mesh.

9.2. L-Bracket: Compliance Minimization Results

The solutions obtained for the compliance-minimization problem were similar across all the algorithms used, differing only in the use of intermediate design densities in some elements. The solution obtained by the continuous version of the OC is shown in Figure 6A as an example of this geometry.

The volume constraint was respected across all iterations. This behavior was also observed for the problem statements presented in Sections 9.3 and 9.4. For this reason, the graphic representation of the constraints was not included in these sections.

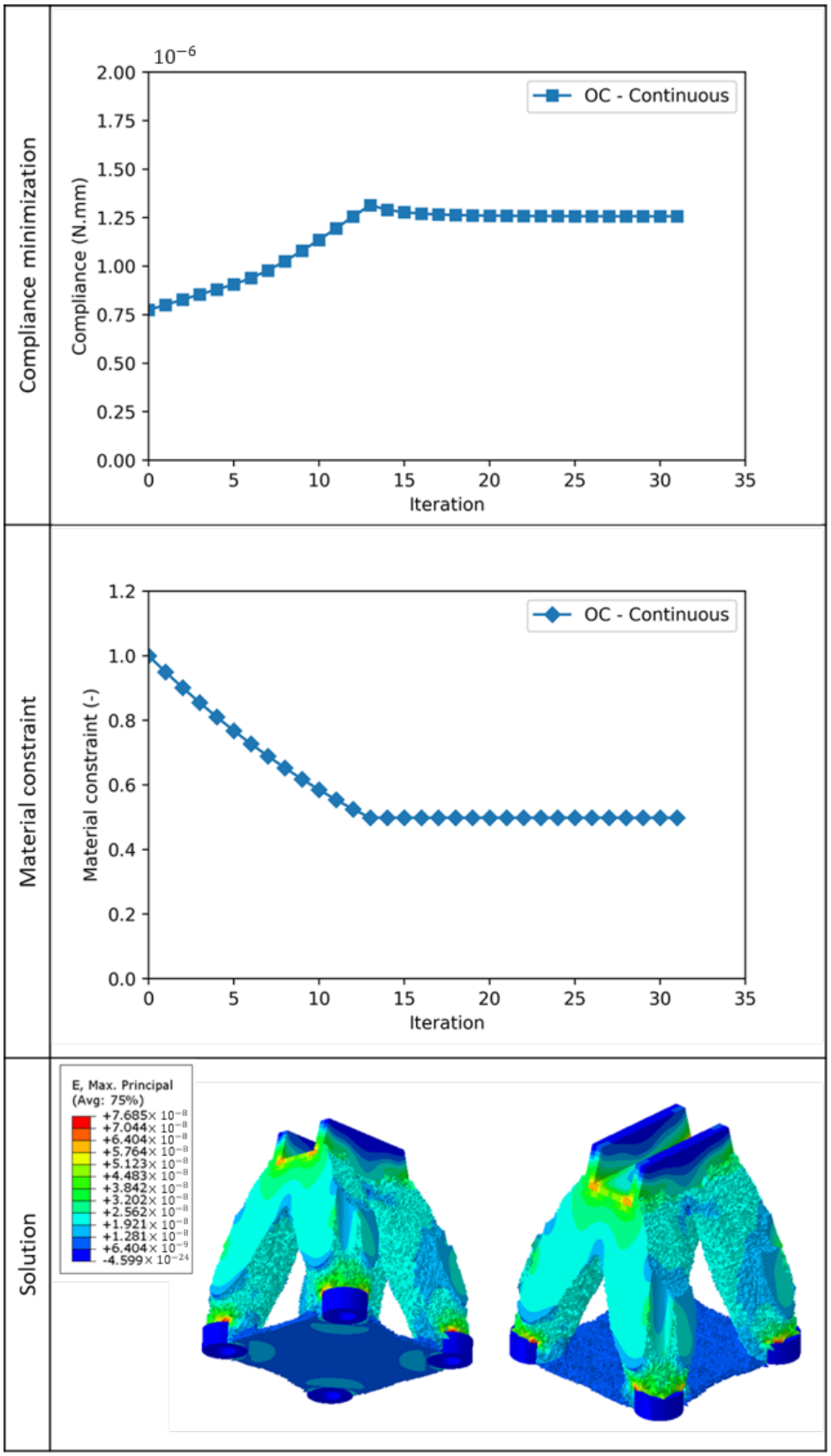

Figure 7. Graphic representation of the topology optimized geometry, objective function, and material constraint obtained during the compliance minimization of the bonded support. The geometry displayed represents the final solution obtained by the OC—Continuous algorithm. The image shown only includes elements with a design density $\rho = 1.0$ to improve the readability and comprehension of the solution obtained.

9.3. L-Bracket: Stress-Constrained Compliance Minimization Results

The stress-constrained compliance-minimization problem statement considered a maximum allowable stress $\sigma^* = 200.0$ MPa. Imposing this constraint caused the MMA to converge towards a curved geometry, shown in Figure 6B, avoiding the creation of a stress concentration point in the inner corner of the L-bracket. This transition is in agreement with the results presented in [53].

It is important to note that the transition from an angular geometry to a curved geometry (from case A to B of Figure 6) is dependent on several factors and not guaranteed to occur. First, the use of the modified p-norm approximation (Equation (47) proposed in [53]) may lead to an underestimate of the maximum stress installed. This fact may cause the optimization algorithm to overlook the influence of the stress concentration point and justifies the selection of $\sigma^* = 200.0$ MPa, a relatively low maximum value for the stress constraint. Second, the algorithm selected may adopt a strategy that is more or less prone to explore and accept solutions that do not respect the imposed constraints. The solution obtained by the SLSQP algorithm had a slightly curved shape closer to the region where the load was applied, but did not avoid the internal angular geometry. The solution obtained by the Trust-constr region was equal to the one obtained in the compliance-minimization problem statement, as the implementation available in SciPy only determined the maximum stress sensitivity for the first iteration. Although this strategy reduced the computational cost, it did not allow a rigorous evaluation of the influence of each element in the maximum stress.

Finally, it is also important to note that the p-norm exponential factor had a significant influence on the result obtained, especially in the use of elements with intermediate densities.

9.4. L-Bracket: Stress Minimization Results

Defining a stress-minimization problem statement may also lead to a transition from an angular to a curved geometry. This is particularly evident in the solution obtained by the SLSQP algorithm, shown in Figure 6C. However, this transition did not occur with the method available in ABAQUS®. As a result, the SLSQP and Trust-constr algorithms converged to solutions with maximum stresses of 315 MPa and 330 MPa, respectively, representing a 35% reduction when compared to the 518 MPa installed in the solution obtained with ABAQUS®.

The data obtained with the SLSQP indicated an unstable behavior with large variability in the objective function. To improve readability, only the cumulative best solution obtained with this method was included in the plots referring to this problem statement. This variability indicates that the SLSQP was more likely to perform larger density redistributions, which may be a result of the strategy adopted by the SciPy implementation of this algorithm, and a result of the influence of their internal parameters, which largely affect the solutions obtained.

Similar to the previous problem statement, the p-norm exponential factor had a significant influence on the result obtained, especially in the use of elements with intermediate densities. Also, although the MMA algorithm was removing material from the stress concentration area, further research should be performed to find appropriate tolerance parameters.

9.5. Bonded Support: Compliance Minimization Results

The application of the compliance-minimization problem statement to the bonded support allows the removal of material in the central region, leaving material in the connection between the adhesive joint and the elements sustaining the external loads and in the lower surface of the design domain. To improve the readability and comprehension of the geometry obtained, Figure 7 only displays the elements where $\rho = 1.0$. The results herein shown also included the deformations installed in the topology-optimized design and the complete geometry obtained by mirroring the initial model along the XY and YZ planes.

10. Conclusions

This paper addressed the difficulties of applying novel topology optimization methods in the industry, where the black-box system common to most commercial software limits the access to relevant information. Therefore, a new open methodology for solving non-self-adjoint topology optimization problems in ABAQUS® was presented.

The functioning of this method was validated through two procedures: first, by comparing the results of the sensitivity analysis with finite differences, which indicated a good correlation between both; then, by solving two benchmark problems with the following algorithms: Optimality Criteria, Method of Moving Asymptotes, Sequential Least-Squares Quadratic Programming, and Trust-constr algorithm. The results obtained, convergence, and influence of the problem statement chosen were in agreement with other state-of-the-art results. In the stress-minimization case study, having access to optimization algorithms unavailable in commercial software allowed the methodology herein presented a maximum stress reduction of 35% when compared to the solution obtained in ABAQUS®.

An implementation of this method is provided in Python, working in conjunction with ABAQUS®, in order to allow a direct application to new problems. The code implementation, as well as the ABAQUS® models necessary to recreate the work reported in this paper are available as Supplementary Material and can also be downloaded in digital format from the following Dataset [94] or repository: https://github.com/pnfernandes/Python-Code-for-Stress-Constrained-Topology-Optimization-in-ABAQUS.

Therefore, this article promotes the advance of the state-of-the-art by providing the tools required to solve stress-dependent geometrically complex problems, as well as a modular path that opens the possibility of implementing novel methods in an industrial environment.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/app132312916/s1 and Dataset [94] or repository: https://github.com/pnfernandes/Python-Code-for-Stress-Constrained-Topology-Optimization-in-ABAQUS. The Python code implementation, the ABAQUS® numerical models, and the complete raw data used in this work.

Author Contributions: Conceptualization, P.F. and A.F.; methodology, A.F.; software, P.F. and M.P.; validation, P.F. and A.F.; formal analysis, P.F., A.F. and P.G.; investigation, P.F.; resources, N.C.; data curation, P.F.; writing—original draft preparation, P.F.; writing—review and editing, A.F.; visualization, P.F.; supervision, A.F., R.P. and N.C.; project administration, R.P. and N.C.; funding acquisition, N.C. All authors have read and agreed to the published version of the manuscript.

Funding: P. Fernandes gratefully acknowledges the financial support from FCT—Fundação para a Ciência e a Tecnologia, I.P., under the scope of the Ph.D. Grant SFRH/BD/145425/2019. A. Ferrer gratefully acknowledges the partial support of Serra Húnter Research Program (Spain).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations, acronyms, and symbols are used in this manuscript:

2DQ4	A 2D element with four nodes
3D	Three-Dimensional
BESO	Bi-directional Evolutionary Structural Optimization
C3D8	A 3D Cube element with eight nodes
C3D10	A 3D Cube element with 10 nodes
CPE4	Plane–strain Element with four nodes
CPS4	Plane–stress element with four nodes
FEA	Finite-Element Analysis
MDPI	Multidisciplinary Digital Publishing Institute
MMA	Method of Moving Asymptotes
OC	Optimality Criterion

PEP8	Style Guide for Python Code
S4	Shell element with four nodes
SIMP	Solid Isotropic Material with Penalization
SLSQP	Sequential Least-Squares Programming
β	Stress penalization factor
$\bigtriangledown^s$	Symmetric gradient
η	Numerical damping coefficient
γ	Generic vector
$\mathbb{C}$	Element stiffness
ψ	Lagrange multiplier
ρ	Filtered design density
σ	Stress
σ_a^{VM}	Amplified von Mises stress
σ_a	Amplified stress
B_a	Strain–displacement matrix
B_e	Optimality condition parameter
E	Young's modulus
E^e	Elastic strain energy
E^p	Plastic strain energy
E_{rr}	Pondered error
e_{vol}	Material constraint evolution ratio
F	Force
f	Generic load function
g	Gradient
$H_0^1(\Omega)$	Space of functions with square integrable derivatives and homogeneous boundary values
J	Cost function
K	Stiffness
l	External load forces
$L^\infty(\Omega)$	Space of bounded functions
m	Mass
M	von Mises matrix
N	Linear shape function
n	Normal direction pointing outwards of the boundary $\partial\Omega$ of v
Ω	Domain of the topology optimization problem
P	SIMP penalty factor
Q_F	Final p-norm approximation factor
Q_i	Initial p-norm approximation factor
r_{max}	Maximum filter search radius
t	Applied boundary forces
u	Displacement
v	Poisson's coefficient
V	Volume
V^*	Maximum allowable volume
x	Design density variable

References

1. Bendsøe, M.P.; Kikuchi, N. Generating optimal topologies in structural design using a homogenization method. *Comput. Methods Appl. Mech. Eng.* **1988**, *71*, 197–224. [CrossRef]
2. Alderliesten, R.C. Designing for damage tolerance in aerospace: A hybrid material technology. *Mater. Des.* **2015**, *66*, 421–428. [CrossRef]
3. Meng, L.; Zhang, W.; Quan, D.; Shi, G.; Tang, L.; Hou, Y.; Breitkopf, P.; Zhu, J.; Gao, T. From Topology Optimization Design to Additive Manufacturing: Today's Success and Tomorrow's Roadmap. *Arch. Comput. Methods Eng.* **2020**, *27*, 805–830. [CrossRef]
4. Zhang, W.; Yang, J.; Xu, Y.; Gao, T. Topology optimization of thermoelastic structures: Mean compliance minimization or elastic strain energy minimization. *Struct. Multidiscip. Optim.* **2014**, *49*, 417–429. [CrossRef]
5. Wang, C.; Zhao, Z.; Zhou, M.; Sigmund, O.; Zhang, X.S. A comprehensive review of educational articles on structural and multidisciplinary optimization. *Struct. Multidiscip. Optim.* **2021**, *64*, 2827–2880. [CrossRef]
6. Sigmund, O. A 99 line topology optimization code written in matlab. *Struct. Multidiscip. Optim.* **2001**, *21*, 120–127. [CrossRef]

7. Andreassen, E.; Clausen, A.; Schevenels, M.; Lazarov, B.S.; Sigmund, O. Efficient topology optimization in MATLAB using 88 lines of code. *Struct. Multidiscip. Optim.* **2011**, *43*, 1–16. [CrossRef]
8. Liu, K.; Tovar, A. An efficient 3D topology optimization code written in Matlab. *Struct. Multidiscip. Optim.* **2014**, *50*, 1175–1196. [CrossRef]
9. Bendsøe, M.P. Optimal shape design as a material distribution problem. *Struct. Optim.* **1989**, *1*, 193–202. [CrossRef]
10. Zhou, M.; Rozvany, G.I. The COC algorithm, Part II: Topological, geometrical and generalized shape optimization. *Comput. Methods Appl. Mech. Eng.* **1991**, *89*, 309–336. [CrossRef]
11. Rozvany, G.I.N.; Sobieszczanski-Sobieski, J. New optimality criteria methods: Forcing uniqueness of the adjoint strains by corner-rounding at constraint intersections. *Struct. Optim.* **1992**, *4*, 244–246. [CrossRef]
12. Osher, S.; Sethian, J.A. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.* **1988**, *79*, 12–49. [CrossRef]
13. Wang, M.Y.; Wang, X.; Guo, D. A level set method for structural topology optimization. *Comput. Methods Appl. Mech. Eng.* **2003**, *192*, 227–246. [CrossRef]
14. Challis, V.J. A discrete level-set topology optimization code written in Matlab. *Struct. Multidiscip. Optim.* **2010**, *41*, 453–464. [CrossRef]
15. Young, V.; Querin, O.M.; Steven, G.P.; Xie, Y.M. 3D and multiple load case bi-directional evolutionary structural optimization (BESO). *Struct. Optim.* **1999**, *18*, 183–192. [CrossRef]
16. Huang, X.; Xie, Y.M. A further review of ESO type methods for topology optimization. *Struct. Multidiscip. Optim.* **2010**, *41*, 671–683. [CrossRef]
17. Zuo, Z.H.; Xie, Y.M. A simple and compact Python code for complex 3D topology optimization. *Adv. Eng. Softw.* **2015**, *85*, 1–11. [CrossRef]
18. Sigmund, O. On the usefulness of non-gradient approaches in topology optimization. *Struct. Multidiscip. Optim.* **2011**, *43*, 589–596. [CrossRef]
19. Zhu, B.; Zhang, X.; Zhang, H.; Liang, J.; Zang, H.; Li, H.; Wang, R. Design of compliant mechanisms using continuum topology optimization: A review. *Mech. Mach. Theory* **2020**, *143*, 103622. [CrossRef]
20. Sigmund, O.; Maute, K. Topology optimization approaches: A comparative review. *Struct. Multidiscip. Optim.* **2013**, *48*, 1031–1055. [CrossRef]
21. Harzheim, L.; Graf, G. A review of optimization of cast parts using topology optimization: II-Topology optimization with manufacturing constraints. *Struct. Multidiscip. Optim.* **2006**, *31*, 388–399. [CrossRef]
22. Suresh, K. A 199-line Matlab code for Pareto-optimal tracing in topology optimization. *Struct. Multidiscip. Optim.* **2010**, *42*, 665–679. [CrossRef]
23. Talischi, C.; Paulino, G.H.; Pereira, A.; Menezes, I.F.M. PolyTop: A Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Struct. Multidiscip. Optim.* **2012**, *45*, 329–357. [CrossRef]
24. Smith, H.; Norato, J.A. A MATLAB code for topology optimization using the geometry projection method. *Struct. Multidiscip. Optim.* **2020**, *62*, 1579–1594. [CrossRef]
25. Sanders, E.D.; Pereira, A.; Aguiló, M.A.; Paulino, G.H. PolyMat: An efficient Matlab code for multi-material topology optimization. *Struct. Multidiscip. Optim.* **2018**, *58*, 2727–2759. [CrossRef]
26. Otomori, M.; Yamada, T.; Izui, K.; Nishiwaki, S. Matlab code for a level set-based topology optimization method using a reaction diffusion equation. *Struct. Multidiscip. Optim.* **2015**, *51*, 1159–1172. [CrossRef]
27. Gao, J.; Luo, Z.; Xia, L.; Gao, L. Concurrent topology optimization of multiscale composite structures in Matlab. *Struct. Multidiscip. Optim.* **2019**, *60*, 2621–2651. [CrossRef]
28. Xia, L.; Breitkopf, P. Design of materials using topology optimization and energy-based homogenization approach in Matlab. *Struct. Multidiscip. Optim.* **2015**, *52*, 1229–1241. [CrossRef]
29. Aage, N.; Andreassen, E.; Lazarov, B.S. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Struct. Multidiscip. Optim.* **2015**, *51*, 565–572. [CrossRef]
30. Liang, Y.; Cheng, G. Further elaborations on topology optimization via sequential integer programming and Canonical relaxation algorithm and 128-line MATLAB code. *Struct. Multidiscip. Optim.* **2020**, *61*, 411–431. [CrossRef]
31. Picelli, R.; Sivapuram, R.; Xie, Y.M. A 101-line MATLAB code for topology optimization using binary variables and integer programming. *Struct. Multidiscip. Optim.* **2021**, *63*, 935–954. [CrossRef]
32. Ferrari, F.; Sigmund, O. A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D. *Struct. Multidiscip. Optim.* **2020**, *62*, 2211–2228. [CrossRef]
33. Du, Z.; Cui, T.; Liu, C.; Zhang, W.; Guo, Y.; Guo, X. An efficient and easy-to-extend Matlab code of the Moving Morphable Component (MMC) method for three-dimensional topology optimization. *Struct. Multidiscip. Optim.* **2022**, *65*, 158. [CrossRef]
34. Chen, Q.; Zhang, X.; Zhu, B. A 213-line topology optimization code for geometrically nonlinear structures. *Struct. Multidiscip. Optim.* **2019**, *59*, 1863–1879. [CrossRef]
35. Sotiropoulos, S.; Kazakis, G.; Lagaros, N.D. Conceptual design of structural systems based on topology optimization and prefabricated components. *Comput. Struct.* **2020**, *226*, 106136. [CrossRef]
36. Giraldo-Londoño, O.; Paulino, G.H. PolyStress: A Matlab implementation for local stress-constrained topology optimization using the augmented Lagrangian method. *Struct. Multidiscip. Optim.* **2021**, *63*, 2065–2097. [CrossRef]

37. Yang, D.; Liu, H.; Zhang, W.; Li, S. Stress-constrained topology optimization based on maximum stress measures. *Comput. Struct.* **2018**, *198*, 23–39. [CrossRef]
38. Bruggi, M.; Venini, P. A mixed FEM approach to stress-constrained topology optimization. *Int. J. Numer. Methods Eng.* **2008**, *73*, 1693–1714. [CrossRef]
39. Lee, E.; James, K.A.; Martins, J.R.R.A. Stress-constrained topology optimization with design-dependent loading. *Struct. Multidiscip. Optim.* **2012**, *46*, 647–661. [CrossRef]
40. Cai, S.; Zhang, W. Stress-constrained topology optimization with free-form design domains. *Comput. Methods Appl. Mech. Eng.* **2015**, *289*, 267–290. [CrossRef]
41. Salazar de Troya, M.A.; Tortorelli, D.A. Adaptive mesh refinement in stress-constrained topology optimization. *Struct. Multidiscip. Optim.* **2018**, *58*, 2369–2386. [CrossRef]
42. Suresh, K.; Takalloozadeh, M. Stress-constrained topology optimization: A topological level-set approach. *Struct. Multidiscip. Optim.* **2013**, *48*, 295–309. [CrossRef]
43. Chu, S.; Gao, L.; Xiao, M.; Luo, Z.; Li, H.; Gui, X. A new method based on adaptive volume constraint and stress penalty for stress-constrained topology optimization. *Struct. Multidiscip. Optim.* **2018**, *57*, 1163–1185. [CrossRef]
44. Oh, M.K.; Lee, D.S.; Yoo, J. Stress-constrained topology optimization simultaneously considering the uncertainty of load positions. *Int. J. Numer. Methods Eng.* **2022**, *123*, 339–365. [CrossRef]
45. Luo, Y.; Wang, M.Y.; Kang, Z. An enhanced aggregation method for topology optimization with local stress constraints. *Comput. Methods Appl. Mech. Eng.* **2013**, *254*, 31–41. [CrossRef]
46. Biyikli, E.; To, A.C. Proportional topology optimization: A new non-sensitivity method for solving stress-constrained and minimum compliance problems and its implementation in MATLAB. *PLoS ONE* **2015**, *10*, e0145041. [CrossRef] [PubMed]
47. Amir, O. Efficient stress-constrained topology optimization using inexact design sensitivities. *Int. J. Numer. Methods Eng.* **2021**, *122*, 3241–3272. [CrossRef]
48. Paris, J.; Navarrina, F.; Colominas, I.; Casteleiro, M. Stress constraints sensitivity analysis in structural topology optimization. *Comput. Methods Appl. Mech. Eng.* **2010**, *199*, 2110–2122. [CrossRef]
49. Holmberg, E.; Torstenfelt, B.; Klarbring, A. Global and clustered approaches for stress-constrained topology optimization and deactivation of design variables. In Proceedings of the 10th World Congress on Structural and Multidisciplinary Optimization, Orlando, FL, USA, 19–24 May 2013; pp. 1–10.
50. Norato, J.A.; Smith, H.A.; Deaton, J.D.; Kolonay, R.M. A maximum-rectifier-function approach to stress-constrained topology optimization. *Struct. Multidiscip. Optim.* **2022**, *65*, 286. [CrossRef]
51. Burger, M.; Stainko, R. Phase-field relaxation of topology optimization with local stress constraints. *SIAM J. Control Optim.* **2006**, *45*, 1447–1466. [CrossRef]
52. De Leon, D.M.; Alexandersen, J.; O. Fonseca, J.S.; Sigmund, O. Stress-constrained topology optimization for compliant mechanism design. *Struct. Multidiscip. Optim.* **2015**, *52*, 929–943. [CrossRef]
53. Holmberg, E.; Torstenfelt, B.; Klarbring, A. Stress-constrained topology optimization. *Struct. Multidiscip. Optim.* **2013**, *48*, 33–47. [CrossRef]
54. Pastore, T.; Mercuri, V.; Menna, C.; Asprone, D.; Festa, P.; Auricchio, F. Topology optimization of stress-constrained structural elements using risk-factor approach. *Comput. Struct.* **2019**, *224*, 106104. [CrossRef]
55. Paris, J.; Colominas, I.; Navarrina, F.; Casteleiro, M. Parallel computing in topology optimization of structures with stress constraints. *Comput. Struct.* **2013**, *125*, 62–73. [CrossRef]
56. Deng, S.; Suresh, K. Multi-constrained topology optimization via the topological sensitivity. *Struct. Multidiscip. Optim.* **2015**, *51*, 987–1001. [CrossRef]
57. Senhora, F.V.; Giraldo-Londoño, O.; Menezes, I.F.M.; Paulino, G.H. Topology optimization with local stress constraints: A stress aggregation-free approach. *Struct. Multidiscip. Optim.* **2020**, *62*, 1639–1668. [CrossRef]
58. Saadlaoui, Y.; Milan, J.L.; Rossi, J.M.; Chabrand, P. Topology optimization and additive manufacturing: Comparison of conception methods using industrial codes. *J. Manuf. Syst.* **2017**, *43*, 178–186. [CrossRef]
59. Holmberg, E.; Torstenfelt, B.; Klarbring, A. Fatigue constrained topology optimization. *Struct. Multidiscip. Optim.* **2014**, *50*, 207–219. [CrossRef]
60. Amir, O.; Lazarov, B.S. Achieving stress-constrained topological design via length scale control. *Struct. Multidiscip. Optim.* **2018**, *58*, 2053–2071. [CrossRef]
61. Granlund, G.; Wallin, M.; Tortorelli, D.; Watts, S. Stress-constrained topology optimization of structures subjected to nonproportional loading. *Int. J. Numer. Methods Eng.* **2023**, *124*, 2818–2836. [CrossRef]
62. Amstutz, S.; Novotny, A.A.; de Souza Neto, E.A. Topological derivative-based topology optimization of structures subject to Drucker–Prager stress constraints. *Comput. Methods Appl. Mech. Eng.* **2012**, *233*, 123–136. [CrossRef]
63. Cheng, L.; Bai, J.; To, A.C. Functionally graded lattice structure topology optimization for the design of additive manufactured components with stress constraints. *Comput. Methods Appl. Mech. Eng.* **2019**, *344*, 334–359. [CrossRef]
64. Deng, H.; Vulimiri, P.S.; To, A.C. An efficient 146-line 3D sensitivity analysis code of stress-based topology optimization written in MATLAB. *Optim. Eng.* **2021**, *23*, 1733–1757. [CrossRef]
65. Mirzendehdel, A.M.; Suresh, K. Support structure constrained topology optimization for additive manufacturing. *Comput.-Aided Des.* **2016**, *81*, 1–13. [CrossRef]

66. Collet, M.; Bruggi, M.; Duysinx, P. Topology optimization for minimum weight with compliance and simplified nominal stress constraints for fatigue resistance. *Struct. Multidiscip. Optim.* **2017**, *55*, 839–855. [CrossRef]
67. Bruggi, M.; Duysinx, P. Topology optimization for minimum weight with compliance and stress constraints. *Struct. Multidiscip. Optim.* **2012**, *46*, 369–384. [CrossRef]
68. Giraldo-Londoño, O.; Aguiló, M.A.; Paulino, G.H. Local stress constraints in topology optimization of structures subjected to arbitrary dynamic loads: A stress aggregation-free approach. *Struct. Multidiscip. Optim.* **2021**, *64*, 3287–3309. [CrossRef]
69. París, J.; Navarrina, F.; Colominas, I.; Casteleiro, M. Topology optimization of continuum structures with local and global stress constraints. *Struct. Multidiscip. Optim.* **2009**, *39*, 419–437. [CrossRef]
70. Long, K.; Wang, X.; Liu, H. Stress-constrained topology optimization of continuum structures subjected to harmonic force excitation using sequential quadratic programming. *Struct. Multidiscip. Optim.* **2019**, *59*, 1747–1759. [CrossRef]
71. Duysinx, P. Topology optimization with different stress limit in tension and compression. In Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization (WCSMO3), Buffalo, NY, USA, 17–21 May 1999.
72. da Silva, A.L.F.; Salas, R.A.; Nelli Silva, E.C.; Reddy, J.N. Topology optimization of fibers orientation in hyperelastic composite material. *Compos. Struct.* **2020**, *231*, 111488. [CrossRef]
73. Miyajima, K.; Noguchi, Y.; Yamada, T. Optimal design of compliant displacement magnification mechanisms using stress-constrained topology optimization based on effective energy. *Finite Elem. Anal. Des.* **2023**, *216*, 103892. [CrossRef]
74. Emmendoerfer Jr, H.; Fancello, E.A.; Silva, E.C.N. Stress-constrained level set topology optimization for compliant mechanisms. *Comput. Methods Appl. Mech. Eng.* **2020**, *362*, 112777. [CrossRef]
75. Bendsoe, M.P.; Sigmund, O. *Topology Optimization: Theory, Methods, and Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2003.
76. Bakhtiary, N.; Allinger, P.; Friedrich, M.; Mulfinger, F.; Sauter, J.; Müller, O.; Puchinger, M. A new approach for sizing, shape and topology optimization. *SAE Trans.* **1996**, *105*, 745–761.
77. Mlejnek, H.P. Some aspects of the genesis of structures. *Struct. Optim.* **1992**, *5*, 64–69. [CrossRef]
78. Stolpe, M.; Svanberg, K. An alternative interpolation scheme for minimum compliance topology optimization. *Struct. Multidiscip. Optim.* **2001**, *22*, 116–124. [CrossRef]
79. Pedersen, C.B.W.; Allinger, P. Industrial implementation and applications of topology optimization and future needs. In *IUTAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 229–238.
80. Bendsøe, M.P.; Sigmund, O. Material interpolation schemes in topology optimization. *Arch. Appl. Mech.* **1999**, *69*, 635–654. [CrossRef]
81. Clausen, P.M.; Pedersen, C.B.W. Non-parametric large scale structural optimization for industrial applications. In *III European Conference on Computational Mechanics: Solids, Structures and Coupled Problems in Engineering: Book of Abstracts*; Springer: Berlin/Heidelberg, Germany, 2006; p. 482.
82. Svanberg, K. The method of moving asymptotes—a new method for structural optimization. *Int. J. Numer. Methods Eng.* **1987**, *24*, 359–373. [CrossRef]
83. Kabus, S.; Pedersen, C.B.W. Optimal bearing housing designing using topology optimization. *J. Tribol.* **2012**, *134*, 021102. [CrossRef]
84. Søndergaard, M.B.; Pedersen, C.B.W. Applied topology optimization of vibro-acoustic hearing instrument models. *J. Sound Vib.* **2014**, *333*, 683–692. [CrossRef]
85. Hansen, L.V. Topology optimization of free vibrations of fiber laser packages. *Struct. Multidiscip. Optim.* **2005**, *29*, 341–348. [CrossRef]
86. Olhoff, N.; Du, J. Topology optimization of vibrating bi-material structures with respect to sound radiation. In *IUTAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 43–52.
87. JOG, C.S. Topology design of structures subjected to periodic loading. *J. Sound Vib.* **2002**, *253*, 687–709. [CrossRef]
88. Sigmund, O.; Søndergaard Jensen, J. Systematic design of phononic band–gap materials and structures by topology optimization. *Philos. Trans. R. Soc. London. Ser. A Math. Phys. Eng. Sci.* **2003**, *361*, 1001–1019. [CrossRef]
89. Tcherniak, D. Topology optimization of resonating structures using SIMP method. *Int. J. Numer. Methods Eng.* **2002**, *54*, 1605–1622. [CrossRef]
90. Guo, L.; Wang, X.; Meng, Z.; Yu, B. Reliability-based topology optimization of continuum structure under buckling and compliance constraints. *Int. J. Numer. Methods Eng.* **2022**, *123*, 4032–4053. [CrossRef]
91. Meng, Z.; Pang, Y.; Pu, Y.; Wang, X. New hybrid reliability-based topology optimization method combining fuzzy and probabilistic models for handling epistemic and aleatory uncertainties. *Comput. Methods Appl. Mech. Eng.* **2020**, *363*, 112886. [CrossRef]
92. Meng, Z.; Keshtegar, B. Adaptive conjugate single-loop method for efficient reliability-based design and topology optimization. *Comput. Methods Appl. Mech. Eng.* **2019**, *344*, 95–119. [CrossRef]
93. Meng, Z.; Wu, Y.; Wang, X.; Ren, S.; Yu, B. Robust topology optimization methodology for continuum structures under probabilistic and fuzzy uncertainties. *Int. J. Numer. Methods Eng.* **2021**, *122*, 2095–2111. [CrossRef]
94. Fernandes, P.; Ferrer, A.; Teixeira, P.; Parente, M.; Pinto, R.; Correia, N. Python Code for Stress Constrained Topology Optimization in ABAQUS. 2023. [CrossRef]

95. Bruggi, M. On an alternative approach to stress constraints relaxation in topology optimization. *Struct. Multidiscip. Optim.* **2008**, *36*, 125–141. [CrossRef]
96. Le, C.; Norato, J.; Bruns, T.; Ha, C.; Tortorelli, D. Stress-based topology optimization for continua. *Struct. Multidiscip. Optim.* **2010**, *41*, 605–620. [CrossRef]
97. Yang, L.; Lavrinenko, A.V.; Hvam, J.M.; Sigmund, O. Design of one-dimensional optical pulse-shaping filters by time-domain topology optimization. *Appl. Phys. Lett.* **2009**, *95*, 261101. [CrossRef]
98. Huang, X.; Xie, Y.M. Bi-directional evolutionary topology optimization of continuum structures with one or multiple materials. *Comput. Mech.* **2009**, *43*, 393–401. [CrossRef]
99. Bendsøe, M.P. *Optimization of Structural Topology, Shape, and Material*; Springer: Berlin/Heidelberg, Germany, 1995. [CrossRef]
100. Ferraro, S. *Topology Optimization and Failure Analysis of Deployable Thin Shells with Cutouts*; California Institute of Technology: Pasadena, CA, USA, 2020.
101. Sigmund, O.; Petersson, J. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Struct. Optim.* **1998**, *16*, 68–75. [CrossRef]
102. Huang, X.; Xie, Y.M. Convergent and mesh-independent solutions for the bi-directional evolutionary structural optimization method. *Finite Elem. Anal. Des.* **2007**, *43*, 1039–1049. [CrossRef]
103. Wang, F.; Lazarov, B.S.; Sigmund, O. On projection methods, convergence and robust formulations in topology optimization. *Struct. Multidiscip. Optim.* **2011**, *43*, 767–784. [CrossRef]
104. Svanberg, K. Algorithm for Optimum Structural Design Using Duality. In *Mathematical Programming Study*; Number 20; Springer: Berlin/Heidelberg, Germany, 1982; pp. 161–177. [CrossRef]
105. Fleury, C. Structural weight optimization by dual methods of convex programming. *Int. J. Numer. Methods Eng.* **1979**, *14*, 1761–1783. [CrossRef]
106. Deetman, A. GCMMA-MMA-Python Home Page. Available online: https://github.com/arjendeetman/GCMMA-MMA-Python (accessed on 26 October 2023).
107. Svanberg, K. MMA and GCMMA Matlab Code Home Page. Available online: https://www.smoptit.se/ (accessed on 1 December 2023).
108. Virtanen, P.; Gommers, R.; Oliphant, T.; Haberland, M.; Reddy, T.; Cournapeau, D. SciPy 1.2.1: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef] [PubMed]
109. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006; pp. 1–664. [CrossRef]
110. Byrd, R.H.; Hribar, M.E.; Nocedal, J. An interior point algorithm for large-scale nonlinear programming. *SIAM J. Optim.* **1999**, *9*, 877–900. [CrossRef]
111. development team van Rossum, P. *The Python Language Reference Release 3.6.0*; Network Theory Ltd.: Godalming, UK, 2017.
112. Buhl, T.; Pedersen, C.B.; Sigmund, O. Stiffness design of geometrically nonlinear structures using topology optimization. *Struct. Multidiscip. Optim.* **2000**, *19*, 93–104. [CrossRef]
113. Duysinx, P.; Sigmund, O. New developments in handling stress constraints in optimal material distribution. In Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, USA, 2–4 September 1998; pp. 1501–1509. [CrossRef]
114. Bendsøe, M.P.; Díaz, A.R. A method for treating damage related criteria in optimal topology design of continuum structures. *Struct. Optim.* **1998**, *16*, 108–115. [CrossRef]