

# Topology Optimization for Fiber in Composite Materials

Andre Luis Ferreira da Silva and Ruben Andres Salas and Emilio Carlos Nelli Silva

## Synonyms

Topology optimization; Fiber angle optimization; Structural optimization.

## Definitions

Composite materials have two or more components designed to perform better than each material acting individually. A particular composite material is reinforced with fiber and is primarily used in numerous engineering applications.

Plane stress is a simplified way of analyzing the stress in a thin object, like a plate or sheet, where the thickness is much smaller than the other two dimensions. It assumes that the stress acting on the object is all in one plane, and there is no stress acting perpendicular to that plane.

The finite element method is a numerical technique for solving differential equations representing physical problems.

Topology optimization is a method used to distribute a design variable in an optimized way, generally in a fixed domain, by minimizing an objective function considering certain constraints. There is no unit interpretation for the design vari-

---

Andre Luis Ferreira da Silva

Polytechnique School of the University of Sao Paulo, Avenida Professor Mello de Moraes, 2231, 05508-030 Sao Paulo, SP, Brazil, e-mail: andre\_fersi@usp.br

Ruben Andres Salas

Polytechnique School of the University of Sao Paulo, Avenida Professor Mello de Moraes, 2231, 05508-030 Sao Paulo, SP, Brazil e-mail: ruansava@yahoo.com.br

Emilio Carlos Nelli Silva

Polytechnique School of the University of Sao Paulo, Avenida Professor Mello de Moraes, 2231, 05508-030 Sao Paulo, SP, Brazil e-mail: ecnsilva@usp.br

able; it depends on the physics that needs to be simulated. In classical structural topology optimization, the design variable represents the presence or lack of material. In fluid topology optimization, the design variable represents the path of fluid and the presence of solid structure in the domain. In the context of this chapter, the design variable is the fiber angle. However, a notable feature in TO is the possibility of using multiple design variables in the same problem (see Silva et al (2023)).

Compliance, in the context of structural mechanics, refers to the ability of the material to deform under a load application. The compliance can also be interpreted as the inverse of the stiffness and calculated using the deformation energy concept, a quantity which allows to design structures.

## Introduction

The use of fiber to reinforce materials is not new. Egyptians used bamboo shoots to reinforce mud walls in 1500 B.C. Kaw (2006). With the development of new technologies, materials reinforced with fibers find extensive applications in various engineering domains, such as automotive, aerospace, sports, and civil engineering. These materials offer notable advantages, particularly in terms of the strength-to-mass ratio. Comprising a reinforcing element (the fiber) and a matrix that positions the fiber within the structure, the orientation of the fiber plays a pivotal role in determining the structure behavior. Any alteration in fiber orientation corresponds to a change in structural behavior. One general example is given by Fig. 1, which shows two unidirectional fiber-reinforced laminas with fibers oriented horizontally and vertically, subject to a surface force  $\mathbf{t}$ . As previously cited, the main function of the matrix is to keep the fiber in a determined orientation; that is, the strength of the matrix is much smaller than the strength of the fiber. So, for the boundary conditions shown in Fig. 1, the lamina with fibers oriented horizontally will have a smaller strain. Consequently, optimization algorithms become feasible to determine the ideal fiber orientation based on an objective function, such as minimizing structure compliance. This chapter provides a detailed explanation of the formulation for the topology optimization process aimed at minimizing the compliance of a fiber-reinforced structure. The implementation of a topology optimization process is performed using the FEniCS open-source computing platform, accompanied by examples of results.

## Theoretical formulation

The solution of a Topology Optimization problem consists of important key points. First of all, it is necessary to define what will be optimized (design requirements), that is, what the objective function is and what the physics that explains the phenomenon is. The objective function must be continuous and differentiable. After

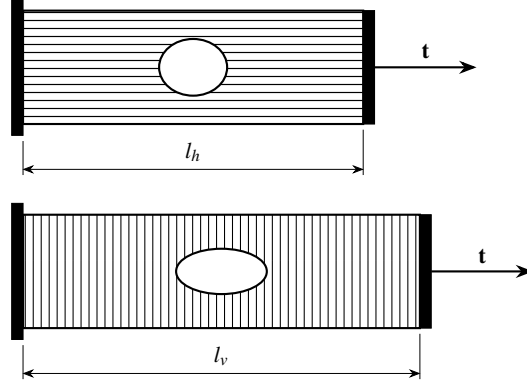


Fig. 1: Comparison between a unidirectional fiber-reinforced lamina with fibers oriented horizontally and vertically.

that, it is necessary to calculate the sensitivities of the problem, which represents how sensitive the problem is if it is subject to a perturbation in the design variable. Finally, applying an optimization algorithm will define the optimized field of design variables iteratively until a stop criterion is reached. In this section, each one of these important steps is explained.

### *Linear elasticity formulation*

The problem solved in this Chapter is the minimization of compliance of a structure. As explained in Section Definitions, compliance can also be interpreted as the inverse of stiffness, which means that topology optimization aims to maximize the structure stiffness. For this, it is necessary to determine the equations that better explain the structure behavior based on some assumed hypothesis. Here, the hypothesis of small displacements, small deformations, and small rotations is assumed. It is also considered that the load applied will not cause any plasticity or damage behavior in the structure and that the load is quasi-static; the loads are applied so slowly that the acceleration effect can be neglected. Another hypothesis adopted is the plane stress state, explained in the Sec. Definition.

Fig. 2 represents a generic design domain where the problem is defined. The domain is represented by  $\Omega$ , the part of the figure filled in gray color. The boundaries of the domain are represented by  $\partial\Omega$ . The letter  $\mathbf{b}$  represents a vector of body forces. In the structural context, the more common type of body force is caused by gravity acceleration. The unit of measure of the body forces is force divided by the volume. For this specific case, it is considered the presence of fibers in the structure represented by the red lines. The vector of surface forces is represented by the letter  $\mathbf{t}$ . Surface forces are external forces applied in the boundary  $\Gamma_t$ , and its unit measure

is force divided by area. The coordinate system can be represented by a vector  $\mathbf{x}$ , which has components  $x_1, x_2$ , and  $x_3$  or  $x, y$ , and  $z$ .

$$\mathbf{x} = x_1 \mathbf{e}_1 + x_2 \mathbf{e}_2 + x_3 \mathbf{e}_3 = x_i \mathbf{e}_i \quad (1)$$

where  $\mathbf{e}_i$  represents an unit vector in the direction  $i$

The behavior of the structure can be represented by the displacements in each point of the domain, which are caused by the forces. The displacements are represented by

$$\mathbf{u} = u_1 \mathbf{e}_1 + u_2 \mathbf{e}_2 + u_3 \mathbf{e}_3 = u_i \mathbf{e}_i \quad (2)$$

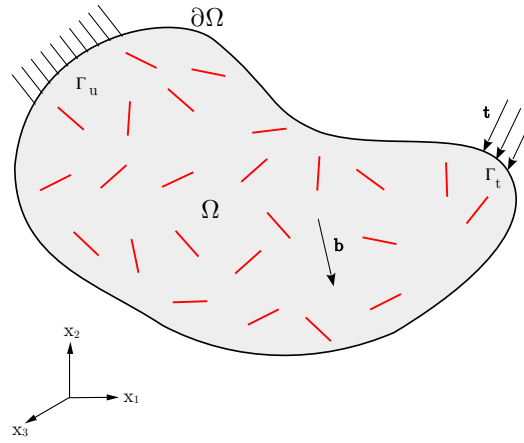


Fig. 2: Generic design domain.

Another important measure for the linear elastic simulation is the strain, which represents a relative deformation in the structure. From the assumed hypothesis, it is possible to write the strain tensor as Lai (2010)

$$\epsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad \text{in } \Omega \quad (3)$$

where  $\frac{\partial u_i}{\partial x_j}$  represents the partial derivative of the  $i^{th}$  component of the displacement vector in relation to the coordinate  $x_j$ , in other words, the variation of  $u_i$  in the direction  $x_j$ .

The next step is defining the *equilibrium equations*, also known as *balance of momentum*. Recalling the hypotheses that the load is quasi-static, that is, the acceleration effects can be neglected, it is possible to write the *balance of linear momentum* as Lai (2010)

$$\frac{\partial \sigma_{ij}}{\partial x_j} + b_i = 0 \quad \text{in } \Omega \quad (4)$$

where  $\sigma_{ij}$  are components of the Cauchy stress tensor. The *balance of angular momentum* yields the symmetry of the Cauchy stress tensor, which is represented by Lai (2010)

$$\sigma_{ij} = \sigma_{ji} \quad \text{in } \Omega \quad (5)$$

Now, defining the *boundary conditions* is necessary. The first type of boundary condition is the displacement or *Dirichlet boundary condition*, which represents a known value of the displacements  $u$  for all points in the boundary delimited by  $\Gamma_u$ . The *Dirichlet boundary conditions* are defined as Lai (2010)

$$u_i = u_i^0 \quad \text{on } \Gamma_u \quad (6)$$

The second boundary condition is the stress or *Neumann boundary conditions* represent by Lai (2010)

$$\sigma_{ij} n_i = t_j \quad \text{on } \Gamma_t \quad (7)$$

where  $n_j$  represents the normal vector to the boundary  $\Gamma_t$ .

The previous equations can be applied to any material that respects the established hypotheses. The particular behavior of the material is defined by the constitutive equation, which relates the stress to a strain. For the linear elastic material, the constitutive equation is represented by Lai (2010)

$$\sigma_{ij} = C_{ijkl} \varepsilon_{kl} \quad \text{in } \Omega \quad (8)$$

where  $C_{ijkl}$  represents the *elastic moduli* or the *constitutive tensor*. Taking advantage of the symmetry of the Cauchy stress tensor, it is possible to write the constitutive equation in its matrix form using the Voigt notation. So, the Cauchy stress tensor and the strain tensor, which are second-order tensors, can be represented as vectors. For 3D problems, both vectors have 6 components. However, in this Chapter, the plane stress state is been considered. So, just 3 components are necessary to represent each vector.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ 2\varepsilon_6 \end{bmatrix} \quad (9)$$

Here,  $\mathbf{Q}$  represents the constitutive tensor for the plane stress state. It is necessary to use a different symbol ( $\mathbf{Q}$  in place of  $\mathbf{C}$ ) because the components of the constitutive tensor are different for the 3D and plane stress cases. It is essential to highlight that the tensor  $\mathbf{Q}$  written in this way holds for an orthotropic material, that is, for a material with distinct behavior in each orthotropic direction. This is important to differentiate the behavior of the structure in the fiber direction and the direction perpendicular to the fiber.

The components of the tensor  $\mathbf{Q}$  can be calculated from the components of the tensor  $\mathbf{S}$ , known as the compliance tensor, which maps the strain from a given stress Kaw (2006)

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ 2\varepsilon_6 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & 0 \\ S_{12} & S_{22} & 0 \\ 0 & 0 & S_{66} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_6 \end{bmatrix} \quad (10)$$

The components of the compliance tensor  $\mathbf{S}$  are functions of the material properties as follows:

$$S_{11} = \frac{1}{E_1} \quad (11a)$$

$$S_{12} = \frac{-\nu_{12}}{E_1} \quad (11b)$$

$$S_{22} = \frac{1}{E_2} \quad (11c)$$

$$S_{66} = \frac{1}{G_{12}} \quad (11d)$$

where  $E_1$  and  $E_2$  represent Young's modulus in the fiber direction and perpendicular to the fiber direction, respectively,  $\nu_{12}$  represents the Poisson ratio considering an extension in direction 1, and a contraction in direction 2, an  $G_{12}$  is the shear modulus in plane 1-2.

Finally, the components of the constitutive tensor  $\mathbf{Q}$  can be calculated from the components of the compliance tensor  $\mathbf{S}$  as follows:

$$Q_{11} = \frac{S_{22}}{S_{11}S_{22} - S_{12}^2} \quad (12a)$$

$$Q_{12} = -\frac{S_{12}}{S_{11}S_{22} - S_{12}^2} \quad (12b)$$

$$Q_{22} = \frac{S_{11}}{S_{11}S_{22} - S_{12}^2} \quad (12c)$$

$$Q_{66} = \frac{1}{S_{66}} \quad (12d)$$

Eqs. from (9) to (12) are represented in local coordinates, that is, coordinate 1 represents the direction of the fiber. By observing Fig. 3 it is possible to note that to solve the problem, it is necessary to consider the angle between the fiber and the global coordinate  $x-y$

So, it is necessary to perform a mapping from the local and to the global coordinates. The mapping is performed by the transformation tensor  $\mathbf{T}$ , which is represented by

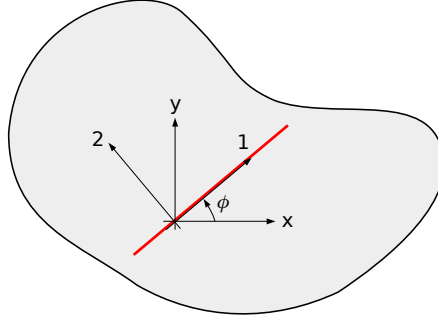


Fig. 3: Local and global coordinates

$$\mathbf{T} = \begin{bmatrix} \cos^2(\phi) & \sin^2(\phi) & 2\sin(\phi)\cos(\phi) \\ \sin^2(\phi) & \cos^2(\phi) & -2\sin(\phi)\cos(\phi) \\ -\sin(\phi)\cos(\phi) & \sin(\phi)\cos(\phi) & \cos^2(\phi) - \sin^2(\phi) \end{bmatrix} \quad (13)$$

The mapping for the Cauchy stress tensor from the local to the global coordinates is performed according to

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \mathbf{T}^{-1} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_6 \end{bmatrix} \quad (14)$$

Substituting the Eq. (9) into Eq. (14):

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \mathbf{T}^{-1} \mathbf{Q} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ 2\varepsilon_6 \end{bmatrix} \quad (15)$$

The following expression is used to map the strain from the local to the global coordinates:

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_6 \end{bmatrix} = \mathbf{R} \mathbf{T} \mathbf{R}^{-1} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{bmatrix} \quad (16)$$

where the Reuter matrix  $\mathbf{R}$  is calculated as follows:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad (17)$$

Substituting Eq. (16) into Eq. (15)

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \mathbf{T}^{-1} \mathbf{Q} \mathbf{R} \mathbf{T} \mathbf{R}^{-1} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{bmatrix} \quad (18)$$

In conclusion, the constitutive tensor  $\mathbf{C}(\phi)$  for orthotropic materials under plane stress can be written as

$$\mathbf{C}(\phi) = (\mathbf{T}(\phi))^{-1} \mathbf{Q} \mathbf{R} \mathbf{T}(\phi) \mathbf{R}^{-1} \quad (19)$$

### ***Weak form formulation***

There are several ways to solve the problem established by Eqs. from (3) to (12). However, because generally the domains  $\Omega$  used to be complex, most of the feasible approaches consists in numerical solutions. The Finite Element Method is an excellent choice for solving the differential equations that represent the behavior of a structure. To use the Finite Element Method, it is necessary to write the boundary value problem represented by by Eqs. from (3) to (12) in its *Weak formulation*. Reminding the boundary value problem with the differential equations in its strong formulation:

$$\begin{aligned} -\frac{\partial \sigma_{ij}}{\partial x_j} &= b_i & \forall x \in \Omega \\ \sigma_{ij} &= C_{ijkl} \varepsilon_{kl} & \forall x \in \Omega \\ \varepsilon_{ij} &= \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) & \forall x \in \Omega \\ u_i &= u_i^0 & \forall x \in \Gamma_u \\ \sigma_{ij} n_i &= t_j & \forall x \in \Gamma_t \\ \sigma_{ij} n_i &= 0 & \forall x \in \partial\Omega / \Gamma_t \end{aligned} \quad (20)$$

Let  $V$  be the space of admissible displacements for formulating the weak form, where  $\mathbf{u} \in V$ . Now, introduce an arbitrary test function, denoted as  $\mathbf{v}$  (also referred to as virtual displacements in structural contexts), which is also defined in  $V$  ( $\mathbf{v} \in V$ ) and satisfies to the Dirichlet boundary conditions. Multiplying both sides of the linear momentum balance by  $\mathbf{v}$  maintains the validity of the equality.

$$-\frac{\partial \sigma_{ij}}{\partial x_j} v_i = b_i v_i \quad (21)$$

The equation is valid for all  $x \in \Omega$ . So, it is possible to integrate Eq. (21) in  $\Omega$  as follows:

$$\int_{\Omega} -\frac{\partial \sigma_{ij}}{\partial x_j} v_i dx = \int_{\Omega} b_i v_i dx \quad (22)$$



From the product rule of the differentiation, it is possible to write th

$$\frac{\partial(\sigma_{ij} v_i)}{\partial x_j} = \frac{\partial \sigma_{ij}}{\partial x_j} v_i + \sigma_{ij} \frac{\partial v_i}{\partial x_j} \quad (23)$$

Rearranging the terms in Eq. (23) it is possible to determine the term  $-\frac{\partial \sigma_{ij}}{\partial x_j}$  of the Eq. (22) as

$$-\frac{\partial \sigma_{ij}}{\partial x_j} = -\frac{\partial(\sigma_{ij} v_i)}{\partial x_j} + \sigma_{ij} \frac{\partial v_i}{\partial x_j} \quad (24)$$

Substituting Eq. (24) in Eq. (22):

$$\int_{\Omega} \left( -\frac{\partial(\sigma_{ij} v_i)}{\partial x_j} + \sigma_{ij} \frac{\partial v_i}{\partial x_j} \right) dx = \int_{\Omega} b_i v_i dx \quad (25)$$

By using the divergent theorem in the first term of the left-hand side integral in Eq. (25), it is possible to rewrite the equation as

$$\int_{\partial \Omega} -\sigma_{ij} n_i v_j dS + \int_{\Omega} \sigma_{ij} \frac{\partial v_i}{\partial x_j} dx = \int_{\Omega} b_i v_i dx \quad (26)$$

According to the Neumann boundary conditions, the product of the Cauchy stress tensor with the unit normal vector is equal to  $\mathbf{t}$  on  $\Gamma_t$  and equal to  $\mathbf{0}$  in the rest of the boundary (see the last two conditions in Eq. (20)). By considering these boundary conditions in Eq. (26), it is possible to write:

$$\begin{aligned} \int_{\Gamma_t} -\sigma_{ij} n_i v_j dS + \int_{\Omega} \sigma_{ij} \frac{\partial v_i}{\partial x_j} dx &= \int_{\Omega} b_i v_i dx \implies \\ \int_{\Gamma_t} -f_i v_i dS + \int_{\Omega} \sigma_{ij} \frac{\partial v_i}{\partial x_j} dx &= \int_{\Omega} b_i v_i dx \implies \\ \int_{\Omega} \sigma_{ij} \frac{\partial v_i}{\partial x_j} dx &= \int_{\Omega} b_i v_i dx + \int_{\Gamma_t} f_i v_i dS \end{aligned} \quad (27)$$

According to the balance of angular momentum (Eq. (5)), the Cauchy stress tensor is symmetric, that is,  $\sigma_{ij} = \sigma_{ji}$ . Taking advantage of this property, it is possible to write:

$$\begin{aligned} \sigma_{ij} \frac{\partial v_i}{\partial x_j} &= \frac{1}{2} \sigma_{ij} \frac{\partial v_i}{\partial x_j} + \frac{1}{2} \sigma_{ij} \frac{\partial v_i}{\partial x_j} \implies \\ \sigma_{ij} \frac{\partial v_i}{\partial x_j} &= \frac{1}{2} \sigma_{ij} \frac{\partial v_i}{\partial x_j} + \frac{1}{2} \sigma_{ji} \frac{\partial v_j}{\partial x_i} \implies \\ \sigma_{ij} \frac{\partial v_i}{\partial x_j} &= \sigma_{ij} \frac{1}{2} \left( \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) \end{aligned} \quad (28)$$

Analyzing the right-hand side of Eq. (28), it is possible to notice that the term  $\frac{1}{2} \left( \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right)$  it is equivalent to the expression for the strain shown in Eq. (3). Con-

sidering this, it is possible to write the weak form formulation for the field equations of solid mechanics, considering linear elastic behavior as

$$\begin{aligned} \int_{\Omega} \sigma_{ij} \varepsilon_{ij}(\mathbf{v}) dx &= \int_{\Omega} b_i v_i dx + \int_{\Gamma_t} f_i v_i dS \implies \\ \int_{\Omega} C_{ijkl}(\phi) \varepsilon_{ij}(\mathbf{u}) \varepsilon_{kl}(\mathbf{v}) dx &= \int_{\Omega} b_i v_i dx + \int_{\Gamma_t} f_i v_i dS \end{aligned} \quad (29)$$

In the context of topology optimization, Eq. (29) is known as *forward problem* or *state equation*. The state equation is solved for the displacements  $\mathbf{u}$ , named *state variable*.

A widespread representation of the state equation is the *Energy bilinear form*  $a(\mathbf{u}, \mathbf{v})$  and *Load linear form*  $l(\mathbf{v})$  Bendsøe and Sigmund (2004). So, Eq. (29) can be rewrite simply as:

$$a(\phi, \mathbf{u}, \mathbf{v}) = l(\mathbf{v}) \quad (30)$$

As observed in Eq. (30) for the case of the composite materials,  $a$  is also a function of the fiber angle  $\phi$ .

### Optimization problem

An optimization problem consists of an objective function that is maximized or minimized and constraints that need to be satisfied. Eq. (31) shows a generic optimization problem that consists of minimizing the objective function  $J$  by changing the values of the design variable  $\phi$  subject to an equality constraint  $F$  and an inequality constraint  $G$ .

$$\begin{aligned} \min_{\phi} \quad & J(\phi) \\ \text{subject to} \quad & F(\phi) = 0 \\ & G(\phi) \leq 0 \end{aligned} \quad (31)$$

In the structural optimization context, some of the most common objective functions are the volume and the compliance of the structure that need to be minimized and the natural frequencies that need to be maximized. Some of the more common constraints are the volume, the stress, and the compliance. In this Chapter, the optimization problem consists of the compliance minimization having the fiber angle  $\phi$  as the design variable and needing to satisfy the field equations of the solid mechanics, which can be written as follows:

$$\begin{aligned} \min_{\phi} \quad & J(\phi, \mathbf{u}) = \int_{\Omega} C_{ijkl}(\phi) \varepsilon_{ij}(\mathbf{u}) \varepsilon_{kl}(\mathbf{u}) dx \\ \text{subject to} \quad & F(\phi, \mathbf{u}, \mathbf{v}) = a(\phi, \mathbf{u}, \mathbf{v}) - l(\mathbf{v}) = 0 \end{aligned} \quad (32)$$

### Sensitivities

The optimization algorithms based on gradients are the most efficient approach to solving topology optimization problems. For this, it is necessary to determine the directional derivative of the Lagrangian of the problem in relation to the design variable, in this case,  $\phi$  with arbitrary increment  $\delta\phi$ . In most cases, it is worth defining the adjoint problem, which has the adjoint variable as the answer, which can make the calculation of the sensitivities easier. The adjoint variable is defined by taking the directional derivative of the Lagrangian in relation to the state variable, in this case,  $\mathbf{u}$  with an arbitrary increment  $\delta\mathbf{u}$ .

The Lagrangian of the problem can be written as

$$L(\phi, \mathbf{u}, \mathbf{v}, \lambda) = J(\phi, \mathbf{u}) - \lambda F(\phi, \mathbf{u}, \mathbf{v}) \quad (33)$$

and the directional derivative in relation to the state variable is defined as follows:

$$\begin{aligned} \delta L(\phi, \mathbf{u}, \mathbf{v}, \lambda; \delta\mathbf{u}) &= \lim_{\alpha \rightarrow 0} \frac{L(\phi, \mathbf{u} + \alpha\delta\mathbf{u}, \mathbf{v}, \lambda) - L(\phi, \mathbf{u}, \mathbf{v}, \lambda)}{\alpha} = \\ &= \left. \frac{d}{d\alpha} L(\phi, \mathbf{u} + \alpha\delta\mathbf{u}, \mathbf{v}, \lambda) \right|_{\alpha=0} = 0 \end{aligned} \quad (34)$$

From Eq. (34), it is possible to conclude that the adjoint variable can be determined as a solution of

$$\left. \frac{dJ(\phi, \mathbf{u} + \alpha\delta\mathbf{u}, \mathbf{v})}{d\alpha} \right|_{\alpha=0} - \lambda \left. \frac{dF(\phi, \mathbf{u} + \alpha\delta\mathbf{u}, \mathbf{v})}{d\alpha} \right|_{\alpha=0} = 0 \quad (35)$$

However, for the optimization problem shown in Eq. (32), it is clear that despite the state equation  $F$  being treated as an equality constraint, it is always satisfied because solving the state equation for  $\mathbf{u}$  is mandatory for all steps of the optimization. So, for the problem solved in this Chapter, the Lagrangian can be written as

$$L(\phi, \mathbf{u}, \mathbf{v}, \lambda) = J(\phi, \mathbf{u}) - \lambda \{a(\phi, \mathbf{u}, \mathbf{v}) - l(\mathbf{v})\} = J(\phi, \mathbf{u}) - 0 = J(\phi, \mathbf{u}) \quad (36)$$

Therefore, the sensitivities for the optimization problem shown in Eq. (32) can be defined as

$$\begin{aligned} \delta L(\phi, \mathbf{u}, \mathbf{v}, \lambda; \delta\phi) &= \delta J(\phi, \mathbf{u}; \delta\phi) = \lim_{\alpha \rightarrow 0} \frac{J(\phi + \alpha\delta\phi, \mathbf{u}) - J(\phi, \mathbf{u})}{\alpha} = \\ &= \left. \frac{d}{d\alpha} J(\phi + \alpha\delta\phi, \mathbf{u}) \right|_{\alpha=0} = \left. \frac{d}{d\alpha} \left( \int_{\Omega} C_{ijkl}(\phi + \alpha\delta\phi) \varepsilon_{ij}(\mathbf{u}) \varepsilon_{kl}(\mathbf{u}) dx \right) \right|_{\alpha=0} = \\ &= \int_{\Omega} \frac{dC_{ijkl}(\phi)}{d\phi} \varepsilon_{ij}(\mathbf{u}) \varepsilon_{kl}(\mathbf{u}) \delta\phi dx \end{aligned} \quad (37)$$

The derivative of the constitutive tensor in relation to the fiber angle for the plane stress in Voigt notation can be written as

$$\begin{aligned}
\frac{dC_{ijkl}(\phi)}{d\phi} &= \begin{bmatrix} DC_{11} & DC_{12} & DC_{16} \\ DC_{12} & DC_{22} & DC_{26} \\ DC_{16} & DC_{26} & DC_{66} \end{bmatrix} \\
DC_{11} &= -4(Q_{11} c_2 - Q_{22} s_2 + (Q_{12} + 2 Q_{66}) s_2 - (Q_{12} + 2 Q_{66}) c_2) s c \\
DC_{12} &= \Delta_c s_4 \\
DC_{16} &= 3(c_4 - 1)\Delta_a/8 + 3(c_4 - 1)\Delta_b/8 + \Delta_c c_2^2 + \Delta_b s_2^2 \\
DC_{22} &= 4(Q_{11} s_2 - Q_{22} c_2 - (Q_{12} + 2 Q_{66}) s_2 + (Q_{12} + 2 Q_{66}) c_2) s c \\
DC_{26} &= 3(c_4 - 1)(-\Delta_a)/8 + 3(c_4 - 1)\Delta_b/8 + (-\Delta_a s_2^2 + \Delta_b c_2^2) \\
DC_{66} &= \Delta_c s_4 \\
c &= \cos(\phi) \\
s &= \sin(\phi) \\
c_2 &= \cos^2(\phi) \\
s_2 &= \sin^2(\phi) \\
c_4 &= \cos(4\phi) \\
s_4 &= \sin(4\phi) \\
\Delta_a &= Q_{11} - Q_{12} - 2 Q_{66} \\
\Delta_b &= Q_{12} - Q_{22} + 2 Q_{66} \\
\Delta_c &= Q_{11}/2 - Q_{12}/2 + Q_{22}/2 - 2 Q_{66}
\end{aligned} \tag{38}$$

## Numerical implementation

The optimization problem Eq. (32) must be solved using an iterative method. For this, let's define the value of the design variable  $\phi$  in the next iteration as a summation of  $\phi$  in the current iteration plus the product of a direction  $d$  multiplied by a scalar  $\alpha$ , as follows:

$$\phi^{k+1} = \phi^k + \alpha d^k \tag{39}$$

Assuming that the previous value for  $\phi$  and the direction  $d$  are known, the only unknown variable is  $\alpha$ . Therefore, suppressing the dependence of  $J$  into  $u$  only to facilitate the visualization, it is possible to write an equivalent objective function as a function of only  $\alpha$

$$J(\phi^{k+1}) = J(\phi^k + \alpha d^k) = \bar{J}(\alpha) \tag{40}$$

where  $\bar{J}(\alpha)$  is the function that has only  $\alpha$  as variable. It is possible to note that for  $\alpha = 0$ ,  $J(\phi^{k+1}) = J(\phi^k + 0 d^k) = \bar{J}(\alpha) = J(\phi^k)$  Arora (2012).

Every time that  $\phi^k$  still has not been a minimum point, it is possible to reduce the value of the objective function by determining a descendent direction  $d^k$ . So, the descendent condition for the objective function  $J$  can be defined as follows Arora (2012):

$$\bar{J}(\alpha) < \bar{J}(0) \quad (41)$$

There are several ways to estimate the value of  $\alpha$ . It is possible to calculate the  $\alpha$  analytically for simple functions. However, the better approach for most optimization problems is to determine  $\alpha$  using an optimization algorithm. Since  $\bar{J}$  depends on only  $\alpha$ , an equal interval search method is a feasible approach to determine what is the  $\alpha^*$  which minimizes  $\bar{J}$  from a known point  $\phi^k$  in a known direction  $d^k$ .

The explanation of how the algorithm works is presented in Algorithm 1. The algorithm receives as input  $\phi^k$  and  $d^k$ , which are known fiber angles and descendent direction for the iteration  $k$ , an initial guess for  $\alpha$ ,  $\delta\alpha$  that is an increment for the variable  $\alpha$ ,  $\alpha_{red}$  that is a reduction factor for the increment  $\Delta\alpha$ , and  $tol$  that represents a tolerance. The algorithm's output is  $\alpha^*$ , the  $\alpha$  which minimizes the functional  $\bar{J}$  in the current iteration. The algorithm starts with  $\alpha^*$  gets the value of  $\alpha$  and with the definition of a variable  $J_{old}$  that gets the value of  $J(\phi^k)$ , which is equivalent to  $J(\phi^k + \alpha d^k)|_{\alpha=0}$ . The loop starts in line 3 and is executed until  $\Delta\alpha$  is less than  $tol$ . In each iteration, the algorithm calculates  $J_{new}$ , which is equivalent to  $J(\phi^k + \alpha d^k)$ , and compares it with  $J_{old}$ . If  $J_{new}$  is less than  $J_{old}$ , it means that the algorithm is going in the right direction, so  $\alpha^*$  and  $J_{old}$  are updated with the current values of  $\alpha$  and  $J_{new}$ , respectively. Otherwise, the algorithm is going in the wrong direction, so  $\Delta\alpha$  is reduced by the factor  $\alpha_{red}$  and  $\alpha$  is reset to  $\alpha^*$ . At the end of each iteration,  $\alpha$  is increased by  $\Delta\alpha$ .

---

**Algorithm 1:** Equal interval search method

---

**Data:**  $\phi^k, d^k, \alpha^0, \Delta\alpha, \alpha_{red}, tol$

**Result:**  $\alpha^*$

```

1  $\alpha^* \leftarrow \alpha^0;$ 
2  $J_{old} \leftarrow J(\phi^k);$ 
3 while  $\Delta\alpha > tol$  do
4    $J_{new} \leftarrow J(\phi^k + \alpha d);$ 
5   if  $J_{new} < J_{old}$  then
6      $\alpha^* \leftarrow \alpha;$ 
7      $J_{old} \leftarrow J_{new};$ 
8   else
9      $\Delta\alpha \leftarrow \Delta\alpha \alpha_{red};$ 
10     $\alpha \leftarrow \alpha^*$ 
11   $\alpha \leftarrow \alpha + \Delta\alpha$ 

```

---

Fig. 4 shows an example of the algorithm. The continuous line represents the function, the circular markers represent the approved estimations, and the diamond markers represent the reprovod estimations. For this case,  $J(\phi) = \sin(\phi^2)$ ,  $\phi = 1.5$ ,  $d \approx -1.8845$ ,  $\Delta\alpha = 5 \cdot 10^{-2}$ ,  $\alpha_{red} = 0.5$ , and  $tol = 5 \cdot 10^{-3}$ .

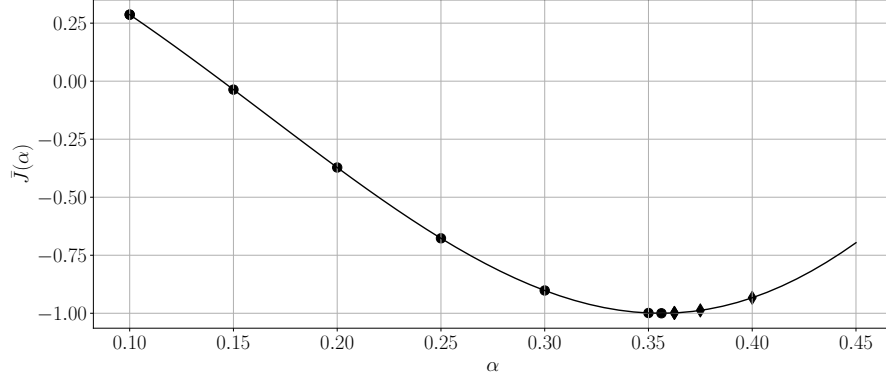


Fig. 4: Equal interval search method.

Table 1 shows the convergenve for the example presented in Fig. 4. The first column represents an improvement in the value of  $J_{new}$ ; that is,  $J_{new}^*$  signifies that the value of  $J$  decreased. The second column shows the values for  $J_{new}$ . The third column shows the values for  $\alpha$ , and the last column shows the values for  $\Delta\alpha$ . It is possible to note that the algorithm converges to  $\alpha^* = 0.3562$ , which is the value of  $\alpha$  that minimizes  $J(\phi^k + \alpha d^k) = \bar{J}(\alpha)$ .

Table 1: Convergence of equal interval search method.

	$J_{new}$	$\alpha$	$\Delta\alpha$
$J_{new}^*$	0.2866	0.1000	0.0500
$J_{new}^*$	-0.0362	0.1500	0.0500
$J_{new}^*$	-0.3720	0.2000	0.0500
$J_{new}^*$	-0.6771	0.2500	0.0500
$J_{new}^*$	-0.9019	0.3000	0.0500
$J_{new}^*$	-0.9988	0.3500	0.0500
-	-0.9333	0.4000	0.0500
-	-0.9877	0.3750	0.0250
-	-0.9986	0.3625	0.0125
$J_{new}^*$	-1.0000	0.3562	0.0063
-	-0.9986	0.3625	0.0063

There are several ways to calculate the descendent direction  $d^k$ . Here, the *Steepest Descent Algorithm* is been employed. For this, the sensitivities of the objective

function calculated in Eq. (37) is used. Therefore, the direction  $d^k = \delta J(\phi^k, \mathbf{u}; \delta \phi)$ . The value of  $\phi^{k+1}$  is updated in Eq. (39) until the following condition is satisfied:

$$\left\| \begin{bmatrix} J(\phi^{k-8}) \\ J(\phi^{k-7}) \\ \vdots \\ J(\phi^k) \\ J(\phi^{k+1}) \end{bmatrix} - \begin{bmatrix} J(\phi^{k-9}) \\ J(\phi^{k-8}) \\ \vdots \\ J(\phi^{k-1}) \\ J(\phi^k) \end{bmatrix} \right\|_{\infty} \leq \varepsilon \quad (42)$$

The idea of Eq. (42) is to compare the last ten values objective functions, and when the maximum difference between  $J(\phi^{k-i})$  and  $J(\phi^{k-i-1})$  be smaller than a tolerance  $\varepsilon$ , the convergence is considered reached.

As previously cited, the finite element method is an excellent numerical method to solve the state equation Eq. (29). This Chapter uses the open-source *FEniCS computation platform* Logg et al (2012) to deal with this task. There are a lot of advantages in using *FEniCS* to solve differential equations by using the finite element method. Among them are:

- It is an open-source platform;
- The user has total control over the equations that will be solved;
- It is not necessary to be concerned about the implementation of finite elements;
- Whole formulation is written in the weak form, very close to the mathematical formulation.

The elements used to solve the optimization problem are shown in Fig. 5. To solve the state equation, the linear Lagrange element Logg et al (2012) is used; that is, the displacements field  $\mathbf{u}$  is calculated in each node of the finite element mesh. The design variable  $\phi$  is calculated in the center of each element by using the Discontinuous Lagrange element Logg et al (2012).

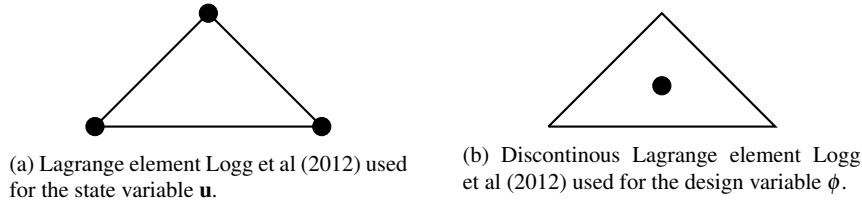


Fig. 5: Elements used to solve the optimization problem.

At this point, it is imperative to highlight a crucial detail about the sensitivities. Until here, all the formulations were written in the continuous context; that is, any discretization like that made in the finite elements method was considered. Therefore, let's note the last term inside the integral in Eq. (37). As explained in the definition of the directional derivative, the term  $\delta \phi$  represents an arbitrary increment. This arbitrary increment is considered the area of the element to condition

the numerical problem better. This choice is not arbitrary and is related to the *Riesz Representation theorem* Farrell (2021), which is not the scope of this Chapter.

## Numerical examples

Two benchmarks are used as examples to show the results of the formulation presented here, the *Cantilever beam* and the *MBB beam* shown in Fig. 6, with the dimension represented in millimeters. The *Cantilever beam* (Fig. 6a) is clamped at the left side and has a surface force applied on the right side. The *MBB beam* (Fig. 6b) is fixed at the bottom on the left side, and it is allowed to move freely on the right side in the  $x$  direction. The surface force is applied on the top of *MBB beam*.

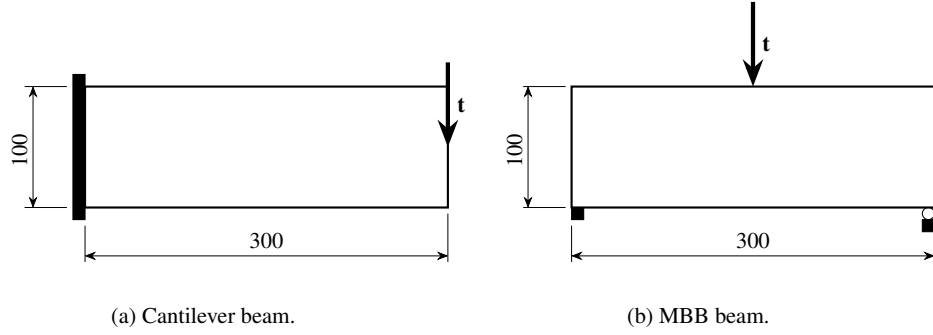


Fig. 6: Boundary conditions for numerical examples.

The material properties and numerical variables used to solve both examples are shown in Table 2 and Table 3, respectively. In both examples, the surface forces are applied in a length of 20 mm,  $10 \text{ N/mm}^2$  for the *Cantilever beam* and  $15 \text{ N/mm}^2$  for the *MBB beam*.

Table 2: Material parameters

$E_1$ [MPa]	$E_2$ [MPa]	$G_{12}$ [MPa]	$\nu_{12}$
$38.6 \cdot 10^3$	$8.17 \cdot 10^3$	$4.14 \cdot 10^3$	0.26

The evolutions of the optimization iterations for both cases are shown in Table 4. The *Cantilever beam* converged in 1923 iterations, while the *MBB beam* converged in 4750. The convergences of the objective functions for both cases are shown in Fig. 7. The codes used to obtain both results are integrally presented in Appendix A and B.



Table 3: Numerical variables

element size [mm]	$\alpha_0$	$\Delta\alpha$	$\alpha_{red}$	$tol$	$\varepsilon$	$\phi_0$
10	$1 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	0.5	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	0

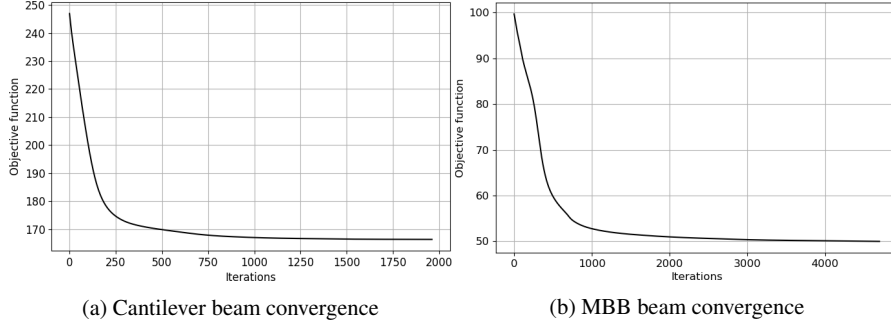


Fig. 7: Convergence of the objective functions for both examples

A simple flowchart for the optimization processes is presented in Fig. 8.

## Conclusions




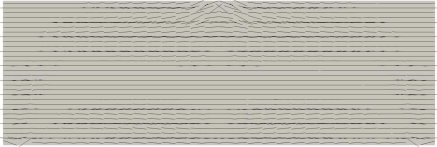
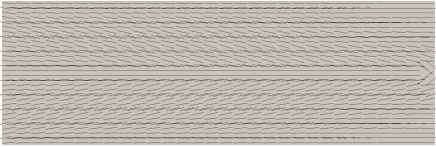

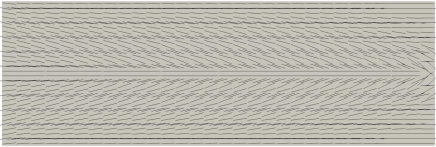
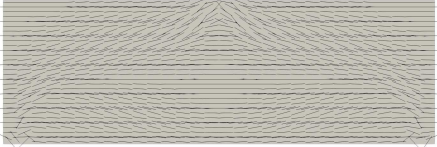
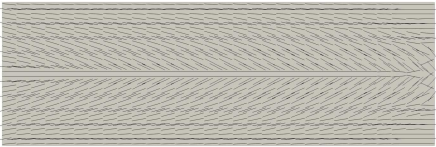
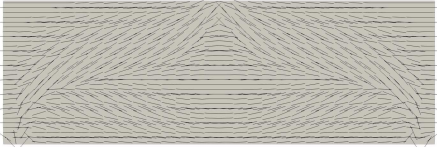
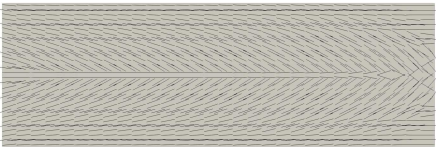
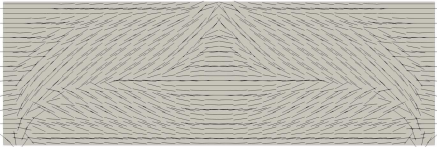
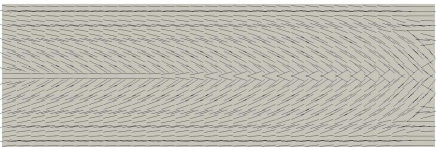
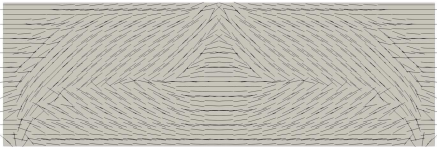
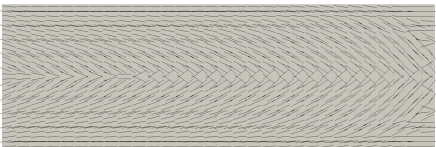
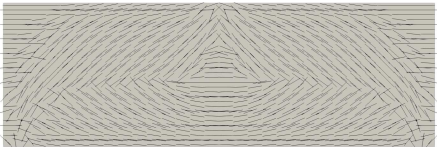
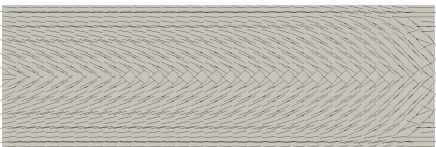
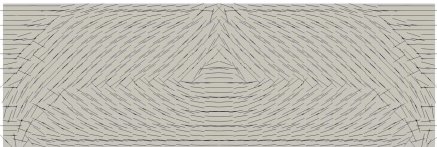
In conclusion, this chapter has comprehensively addressed the formulation for optimizing composite materials reinforced with fibers, utilizing the continuous mechanics framework. The linear elastic formulation, weak form formulation, optimization problem, and sensitivities have been systematically presented, providing a solid foundation for the understanding and application of optimization techniques in this domain.

Furthermore, the numerical implementation aspects were thoroughly discussed, shedding light on the equal interval search method and the steepest descent method.

To validate the effectiveness of the proposed formulations and numerical methods, two benchmark results were solved. These results serve as benchmarks for future studies and provide insights into the practical application of the developed optimization framework.

For the benefit of researchers and practitioners, the FEniCS code associated with the discussed formulations and numerical methods has been attached to this chapter. This code serves as a valuable resource, enabling the replication of the presented results and facilitating further exploration and development in the field of optimizing composite materials reinforced with fibers.

Table 4: Evolution of the optimization

it	Cantilever beam	it	MBB beam
0		0	
100		50	
200		100	
150		300	
250		500	
400		800	
650		1300	
1050		2100	
result		result	

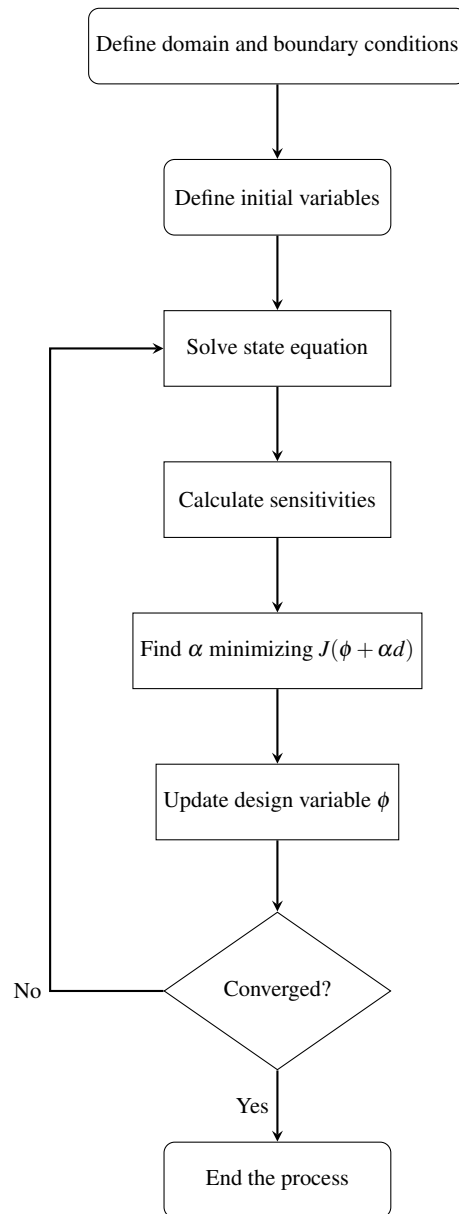


Fig. 8: Optimization flowchart

## Appendix A - Cantilever beam FEniCS code

```

from fenics import *
from fenics_adjoint import *
import numpy as np
import matplotlib.pyplot as plt

# Define the domain dimension
height = 100 # mm
width = 300 # mm

# Define the approximate size of the element
elsize = 10 # mm

# Initialize alpha
alpha = 1e-2

# Define the load
t_ = 10 # N mm-2

# Increment for alpha
delta_alpha = 1e-2

# Reduction factor
alpha_red = 0.5

# Tolerance for the bisection method convergence
tol = 1e-3

# Tolerance for the steepest descent convergence
eps = 1e-4

# Material properties
E1 = Constant(38.6e3) # MPa - Young's modulus in fiber direction
E2 = Constant(8.17e3) # MPa - Young's modulus in isotropic plane
G12 = Constant(4.14e3) # MPa - Shear modulus
nu12 = Constant(0.26) # Poisson ratio

# Define the load as a Constant vector
t = Constant((0, -t_))

# Define the number of elements
nx = int(width / elsize)
ny = int(height / elsize)

# Create the mesh
mesh = RectangleMesh(Point(0, 0), Point(width, height), nx, ny, 'crossed')

# Define space functions
V = VectorFunctionSpace(mesh, 'CG', 1)
P = FunctionSpace(mesh, 'DG', 0)

```

```

# Define the initial variables
phi = Function(P, name='Fiber angle')
u = Function(V, name='Displacement')
u_ = TrialFunction(V)
v = TestFunction(V)
phi_inc = Function(P, name='Fiber angle + delta_phi * DLa(phi)')
DL_phi = Function(P, name='DL_phi')

class Clamped(SubDomain):
    """
    Clamped boundary coordinates
    """

    def inside(self, x, on_boundary):
        """
        """
        return near(x[0], 0) and on_boundary

class Load(SubDomain):
    """
    Load coordinates
    """

    def inside(self, x, on_boundary):
        """
        """
        return near(x[0], width) and \
            between(x[1], (0.5 * height - 10, 0.5 * height + 10)) \
            and \
            on_boundary

# Create a mesh function to apply the boundary conditions
facets = MeshFunction("size_t", mesh, 1)

# Create dx and ds
dx = Measure('dx', domain=mesh)
ds = Measure('ds', domain=mesh, subdomain_data=facets)

# Setting all mesh as 0
facets.set_all(0)

# Defining different colors for Clamped, Slide and Load
Clamped().mark(facets, 1)
Load().mark(facets, 2)

# Create the Dirichlet boundary conditions
bc = DirichletBC(V, ((0., 0.)), facets, 1)

def strain(u):

```

```

    """
    Calculate the strain in Voigt notation.
    """

    eps_x = u[0].dx(0)
    eps_y = u[1].dx(1)
    gamma_xy = 0.5 * (u[0].dx(1) + u[1].dx(0))

    return as_vector((eps_x, eps_y, gamma_xy))

def reuter_matrix():
    """
    Reuter matrix used to calculate the orthotropic
    constitutive tensor
    """
    return as_tensor(((1, 0, 0), (0, 1, 0), (0, 0, 2)))

def transformation_matrix(phi):
    """
    Matrix that transforms the coordinates.
    """

    # Cossine of c
    c = cos(phi)

    # Sine of c
    s = sin(phi)

    # Tranformation matrix
    T = as_tensor((
        (c*c, s*s, 2*c*s),
        (s*s, c*c, -2*s*c),
        (-s*c, s*c, c*c-s*s)))

    return T

def orthotropic_constitutive_tensor(E1, E2, G12, nu12):
    """
    Calculate the orthotropic constitutive tensor
    """

    # Tensor S
    S11 = 1. / E1
    S12 = -nu12 / E1
    S22 = 1. / E2
    S66 = 1. / G12

    # Constitutive tensor elements
    Q11 = S22 / (S11 * S22 - S12**2)
    Q12 = - S12 / (S11 * S22 - S12**2)
    Q22 = S11 / (S11 * S22 - S12**2)
    Q66 = 1. / S66

```

```

    # Constitutive tensor
    Q = as_tensor(((Q11, Q12, 0),
                    (Q12, Q22, 0),
                    (0, 0, Q66)))

    return Q

def a(phi, u, v):
    """
    Bilinear expression for the weak formulation of the problem.
    """

    # Define the tensor to calculate the constitutive tensor C
    R = reuter_matrix()
    Q = orthotropic_constitutive_tensor(E1, E2, G12, nu12)
    T = transformation_matrix(phi)

    # Calculate the constitutive tensor C
    C = inv(T) * Q * R * T * inv(R)

    return inner(C * strain(u), strain(v)) * dx

def L(v):
    """
    Linear expression for the weak formulation of the problem.
    """
    return inner(t, v) * ds(2)

def F(phi, u, v, u_):
    """
    Solve the bilinear problem for u.
    """

    # Assemble the system
    A, b = assemble_system(a(phi, u_, v), L(v), bc)

    # Solve the system for u
    solve(A, u.vector(), b)

    return u

def compliance(phi, u):
    """
    """
    return assemble(a(phi, u, u), annotate=True)

def DC_phi(phi):
    """

```

```

...

# Orthotropic constitutive tensor
Q = orthotropic_constitutive_tensor(E1, E2, G12, nu12)
Q11 = Q[0, 0]
Q12 = Q[0, 1]
Q22 = Q[1, 1]
Q66 = Q[2, 2]

# Components of DC_phi
c = cos(phi)
s = sin(phi)
c2 = c**2
s2 = s**2
c4 = cos(4*phi)
s4 = sin(4*phi)
Delta_a = Q11 - Q12 - 2*Q66
Delta_b = Q12 - Q22 + 2*Q66
Delta_c = Q11/2 - Q12/2 + Q22/2 - 2*Q66

DC_phi11 = -4 * (Q11*c2 - Q22*s2 + (Q12 + 2*Q66)*s2 - (Q12 +
2*Q66)*c2
                ) * s * c
DC_phi12 = Delta_c * s4
DC_phi16 = 3 * (c4 - 1) * Delta_a/8 + 3 * (c4 - 1) * Delta_b
/8 + Delta_c * \
c2**2 + Delta_b * s2**2
DC_phi22 = 4 * (Q11*s2 - Q22*c2 - (Q12 + 2*Q66)*s2 + (Q12 +
2*Q66)*c2) * s \
* c
DC_phi26 = 3 * (c4 - 1) * (-Delta_a)/8 + 3 * (c4 - 1) *
Delta_b/8 + (
-Delta_a * s2**2 + Delta_b * c2**2)
DC_phi66 = Delta_c * s4

# Derivative of the constitutive tensor
return as_tensor([[DC_phi11, DC_phi12, DC_phi16],
                  [DC_phi12, DC_phi22, DC_phi26],
                  [DC_phi16, DC_phi26, DC_phi66]])

# Create *.xdmf
xdmf_file = XDMFFile('fields/fields_cg_cantilever.xdmf')

# Initialize c list
J_list = []

# Set some parameters
xdmf_file.parameters['flush_output'] = True
xdmf_file.parameters['functions_share_mesh'] = True
xdmf_file.parameters['rewrite_function_mesh'] = False

# Initialize convergence variable
conv = False

```



```

# Initialize max_diff
max_diff = '_'

# Initialize d_old
d_old = np.zeros(P.dim())
DL_phi_old = np.zeros(P.dim())

# Calculate the field of the area of elements
delta_rho = project(CellVolume(mesh), P).vector().get_local()

while not conv:

    # Calculate the displacements
    u = F(phi, u, v, u_)

    # Calculate the gradient field
    DL_phi_new = project(inner(DC_phi(phi) * strain(u), strain(u)
    ), P).vector(
    ).get_local() * delta_rho

    # Initialize alpha_best
    alpha_best = alpha

    # Old compliance (first compliance in this casa)
    J_old = compliance(phi, u)
    J_new = J_old

    DL_phi_old[:] = DL_phi_new[:]

    # Update phi new
    while delta_alpha > tol:

        # Calculate the new phi with an increment
        phi_inc.assign(phi)
        phi_inc.vector()[:] += alpha * DL_phi_new[:]

        # New compliance
        J_new = compliance(phi_inc, u)

        if J_new < J_old:

            # Update J_old
            J_old = J_new

            # Best alpha
            alpha_best = alpha

        else:

            # Update delta_alpha
            delta_alpha *= alpha_red

            # Update alpha

```

```

        alpha = alpha_best

    # Update alpha
    alpha += delta_alpha

    # Update phi_old
    phi_old_array = phi.vector().get_local()

    # Update phi
    phi.vector()[:] += alpha_best * DL_phi_new[:]

    # Save the fields of fiber angle and displacements
    if len(J_list) % 5 == 0:
        xdmf_file.write(phi, len(J_list))
        xdmf_file.write(u, len(J_list))
        xdmf_file.write(DL_phi, len(J_list))

    # Append c to the list
    J_list.append(J_new)

    # Print the iteration and objective function
    if len(J_list) <= 11:
        print('i: {:3d} - J: {:.6f} - max_diff: {}'.format(
            len(J_list), J_new, max_diff))
    else:
        print('i: {:3d} - J: {:.6f} - max_diff: {:.6e}'.format(
            len(J_list), J_new, max_diff))

    if len(J_list) > 10:

        # Calculate the convergence
        max_diff = abs(
            (np.array(J_list[-10:]) - np.array(J_list[-11:-1])).
            max())
        conv = max_diff < eps

    print('i: {:3d} - J: {:.6f} - max_diff: {:.6e}'.format(
        len(J_list), J_new, max_diff))

# Plot the convergence of the objective function
plt.plot(range(len(J_list[1:])), J_list[1:], color='black')
plt.xlabel('Iterations')
plt.ylabel('Objective function')
plt.grid()
plt.savefig('convergence_gc_cantilever.png')

```

## Appendix B - MBB beam FEniCS code

```

from fenics import *
from fenics_adjoint import *
import numpy as np
import matplotlib.pyplot as plt

# Define the domain dimension
height = 100
width = 300

# Define the approximate size of the element
elsize = 10

# Initialize alpha
alpha = 1e-2

# Define the load
t_ = 15

# Increment for alpha
delta_alpha = 1e-2

# Reduction factor
alpha_red = 0.5

# Tolerance for the bisection method convergence
tol = 1e-3

# Tolerance for the steepest descent convergence
eps = 1e-4

# Material properties
E1 = Constant(38.6e3) # MPa - Young's modulus in fiber direction
E2 = Constant(8.17e3) # MPa - Young's modulus in isotropic plane
G12 = Constant(4.14e3) # MPa - Shear modulus
nu12 = Constant(0.26) # Poisson ratio

# Define the load as a Constant vector
t = Constant((0, -t_))

# Define the number of elements
nx = int(width / elsize)
ny = int(height / elsize)

# Create the mesh
mesh = RectangleMesh(Point(0, 0), Point(width, height), nx, ny, 'crossed')

# Define space functions
V = VectorFunctionSpace(mesh, 'CG', 1)
P = FunctionSpace(mesh, 'DG', 0)

# Define the initial variables
phi = Function(P, name='Fiber angle')
u = Function(V, name='Displacement')

```

```

u_ = TrialFunction(V)
v = TestFunction(V)
phi_inc = Function(P, name='Fiber angle + delta_phi * DL_a(phi)')
DL_phi = Function(P, name='DL_phi')

class Clamped(SubDomain):
    """
    Clamped boundary coordinates
    """
    def inside(self, x, on_boundary):
        """
        """
        return between(x[0], (0., 10.)) and near(x[1], 0) and
            on_boundary

class Slide(SubDomain):
    """
    Slide boundary coordinates
    """
    def inside(self, x, on_boundary):
        """
        """
        return between(x[0], (width - 10, width)
            ) and near(x[1], 0) and on_boundary

class Load(SubDomain):
    """
    Load coordinates
    """
    def inside(self, x, on_boundary):
        """
        """
        return near(x[1], height) and \
            between(x[0], (0.5 * width - 10, 0.5 * width + 10))
            and \
            on_boundary

# Create a mesh function to apply the boundary conditions
facets = MeshFunction("size_t", mesh, 1)

# Create dx and ds
dx = Measure('dx', domain=mesh)
ds = Measure('ds', domain=mesh, subdomain_data=facets)

# Setting all mesh as 0
facets.set_all(0)

```

```

# Defining different colors for Clamped, Slide and Load
Clamped().mark(facets , 1)
Load().mark(facets , 2)
Slide().mark(facets , 3)

# Create the Dirichlet boundary conditions
bc1 = DirichletBC(V, ((0., 0.)), facets , 1)
bc2 = DirichletBC(V.sub(1), 0., facets , 3)
bc = [bc1, bc2]

def strain(u):
    """
    Calculate the strain in Voigt notation.
    """
    eps_x = u[0].dx(0)
    eps_y = u[1].dx(1)
    gamma_xy = 0.5 * (u[0].dx(1) + u[1].dx(0))

    return as_vector((eps_x, eps_y, gamma_xy))

def reuter_matrix():
    """
    Reuter matrix used to calculate the orthotropic
    constitutive tensor
    """
    return as_tensor(((1, 0, 0), (0, 1, 0), (0, 0, 2)))

def transformation_matrix(phi):
    """
    Matrix that transforms the coordinates.
    """
    # Cossine of c
    c = cos(phi)

    # Sine of c
    s = sin(phi)

    # Tranformation matrix
    T = as_tensor((
        (c*c, s*s, 2*c*s),
        (s*s, c*c, -2*s*c),
        (-s*c, s*c, c*c-s*s))

    return T

def orthotropic_constitutive_tensor(E1, E2, G12, nu12):
    """
    Calculate the orthotropic constitutive tensor

```

```

    """
    # Tensor S
    S11 = 1. / E1
    S12 = -nu12 / E1
    S22 = 1. / E2
    S66 = 1. / G12

    # Constitutive tensor elements
    Q11 = S22 / (S11 * S22 - S12**2)
    Q12 = - S12 / (S11 * S22 - S12**2)
    Q22 = S11 / (S11 * S22 - S12**2)
    Q66 = 1. / S66

    # Constitutive tensor
    Q = as_tensor(((Q11, Q12, 0),
                    (Q12, Q22, 0),
                    (0, 0, Q66)))

    return Q

def a(phi, u, v):
    """
    Bilinear expression for the weak formulation of the problem.
    """

    # Define the tensor to calculate the constitutive tensor C
    R = reuter_matrix()
    Q = orthotropic_constitutive_tensor(E1, E2, G12, nu12)
    T = transformation_matrix(phi)

    # Calculate the constitutive tensor C
    C = inv(T) * Q * R * T * inv(R)

    return inner(C * strain(u), strain(v)) * dx

def L(v):
    """
    Linear expression for the weak formulation of the problem.
    """
    return inner(t, v) * ds(2)

def F(phi, u, v, u_):
    """
    Solve the bilinear problem for u.
    """

    # Assemble the system
    A, b = assemble_system(a(phi, u_, v), L(v), bc)

    # Solve the system for u
    solve(A, u.vector(), b)

```

```

        return u

def compliance(phi, u):
    """
    """
    return assemble(a(phi, u, u), annotate=True)

def DC_phi(phi):
    """
    """

    # Orthotropic constitutive tensor
    Q = orthotropic_constitutive_tensor(E1, E2, G12, nu12)
    Q11 = Q[0, 0]
    Q12 = Q[0, 1]
    Q22 = Q[1, 1]
    Q66 = Q[2, 2]

    # Components of DC_phi
    c = cos(phi)
    s = sin(phi)
    c2 = c**2
    s2 = s**2
    c4 = cos(4*phi)
    s4 = sin(4*phi)
    Delta_a = Q11 - Q12 - 2*Q66
    Delta_b = Q12 - Q22 + 2*Q66
    Delta_c = Q11/2 - Q12/2 + Q22/2 - 2*Q66

    DC_phi11 = -4 * (Q11*c2 - Q22*s2 + (Q12 + 2*Q66)*s2 - (Q12 +
        2*Q66)*c2
        ) * s * c
    DC_phi12 = Delta_c * s4
    DC_phi16 = 3 * (c4 - 1) * Delta_a/8 + 3 * (c4 - 1) * Delta_b
        /8 + Delta_c * \
        c2**2 + Delta_b * s2**2
    DC_phi22 = 4 * (Q11*s2 - Q22*c2 - (Q12 + 2*Q66)*s2 + (Q12 +
        2*Q66)*c2) * s \
        * c
    DC_phi26 = 3 * (c4 - 1) * (-Delta_a)/8 + 3 * (c4 - 1) *
        Delta_b/8 + (
        -Delta_a * s2**2 + Delta_b * c2**2)
    DC_phi66 = Delta_c * s4

    # Derivative of the constitutive tensor
    return as_tensor([[DC_phi11, DC_phi12, DC_phi16],
        [DC_phi12, DC_phi22, DC_phi26],
        [DC_phi16, DC_phi26, DC_phi66]])

# Create *.xdmf

```

```

xdmf_file = XDMFFile('fields/fields_cg_mbb.xdmf')

# Initialize c list
J_list = []

# Set some parameters
xdmf_file.parameters['flush_output'] = True
xdmf_file.parameters['functions_share_mesh'] = True
xdmf_file.parameters['rewrite_function_mesh'] = False

# Initialize convergence variable
conv = False

# Initialize max_diff
max_diff = '-'

# Initialize d_old
d_old = np.zeros(P.dim())
DL_phi_old = np.zeros(P.dim())

# Calculate the field of the area of elements
delta_rho = project(CellVolume(mesh), P).vector().get_local()

while not conv:

    # Calculate the displacements
    u = F(phi, u, v, u_)

    # Calculate the gradient field
    DL_phi_new = project(inner(DC_phi(phi) * strain(u), strain(u)), P).vector().get_local() * delta_rho

    # Initialize alpha_best
    alpha_best = alpha

    # Old compliance (first compliance in this casa)
    J_old = compliance(phi, u)
    J_new = J_old

    DL_phi_old[:] = DL_phi_new[:]

    # Update phi new
    while delta_alpha > tol:

        # Calculate the new phi with an increment
        phi_inc.assign(phi)
        phi_inc.vector()[:] += alpha * DL_phi_new[:]

        # New compliance
        J_new = compliance(phi_inc, u)

        if J_new < J_old:

```



```

        # Update J_old
        J_old = J_new

        # Best alpha
        alpha_best = alpha

    else:

        # Update delta_alpha
        delta_alpha *= alpha_red

        # Update alpha
        alpha = alpha_best

    # Update alpha
    alpha += delta_alpha

# Update phi_old
phi_old_array = phi.vector().getLocal()

# Update phi
phi.vector()[:] += alpha_best * DL_phi_new[:]

# Save the fields of fiber angle and displacements
if len(J_list) % 10 == 0:
    xdmf_file.write(phi, len(J_list))
    xdmf_file.write(u, len(J_list))
    xdmf_file.write(DL_phi, len(J_list))

# Append c to the list
J_list.append(J_new)

# Print the iteration and objective function
if len(J_list) <= 11:
    print('i: {:3d} - J: {:.6f} - max_diff: {}'.format(
        len(J_list), J_new, max_diff))
else:
    print('i: {:3d} - J: {:.6f} - max_diff: {:.6e}'.format(
        len(J_list), J_new, max_diff))

if len(J_list) > 10:

    # Calculate the convergence
    max_diff = abs(
        (np.array(J_list[-10:]) - np.array(J_list[-11:-1])).
        max())
    conv = max_diff < eps

print('i: {:3d} - J: {:.6f} - max_diff: {:.6e}'.format(
    len(J_list), J_new, max_diff))

# Plot the convergence of the objective function
plt.plot(range(len(J_list[1:])), J_list[1:], color='black')
```

```
plt.xlabel('Iterations ')\nplt.ylabel('Objective function ')\nplt.grid()\nplt.savefig('convergence_gc_mbb.png')
```

## References

- Arora JS (2012) Introduction to optimum design, 3rd edn. MATLAB examples, Elsevier Academic Press, Amsterdam, literaturverz. S. 841 - 850
- Bendsøe MP, Sigmund O (2004) Topology Optimization. Springer Berlin Heidelberg, DOI 10.1007/978-3-662-05086-6
- Farrell P (2021) Finite element methods for pdes. URL <https://people.maths.ox.ac.uk/farrellp/femvideos/notes.pdf>, c6.4 Hilary Term, University of Oxford
- Kaw AK (2006) Mechanics of composite materials, 2nd edn. No. 29 in Mechanical engineering series, CRC Taylor & Francis, Boca Raton, FL u.a.
- Lai WM (2010) Introduction to continuum mechanics. Butterworth-Heinemann/Elsevier, Amsterdam, includes bibliographical references (pages 511-512) and index
- Logg A, Ølgaard KB, Rognes ME, Wells GN (2012) FFC: the FEniCS form compiler, Springer Berlin Heidelberg, pp 227–238. DOI 10.1007/978-3-642-23099-8\_11
- Silva ALFd, Salas RA, Silva ECN (2023) Topology optimization of fiber reinforced structures considering stress constraint and optimized penalization. Composite Structures 316:117,006, DOI 10.1016/j.compstruct.2023.117006