ISSN: 0871-7869

Civil Engineering Research and Innovation for Sustainability

Development of a Web APP for the Assessment of General Stack FRP Laminates Subjected to Different Loading Conditions

F. Nunes; M. Garrido; J.R.Correia

- Janeiro de 2020 -

Relatório CERIS
DTC nº 5/2020

INSTITUTO SUPERIOR TÉCNICO

DEVELOPMENT OF A WEB APP FOR THE ASSESSMENT OF GENERAL STACK FRP LAMINATES SUBJECTED TO DIFFERENT LOADING CONDITIONS

Authors	FRANCISCO NUNES, MÁRIO GARRIDO, JOÃO RAMÔA CORREIA
Project	ECOCOMPOSITE – Development of eco-efficient bio-composites for civil engineering structural applications (PTDC/ECI-EGC/29597/2017)
Task	2. Multi-criteria optimization of composite layups
Report no.	2.1.
Date	January 2020

This work was funded by FCT – *Fundação para a Ciência e a Tecnologia, I.P.* within the framework of the EcoComposite project (PTDC/ECI-EGC/29597/2017).

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Overview

This report presents the development of a web-app for the assessment of general stack FRP laminates subjected to different loading conditions using the rule of mixtures, the Classical Laminate Theory (CLT) and a progressive failure analysis called the 'ply discount method'. It comprises the following three main sections:

- theoretical background;
- app development;
- user guide.

In the first section, a thorough theoretical background is presented, describing the mathematical formulation behind the development of the app. It covers all the topics included in the app which are (i) raw materials, (ii) rule of mixtures, (iii) CLT, (iv) Halpin-Tsai formulation, (v) initial failure and (vi) progressive failure analysis. The following section presents an overview of how the app was developed, covering the technologies, programming languages and frameworks used to develop the back-end, the front-end and the database management of the app. Finally, the last section provides detailed instructions and guidelines on how to use the app.

1.2 Motivation

There were three main motivations for the development of this web-app. The first was related to the need of developing a script that would be comprehensive, easy to use and able to interact with other applications, such as Excel or MATLAB, so as to enable its integrated use in the project's optimization related tasks. As the main goal of the project is to develop new bio-based resins to be incorporated in FRP composites, the suitability of the aforementioned theoretical tools (rules of mixtures, CLT, etc.) needs to be assessed for laminates incorporating such new resins. Accordingly, this web-app provides a simple and straightforward way of applying such tools, the results of which can subsequently be experimentally validated. Lastly, another main objective was to develop an app that could be used not only within the framework of the project, but also disseminated over the scientific community. The requirements set prior to the development were that it should be user-friendly and have an appealing design. It should also comprise a database of raw materials and laminates that can be updated by users over time.

1.3 Methodology

The methodology for the development of the app was divided in four steps: development of the back-end, development of the front-end, development of the database management framework and deployment to an online server.

The development of the back-end scripts was made through the use of the Python 3 programming language within the *Spyder IDE* (integrated development environment). Whenever possible, Object Oriented Programming (OOP) was used, which basically treats all the elements as objects with different attributes and functions that can be applied to themselves. A total of five objects (Python classes) were created: resin, fibre, fabric, lamina and laminate, each one with its own features. This type of programming allows for a more readable code and it is also easier to understand than function-based programming.

The front-end was developed using *Dash* by *Plotly*, a framework that allows to build fully interactive apps directly in the browser with little knowledge of HTML and using only Python scripts.

The database management was carried out using Structured Query Language (SQL), within the *PostgreSQL* framework and the Python library *psycopg2*. This library allows executing SQL code within three lines of code, directly in the Python script by creating a connection, a cursor and an execution command.

Finally, the app was deployed to an online server provider called *Heroku*, which has a free-tier service that allows for the app to be online 24/7, with a user-defined domain and enough computing power for production purposes.

2 THEORETICAL BACKGROUND

2.1 Materials

Composite FRP laminates comprise two main constituent materials: fibres and resin. While the fibres provide strength and stiffness along their direction, the polymeric resin provides support to the fibres maintaining them in the same position, distributing the stresses uniformly across the composite lamina and protecting them from the environmental agents. Other materials, such as fillers and additives may also be added to the resin matrix to improve the manufacturing process, to improve/change specific properties (e.g. colour requirements, fire reaction) or simply for cost reduction [1].

One of the types of fibre reinforcement most widely used in civil engineering and construction applications are glass fibres (used in glass fibre reinforced polymer, or GFRP, products). However, other types of fibre reinforcement (such as carbon fibres, CF) may also be used in some specific applications, mainly when higher stiffness is required, such as in several structural strengthening applications. E-glass (electric glass) fibres are the most commonly used in the production of FRP pultruded profiles, which in turn are the most common type of FRP structural element used in construction. This is greatly due to the cost-effectiveness of the E-glass fibre reinforcement. Other special types of glass fibres are available, such as S-glass (structural glass) fibres, which are mainly limited to aerospace composites as their cost is significantly higher than E-glass. All glass fibres present high electrical resistivity, namely when comparing to carbon fibres. Carbon fibres are stiffer than glass fibres, but are also more expensive and electrically conductive. There are several types of carbon fibres available in the market, such as the standard modulus (SM) type, which is the most commonly used across most applications. Ultra-high modulus (UHM) carbon fibres are considerably stiffer than the standard modulus ones [1], but are significantly costlier.

The polymeric resins most often used in the production of FRP composites are unsaturated polyester, vinyl ester and epoxy. Unsaturated polyester resins are a very cost-effective solution as they are the most economical, are easily processed and present reasonably good mechanical properties [2]. Epoxy on the other hand is the most expensive polymeric matrix. It presents very high mechanical properties, durability and low shrinkage, namely when compared with unsaturated polyester. They are thus mainly used in carbon fibre reinforced polymer (CFRP) composites. Vinyl ester results from the esterification of epoxy groups with acrylic or methacrylic acids and presents intermediate characteristics relative to epoxy and unsaturated polyester resins; namely, it presents a reasonable balance between durability, mechanical properties, ease of processing and cost [1].

The relatively high environmental footprint of conventional oil-based resins together with their obvious dependency from non-renewable resources have led manufacturers to revise their production materials to more sustainable options. The analysis of the product lifecycle is pushing these manufacturers away from the oil-andgas and back to renewable resources, such as plants and biomass. However, most of these efforts are still insufficient, with the incorporation rates of bio-based feedstock typically being lower than 50% in the final products available in the market. Nonetheless, bio-based resins are then emerging as a sustainable alternative to traditional oil-based resins. One of the biggest challenges for the production of structural elements with bio-based resins is the cost-effectiveness of such resins (current products are still expensive) and obviously the need to assess their functional efficiency (e.g., relative to their stiffness, strength and durability in harsh environments) in comparison with conventional resins. Another relevant aspect is the need to understand in depth the interactions between bio-based resin matrices and reinforcing fibres aiming at developing high performance bio-composites.

In the following sub-sections, one will go through the process of computing the properties of a composite laminate based on its constituent materials using the rule of mixtures, the CLT and a progressive failure analysis.

2.2 Rule of mixtures

FRP structures might be analysed at four different scales (Figure 1): (i) micro – at the material (fibre and matrix) level and the interaction between the two, (ii) lamina – considering a given fibre volume and orientation, (iii) laminate – considering a stack of laminae with different orientations, thicknesses or even reinforcement, and (iv) structure – considering the full section of the laminates and the full-section equivalent properties. In the scope of this report, one is mostly concerned with the lamina and laminate levels.

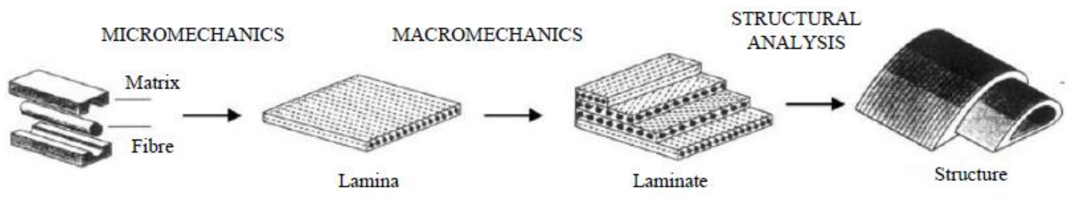


Figure 1 – Different levels of analysis [1].

Since both the fibres and the resin are assumed to be isotropic, the following set of equations show how the shear modulus (*G*) and the shear strength (*τ*) are obtained from the remaining properties (elastic modulus, *E*, Poisson ratio, *ν*, and axial strength, *σ*).

$$G = \frac{E}{2(1+\nu)} \tag{1}$$

$$\tau = \frac{\sigma}{\sqrt{3}} \tag{2}$$

At the lamina level, knowing the isotropic properties of the resin and the fibre reinforcement, one may use the rule of mixtures [1] to define homogeneous equivalent properties for the whole lamina (often referred to as engineering constants, due to the orthotropic behaviour that characterises FRP), such as the longitudinal elastic modulus (E_L), the transverse elastic modulus (E_T), the shear modulus (G_{LT}) and the Poisson's ratios (major, ν_{LT}, and minor, ν_{TL}). The following set of equations defines all the mentioned properties,

$$E_L = \upsilon_f E_f + \upsilon_m E_m \tag{3}$$

$$\frac{1}{E_T} = v_f \frac{1}{E_f} + v_m \frac{1}{E_m} \tag{4}$$

$$\frac{1}{G_{LT}} = v_f \frac{1}{G_f} + v_m \frac{1}{G_m} \tag{5}$$

$$\nu_{LT} = v_f \nu_f + v_m \nu_m \tag{6}$$

$$\nu_{LT} E_T = \nu_{TL} E_L \tag{7}$$

in which E_f and G_f are the fibre elastic and shear moduli, E_m and G_m are the resin elastic and shear moduli, ν_f and ν_m are the fibre and resin Poisson's ratios and v_f and v_m are the fibre and resin volume fractions. The following equation provides the relation between the thickness (t in mm) of the lamina and the fibre volume fraction for a given areal weight of the fibre mats or fabrics.

$$t = \frac{m_f}{\rho_f v_f} \times 10^3 \tag{8}$$

In the above equation m_f is the areal weight of the fibres, ρ_f is the fibre density and v_f is the fibre volume.

2.3 Classical Laminate Theory (CLT)

With a given laminae stacking sequence, the resulting laminate's equivalent properties can be estimated using the Classical Laminate Theory (CLT) [1]. One should note that, due to the through-thickness heterogeneity, the axial and flexural stiffness may differ widely and thus calculations often address the laminate stiffness matrix rather than the engineering constants. The laminate stiffness matrix is usually defined as the A-B-D (6×6) matrix, in which the A, B and D (sub-)matrices correspond respectively to the axial, coupling and flexural stiffness. For symmetric stacking sequences, the B matrix is null and the axial and flexural behaviours might be addressed separately. First, the actual stiffness of the lamina is computed relatively to the longitudinal direction of the laminate.

$$Q_{LT} = \begin{bmatrix} Q_{xx} & Q_{xy} & 0 \\ Q_{yx} & Q_{yy} & 0 \\ 0 & 0 & Q_{ss} \end{bmatrix} = \begin{bmatrix} \frac{E_L}{1-\nu_{LT}\nu_{TL}} & \frac{\nu_{TL}E_L}{1-\nu_{LT}\nu_{TL}} & 0 \\ \frac{\nu_{LT}E_T}{1-\nu_{LT}\nu_{TL}} & \frac{E_T}{1-\nu_{LT}\nu_{TL}} & 0 \\ 0 & 0 & G_{LT} \end{bmatrix} \tag{9}$$

Next, according to the angle of the fibre direction (θ) and using the appropriate transformation matrix, the stiffness of the lamina is converted from the orientation of the lamina to the orientation of the laminate. The compliance matrix (S_{ij}) is then computed by inverting the lamina oriented stiffness matrix (Q_{ij}).

$$\begin{Bmatrix} Q_{11} \\ Q_{22} \\ Q_{12} \\ Q_{66} \\ Q_{16} \\ Q_{26} \end{Bmatrix} = \begin{bmatrix} \cos^4\theta & \sin^4\theta & 2\cos^2\theta\sin^2\theta & 4\cos^2\theta\sin^2\theta \\ \sin^4\theta & \cos^4\theta & 2\cos^2\theta\sin^2\theta & 4\cos^2\theta\sin^2\theta \\ \cos^2\theta\sin^2\theta & \cos^2\theta\sin^2\theta & \cos^4\theta+\sin^4\theta & -4\cos^2\theta\sin^2\theta \\ \cos^2\theta\sin^2\theta & \cos^2\theta\sin^2\theta & -2\cos^2\theta\sin^2\theta & (\cos^2\theta+\sin^2\theta)^2 \\ \cos^3\theta\sin\theta & -\cos\theta\sin^3\theta & \cos\theta\sin^3\theta-\cos^3\theta\sin\theta & 2(\cos\theta\sin^3\theta-\cos^3\theta\sin\theta) \\ \cos\theta\sin^3\theta & -\cos^3\theta\sin\theta & \cos^3\theta\sin\theta-\cos\theta\sin^3\theta & 2(\cos^3\theta\sin\theta-\cos\theta\sin^3\theta) \end{bmatrix} \begin{Bmatrix} Q_{xx} \\ Q_{yy} \\ Q_{xy} \\ Q_{ss} \end{Bmatrix} \tag{10}$$

$$Q_{ij}=\begin{bmatrix}Q_{11} & Q_{12} & Q_{16}\\ Q_{21} & Q_{22} & Q_{26}\\ Q_{61} & Q_{62} & Q_{66}\end{bmatrix} \tag{11}$$

$$S_{ij}=Q_{ij}^{-1} \tag{12}$$

$$E_1=\frac{1}{S_{11}} \tag{13}$$

$$\nu_{12}=-\frac{S_{21}}{S_{11}} \tag{14}$$

$$\nu_{61}=\frac{S_{61}}{S_{11}} \tag{15}$$

$$E_2=\frac{1}{S_{22}} \tag{16}$$

$$\nu_{21}=-\frac{S_{12}}{S_{22}} \tag{17}$$

$$\nu_{62}=\frac{S_{62}}{S_{22}} \tag{18}$$

$$E_6=\frac{1}{S_{66}} \tag{19}$$

$$\nu_{16}=\frac{S_{16}}{S_{66}} \tag{20}$$

$$\nu_{26}=\frac{S_{26}}{S_{66}} \tag{21}$$

At the laminate level, one is mostly concerned in obtaining the A-B-D matrix, which relates the strain array of the laminate with the force per unit width array [1]. It may be computed as follows:

$$\begin{Bmatrix}N_1\\ N_2\\ N_6\\ M_1\\ M_2\\ M_6\end{Bmatrix}=\begin{bmatrix}A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16}\\ A_{21} & A_{22} & A_{26} & B_{21} & B_{22} & B_{26}\\ A_{61} & A_{62} & A_{66} & B_{61} & B_{62} & B_{66}\\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16}\\ B_{21} & B_{22} & B_{26} & D_{21} & D_{22} & D_{26}\\ B_{61} & B_{62} & B_{22} & D_{61} & D_{62} & D_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_1^0\\ \varepsilon_2^0\\ \varepsilon_6^0\\ \kappa_1\\ \kappa_2\\ \kappa_6\end{Bmatrix} \tag{22}$$

$$A_{ij}=\sum_{k=1}^{n}Q_{ij}^{k}(z_k-z_{k-1})=\sum_{k=1}^{n}Q_{ij}^{k}h_k \tag{23}$$

$$B_{ij}=\frac{1}{2}\sum_{k=1}^{n}Q_{ij}^{k}(z_k^2-z_{k-1}^2)=\sum_{k=1}^{n}Q_{ij}^{k}(-\bar{z}_k h_k) \tag{24}$$

$$D_{ij}=\frac{1}{3}\sum_{k=1}^{n}Q_{ij}^{k}(z_k^3-z_{k-1}^3)=\sum_{k=1}^{n}Q_{ij}^{k}\left(h_k\bar{z}_k{}^2+\frac{h_k^3}{12}\right) \tag{25}$$

2.4 Halpin-Tsai formulation

The Halpin-Tsai equations [3,4] are said to be more accurate in the determination of the transverse and shear moduli than the CLT. Thus, these equations may be included in the analyses by overwriting the corresponding values that result from CLT. The Halpin-Tsai formulation consists on the application of the following equations,

$$\frac{M}{M_m} = \frac{1 + \xi \eta v_f}{1 - \eta v_f} \tag{26}$$

$$\eta = \frac{\left(\frac{M_f}{M_m} - 1\right)}{\left(\frac{M_f}{M_m} - \xi\right)} \tag{27}$$

in which *M* is the property being analysed, the subscripts *c*, *m* and *f* stand for composite, matrix and fibre, respectively, v_f is the fibre volume and ξ is an empirical value which corresponds to a measure of reinforcement of the composite material, dependent on the fibre geometry and loading. According to the specific bibliography, one may adopt the value of ξ = 2.0 for the transverse elastic modulus [4] and ξ = 1.0 for the shear modulus [3].

2.5 Initial failure and progressive failure analysis

To analyse the failure behaviour of the laminate, one should first define an appropriate initial failure index, which is a parameter that accounts for the current stress state and yields a value between 0.0 and 1.0 to indicate how much more loading the laminate can sustain before reaching the strength of the limiting lamina (initial failure). Some of the most widely used failure initiation criteria are the maximum stress index or the maximum strain index [1], the Tsai-Hill index [5] and the Hashin damage initiation index [6,7]. In this report, one will only address the failure indices used in the web-app, which are the maximum stress index and the Tsai-Hill index.

The first step of a failure analysis is computing the stresses in each lamina due to a given loading condition (using the ABD matrix shown in section 2.3) and transforming those stresses from the laminate orientation to the lamina orientation using the transformation matrix shown below.

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_s \end{Bmatrix} = \begin{bmatrix} \cos^2\theta & \sin^2\theta & 2\cos\theta\sin\theta \\ \sin^2\theta & \cos^2\theta & -2\cos\theta\sin\theta \\ -\cos\theta\sin\theta & \cos\theta\sin\theta & \cos^2\theta - \sin^2\theta \end{bmatrix} \begin{Bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_6 \end{Bmatrix} \tag{28}$$

With the stresses oriented in the direction of each lamina, the maximum stress index, as the name implies, consists of computing a ratio between the given stress and the corresponding strength. The maximum stress index of each lamina-stress pair is assumed to be the maximum stress index of the laminate.

Regarding the Tsai-Hill index, it takes into account all the stresses and strength values in its formulation, which is computed in each lamina according to the following equation.

$$I_F^{Tsai-Hill} = \frac{\sigma_x^2}{\sigma_{u,x}^2} - \frac{\sigma_x \sigma_y}{\sigma_{u,x}^2} + \frac{\sigma_y^2}{\sigma_{u,y}^2} + \frac{\sigma_s^2}{\sigma_{u,s}^2} \tag{29}$$

The progressive failure analysis consists of applying the ply discount method to the laminate stacking and evaluating the respective load-strain path. This method consists of three main steps. The first one is to incrementally increase the load until reaching the limit of the failure index (1.0) in one or more laminae. Next, a new ABD matrix is computed (for the laminate without the discounted laminae that reached the failure index of 1.0). Finally, the force array is calculated based of the strain state of the previous step, using the new ABD matrix. These three steps are looped until all the laminae are discounted from the analysis and the force array reaches 0. It is worth mentioning that this approach involves the simplifying assumption that there is no post-failure capacity after a given lamina attains the failure index.

3 SOFTWARE DEVELOPMENT

3.1 Back-end

The back-end of the web application was developed using the Python 3 programming language [8] and importing some useful libraries, such as *NumPy* [9] or *Pandas* [10]. The programming scripts were developed using Object Oriented Programming (OOP) whenever possible, defining the relevant object classes, attributes and functions that would allow for a straightforward and readable implementation of the problem. All the objects, functions and other relevant snippets of code are highlighted in `grey colour with light grey background`. The back-end coding script is presented in the Appendix.

3.1.1 Materials

The first step of the development was the definition of the raw materials: `Resin` and `Fibre`. For each material, the developer needs to specify the longitudinal elastic modulus (*E*), the Poisson's ratio and the strength as mandatory arguments. The density argument is optional and if not specified it is assumed to be 1200 kg/m^3 for the resin and 2500 kg/m^3 for the fibre.

Afterwards, one defines the `Fabric` object, which basically consists of a layer of fibres with orthogonal orientations weighted in a certain proportion. The arguments of the class are the fibre that it is made of the areal weight in the longitudinal direction and the areal weight in the transverse direction (in g/m^2). If the `Fabric` object is used in a lamina instead of a `Fibre` object, the thickness is computed automatically according to the rule of mixtures (section 2.2).

3.1.2 Lamina

The `Lamina` object was defined taking into account the following arguments: resin, fibre, volume, angle (with the longitudinal direction of the laminate) and the Halpin-Tsai Boolean variable (set to `False` by default). A certain number of attributes were set in order to fully define the `Lamina` object. The interested reader should check the Appendix for the detailed code snippet. Also, three functions with different goals were defined and are addressed in the following paragraph.

The `compute_moduli()` function uses the properties from the resin and the fibre materials and computes the longitudinal, transverse and shear elastic moduli of the lamina, as well as the Poisson's ratios according to the specified fibre volume by applying the rule of mixtures. The Halpin-Tsai equations (section 2.4) are used for the transverse and shear elastic moduli if the Boolean attribute `self.halpin_tsai` is set to `True`. The `compute_matrices()` function transforms the elastic properties of the lamina into the laminate longitudinal orientation angle using the appropriate transformation matrix.

3.1.3 Laminate

The `Laminate` object is instantiated without passing in any arguments. All of the analysis is done by the corresponding object functions. The `add_lamina()` function adds a single lamina to the `Laminate` object from the bottom up. It takes in the `Lamina` object to be passed onto the laminate, the thickness of that given lamina and the Boolean variable `compute_properties,` which is set to `True` by default. If the `compute_properties` argument is set to `False`, the script will not compute the laminate properties after adding that particular lamina. The developer should be aware of the fact that, while using such approach significantly minimizes the computational times spent in the laminate instantiation, the last lamina object that is added to the laminate should always have `compute_properties` set to `True`, otherwise, all the stiffness properties of the laminate will not be correct.

Similarly, the `add_multiple_laminae()` function adds a set of laminae to the laminate rather than a single one. It takes in the following parameters: resin, fibre, laminae thickness list, laminae angle list and the Halpin-Tsai Boolean variable.

The `compute_ABD_matrix()` and the `compute_effective_moduli()` do not take in any arguments and, as their names state, they are used to compute the laminate ABD matrix and the effective elastic properties under pure tension and simple bending.

The `compute_stress_state()` function takes in the force per unit width parameters (N_{11}, N_{22}, N_{66}, M_{11}, M_{22} and M_{66}) and computes the corresponding stress state of the laminate. The output consists of a *Pandas* DataFrame with results in the bottom and top faces of each lamina with the following parameters (columns):

- Height (m);
- Angle of the lamina (º);
- Strain 11 (m/m);
- Strain 22 (m/m);
- Strain 66 (m/m);
- Stress 11 (MPa);
- Stress 22 (MPa);
- Stress 66 (MPa);
- Stress L (MPa);
- Stress T (MPa);
- Stress LT (MPa);
- Maximum stress;
- Tsai-Hill index.

These DataFrame results are then used is subsequent functions and for building charts in the front-end script of the web-app. The `compute_maximum_stress_index()` and `compute_tsai_hill_index()` functions check what is the maximum failure index in the laminate for a given stress state. The functions take in only the DataFrame computed in `compute_stress_state()`.

The `get_max_force()` function computes the failure initiation force of the laminate (i.e., when the considered failure index reaches 1.0). It is only designed to analyse a single force component (either N_{11}, N_{22}, N_{66}, M_{11}, M_{22} or M_{66}) and it takes in the failure index target value (default is 1.0), the force component (the default is N_{11}), the force sign (the default is `plus` but it can be set to `minus`) the type (Tsai-Hill or Maximum stress) and the tolerance of the error (default is 10^{-3}).

The `get_max_force_list_factor()` presents a similar scope to that of the `get_max_force()` but it computes the maximum factor for a proportional force array rather than a single component. It takes in the force array as a list, the index target value (default is 1.0), the index type (Tsai-Hill or Maximum stress) and the tolerance of the error (default is 10^{-3}).

3.1.4 Other useful functions

This section presents other useful functions that may be used in the analysis of laminates. The `compute_load_strain_path()` function takes in the `Laminate` object, the force component to be analysed and the failure index type, and yields the force-strain path of that laminate object up to failure. Each step consists basically of a loop of procedures to account for the stress states of the laminate up to the point when all the laminae reach failure. The `Laminate` object starts off the unloaded stress state. Next the `get_max_force()` function is used to compute the maximum force that the laminate will attain with the current stack configuration. Then, the lamina or laminae that reach the failure index are discounted from the analysis. For that purpose, and since removing the lamina from the laminate would be a complex programming task, the stiffness of the laminae to be removed are multiplied by a factor of 10^{-6}, making their contribution negligible in the analysis. Finally, the `get_max_force()` function is used for the new stacking configuration. This function also outputs, for each step, several other parameters to be used in the front-end charts, namely:

- A Boolean list to check if a given lamina is already removed from the analysis or not;
- A list of the laminae bottom and top face heights relatively to the bottom face of the laminate;
- The maximum stress index along the height of the laminate;
- The Tsai-Hill index along the height of the laminate.

3.2 Front-end

The front-end of the web application was entirely developed using the *Dash* by *Plotly* [14] framework and *Bootstrap 4* style sheets. *Dash* is a web framework that allows developers to create seamless interaction between user input data and output text, charts or tables. *Dash* abstracts away all of the technologies and protocols that are required to build an interactive web-based application. After being ready, the app can be deployed to an online server and then shared through a URL. Since *Dash* apps are viewed in the web browser, *Dash* is inherently cross-platform and mobile ready. In Figure 2 one may find some examples available in the *Dash* gallery webpage.

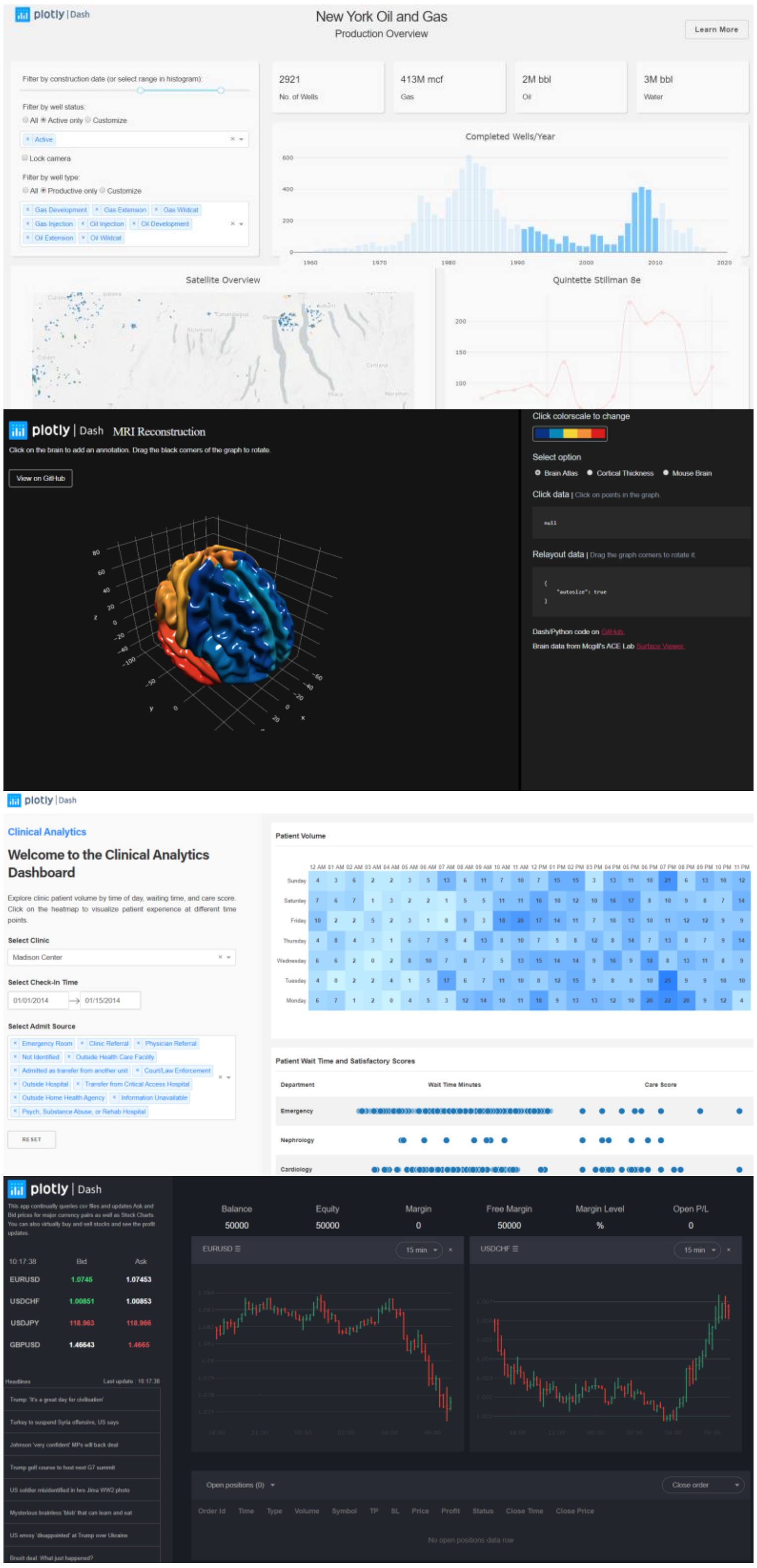
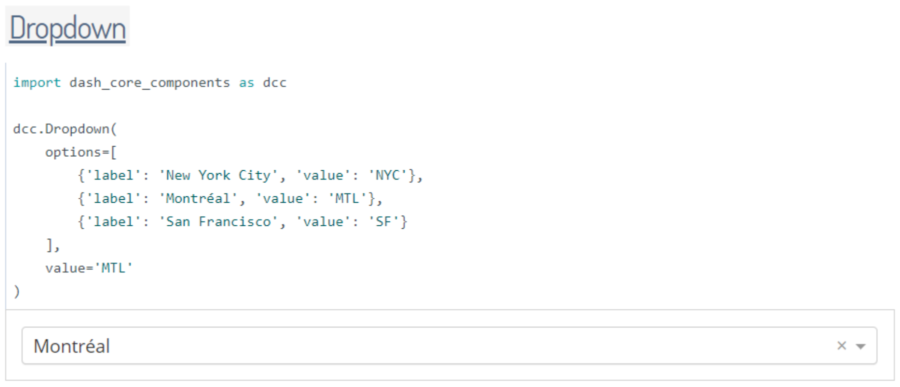


Figure 2 – Dash app examples.

Dash is built using *Plotly*, which is an open source graphing library that allows developing complex JavaScript interactive charts using only pure Python, R or MATLAB code. The charts' features include hovering over data, zoom-in, zoom-out, pan, download chart as ".png", live update, animations, among others [14]. Its chart types range from scientific to statistical, financial or choropleth maps, 2D or 3D with extensive customisation possibilities to fit the needs of the developer and the users.

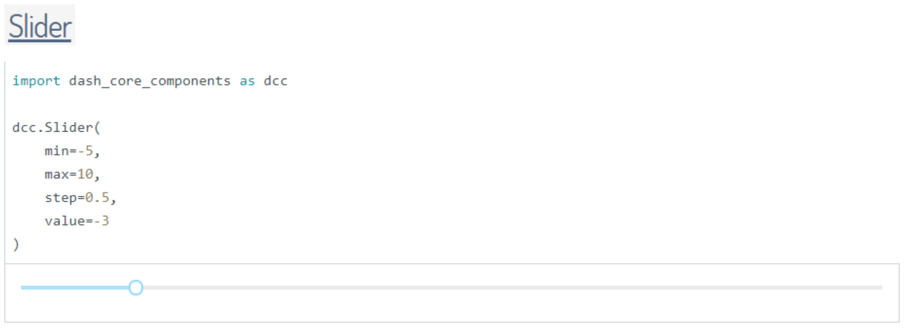
Figure 3 to Figure 12 show the main *Dash* app components with a code snippet implementation example. *Dash* apps basically consist of two parts: “Layout” and “Callbacks”. Layout, as its name implies, is the part in which the developer sets the “skeleton” of the app regarding its graphic design. All the components and HTML that will be used in the app are defined in this part. Callbacks is the part in which the developer sets how the app reacts to user interactivity. This may be through the update of charts, data tables or simply HTML text.


Dropdown

```
import dash_core_components as dcc

dcc.Dropdown(
    options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': 'Montréal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value='MTL'
)
```

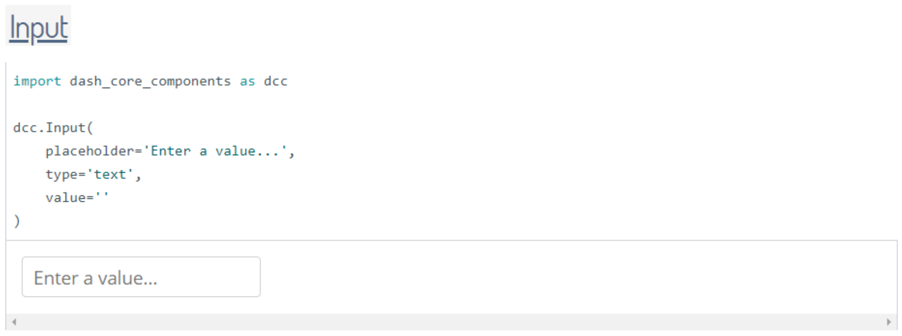
Montréal

Figure 3 – Dropdown component.


Slider

```
import dash_core_components as dcc

dcc.Slider(
    min=-5,
    max=10,
    step=0.5,
    value=-3
)
```

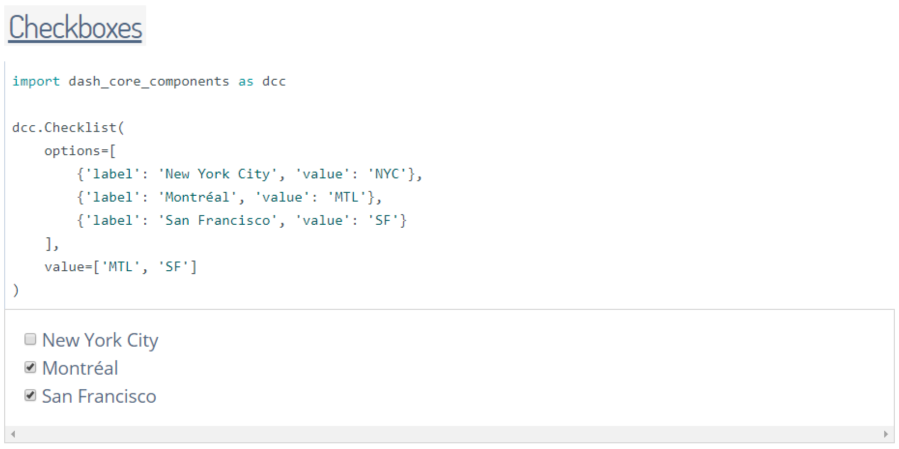
Figure 4 – Slider component.


Input

```
import dash_core_components as dcc

dcc.Input(
    placeholder='Enter a value...',
    type='text',
    value=''
)
```

Enter a value...

Figure 5 – Input component

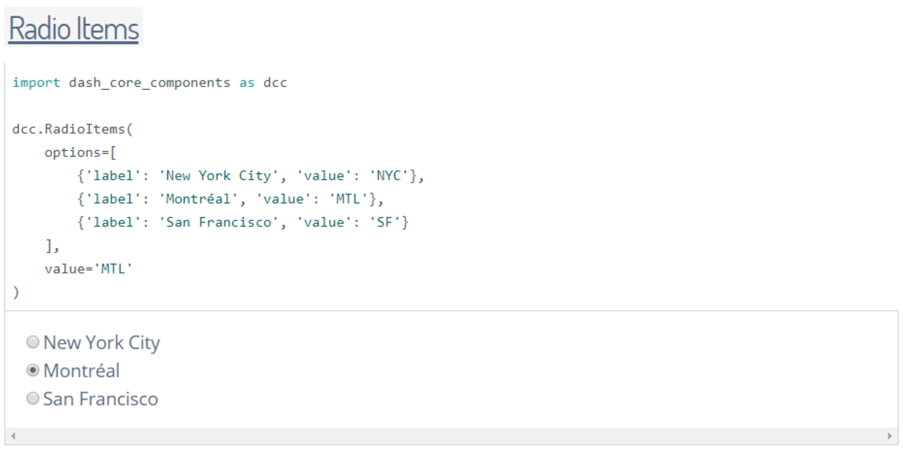


Checkboxes

```
import dash_core_components as dcc

dcc.Checklist(
    options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': 'Montréal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value=['MTL', 'SF']
)
```

- [ ] New York City
- [x] Montréal
- [x] San Francisco

Figure 6 – Checkboxes component.

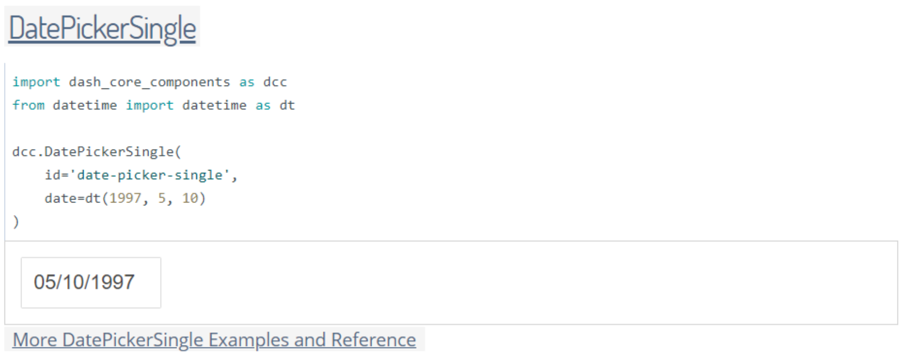


Radio Items

```
import dash_core_components as dcc

dcc.RadioItems(
    options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': 'Montréal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value='MTL'
)
```

○ New York City
● Montréal
○ San Francisco

Figure 7 – Radioboxes component.



DatePickerSingle

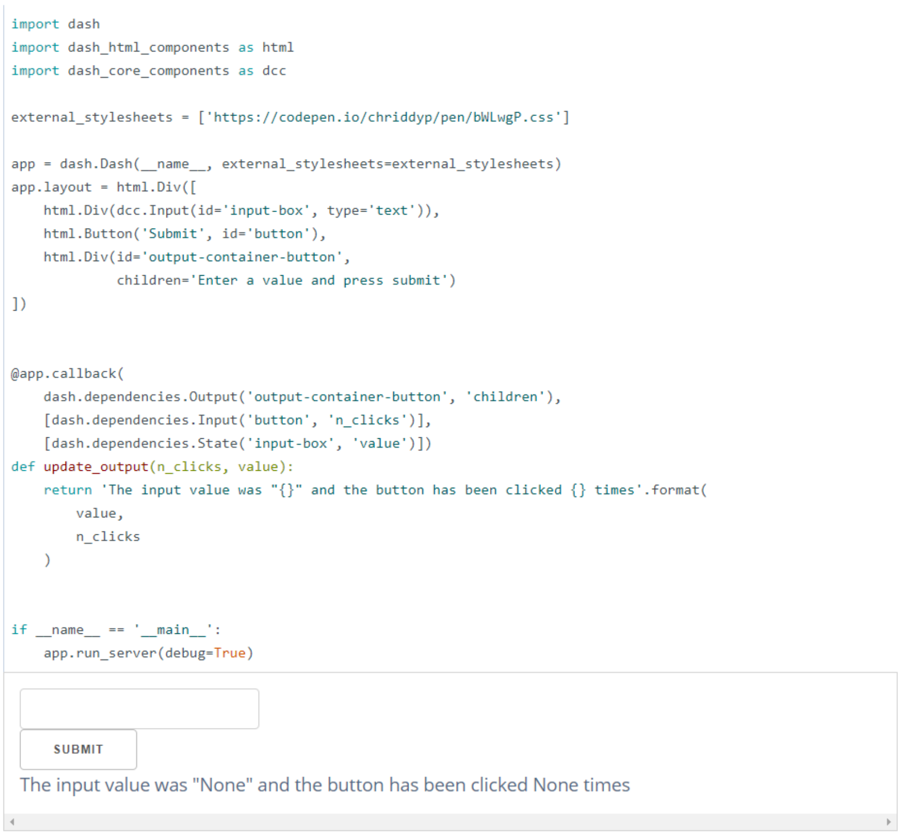
```
import dash_core_components as dcc
from datetime import datetime as dt

dcc.DatePickerSingle(
    id='date-picker-single',
    date=dt(1997, 5, 10)
)
```

05/10/1997

More DatePickerSingle Examples and Reference

Figure 8 – Date picker component

Button



```python
import dash
import dash_html_components as html
import dash_core_components as dcc

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div([
    html.Div(dcc.Input(id='input-box', type='text')),
    html.Button('Submit', id='button'),
    html.Div(id='output-container-button',
             children='Enter a value and press submit')
])


@app.callback(
    dash.dependencies.Output('output-container-button', 'children'),
    [dash.dependencies.Input('button', 'n_clicks')],
    [dash.dependencies.State('input-box', 'value')])
def update_output(n_clicks, value):
    return 'The input value was "{}" and the button has been clicked {} times'.format(
        value,
        n_clicks
    )


if __name__ == '__main__':
    app.run_server(debug=True)
```

The input value was "None" and the button has been clicked None times

Figure 9 – Button component

Upload Component

The `dcc.Upload` component allows users to upload files into your app through drag-and-drop or the system's native file explorer.

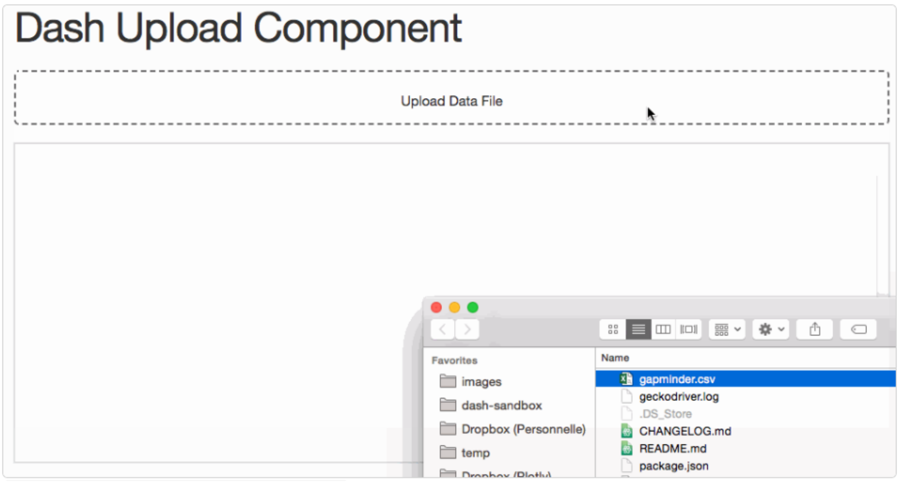


Figure 10 – Upload file component.

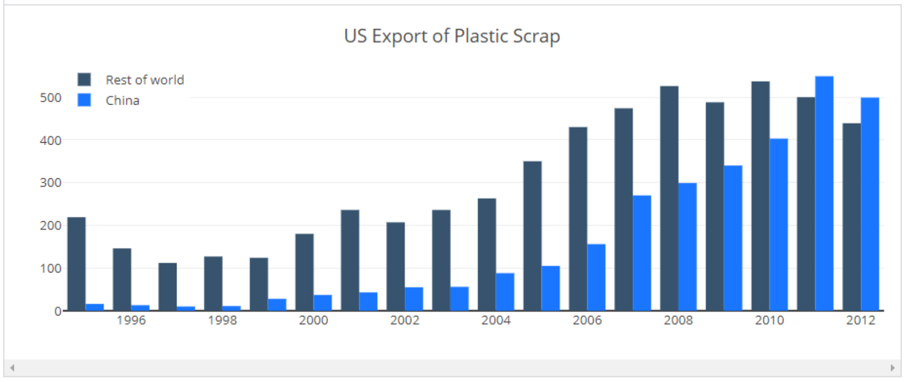
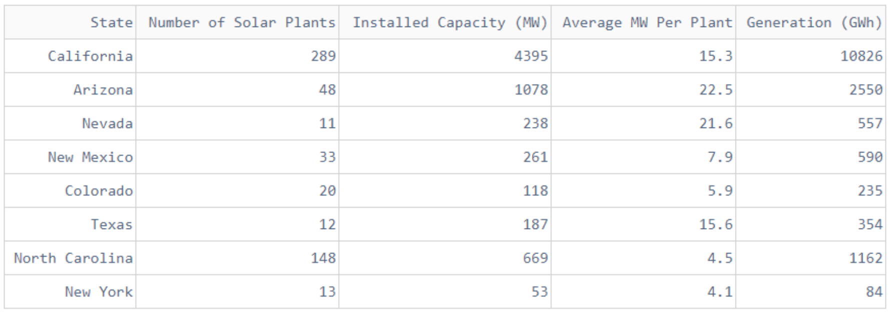


Figure 11 – Plotly graph component.



State	Number of Solar Plants	Installed Capacity (MW)	Average MW Per Plant	Generation (GWh)
California	289	4395	15.3	10826
Arizona	48	1078	22.5	2550
Nevada	11	238	21.6	557
New Mexico	33	261	7.9	590
Colorado	20	118	5.9	235
Texas	12	187	15.6	354
North Carolina	148	669	4.5	1162
New York	13	53	4.1	84

Figure 12 – Data table component.

Regarding the *Dash* app that was developed within the framework of the EcoComposite project and which is the scope of this report, one will not get into much detail as it presents a rather complex implementation that is beyond the scope of this report. It features over 1600 lines of code with URL mapping, hidden elements, stored data between callbacks and thus it would require the reader to be fully into the *Dash* framework for understanding those concepts[1].

3.3 Database management

The database management was developed using Structured Query Language or SQL (using the PostgreSQL framework) and a Python library `psycopg2`. Two different SQL tables were defined: `Fibres` and `Resin`. For each table, 8 different columns were set:

- `id` – unique labelling for each resin/fibre (type integer sequential);
- `name` – name of the resin/fibre for easier identification (type text);
- `elasticmodulus` – isotropic elastic modulus in GPa (type numeric);
- `poisson` – Poisson's ratio (type numeric);
- `strength` – isotropic tensile strength in MPa (type numeric);
- `density` – density of the material in kg/m^3 (type numeric).

[1] The interested reader may contact Francisco Nunes (francisco.nunes@tecnico.ulisboa.pt) or Mário Garrido (mario.garrido@tecnico.ulisboa.pt) for further information on the Dash app development.

- `author` – username of the author who created the material (type text);
- `datecreated` – the date in which the material was created (type date).

By using the `psycopg2` library the developer is able to use SQL commands and interact with the database by using an additional Python script. One defined several functions to accomplish all the desired interaction between the user and the SQL database. Each python function is briefly described next and the SQL code snippet is shown below.

The `insert_new_resin()` and `insert_new_fibre()` functions insert a new resin material in the `Resins` and `Fibres` tables, respectively. They take in the name, the elastic modulus, the Poisson's ratio, the strength, the density, and the author arguments. The `id` is computed automatically since it is a sequential variable and the date created is filled using the `NOW()` SQL function.

```
execute('INSERT INTO "Resins" (name, elasticmodulus, poisson, strength, density,
author, datecreated) VALUES (%s,%s,%s,%s,%s,%s, NOW())', (name, elastic_modulus,
poisson, strength, density, author)

execute('INSERT INTO "Fibres" (name, elasticmodulus, poisson, strength, density,
author, datecreated) VALUES (%s,%s,%s,%s,%s,%s, NOW())', (name, elastic_modulus,
poisson, strength, density, author))
```

The `read_all_resins()` and `read_all_fibres()` functions read all the resins or fibres to compute the material manager tables of the *Dash* app (presented ahead in Figure 16). They are also used on the dropdown resin selector of the *Dash* app (presented ahead in Figure 14).

```
execute('SELECT * FROM "Resins";')

execute('SELECT * FROM "Fibres";')
```

The `delete_resin()` and `delete_fibre()` functions take in the argument `id` and delete the corresponding resin or fibre from the `Resins` or `Fibres` tables, respectively.

```
execute('DELETE FROM "Resins" WHERE Id = {};'.format(Id))

execute('DELETE FROM "Fibres" WHERE Id = {};'.format(Id))
```

4 USER GUIDE

4.1 Overview

The web application interface was named LAYUP | FRP Laminate Analysis. As seen in Figure 13, the LAYUP application is structured in three different tabs:

- Materials;
- Laminate;
- Analysis.

The Materials tab allows choosing the raw materials that compose the laminate (resin and fibres) to be analysed. It also comprises the options to add new materials with given properties and to enter the material manager for both the resin and fibre.

The Laminate tab is where the stacking sequence and fibre volume are provided. From the raw materials defined in the former tab, the user is able to define a build-up sequence of laminae with the desired properties.

Finally, the user is prompted with the Analysis tab. Herein, for a given force array, the user is allowed to analyse the stresses along the laminate thickness as well as the corresponding failure indices. Moreover, the progressive failure of the given laminate is addressed using the ply discount method described earlier in section 2.5.

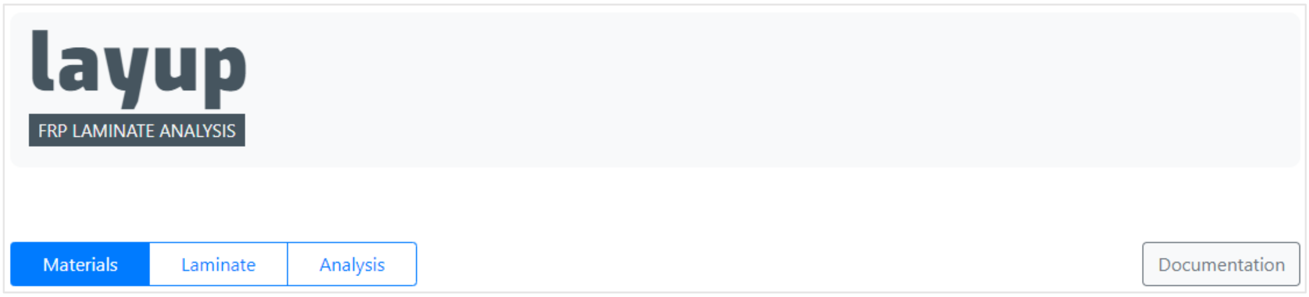


Figure 13 – Web application banner and navigation tabs.

4.2 Materials

As mentioned, the Materials tab allows choosing the resin and fibres that compose the laminate to be analysed. Each material is considered to be isotropic and has four different properties that are used in the CLT and progressive failure analyses:

- Elastic modulus;

- Poisson’s ratio;
- Strength;
- Density.

The materials may be chosen from the database by selecting in the dropdown bar or the user may also fill in the properties, for user-defined materials (Figure 14).

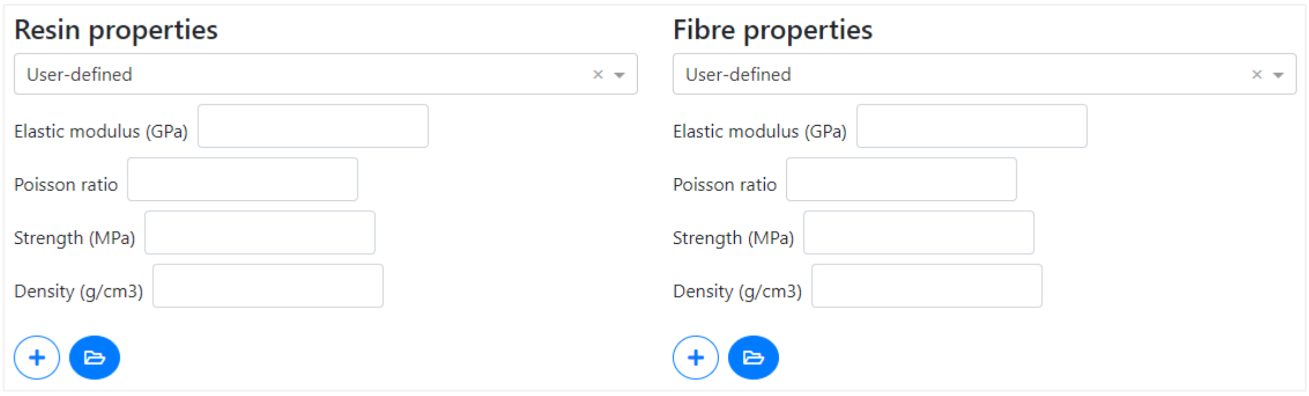
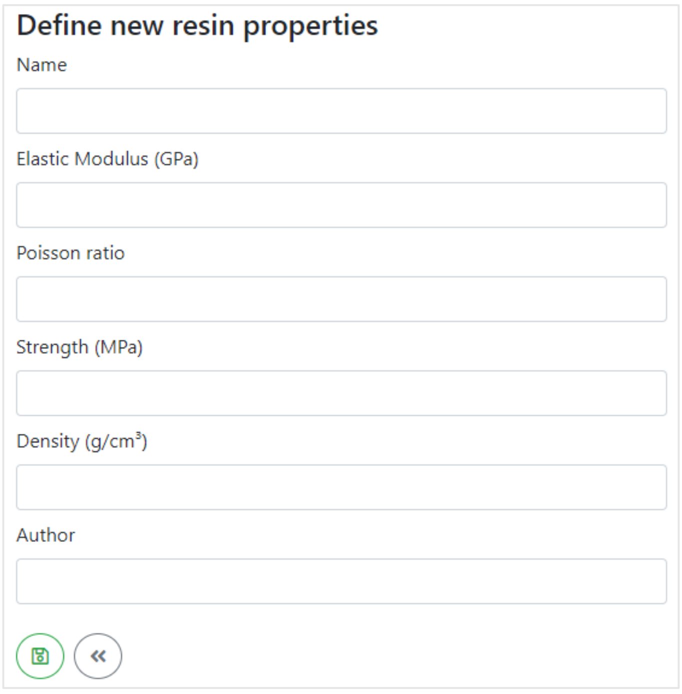

Resin properties		Fibre properties	
User-defined		User-defined	
Elastic modulus (GPa)		Elastic modulus (GPa)	
Poisson ratio		Poisson ratio	
Strength (MPa)		Strength (MPa)	
Density (g/cm3)		Density (g/cm3)	

Figure 14 – Material selection panel.

Upon clicking the Add Material button, ⊕, the user is prompted with a page that allows saving a new material to the database (Figure 15). The user should provide information for all available fields to save the new material. This new material can be used in later analyses carried out by each of the app’s users.


Define new resin properties

Name

Elastic Modulus (GPa)

Poisson ratio

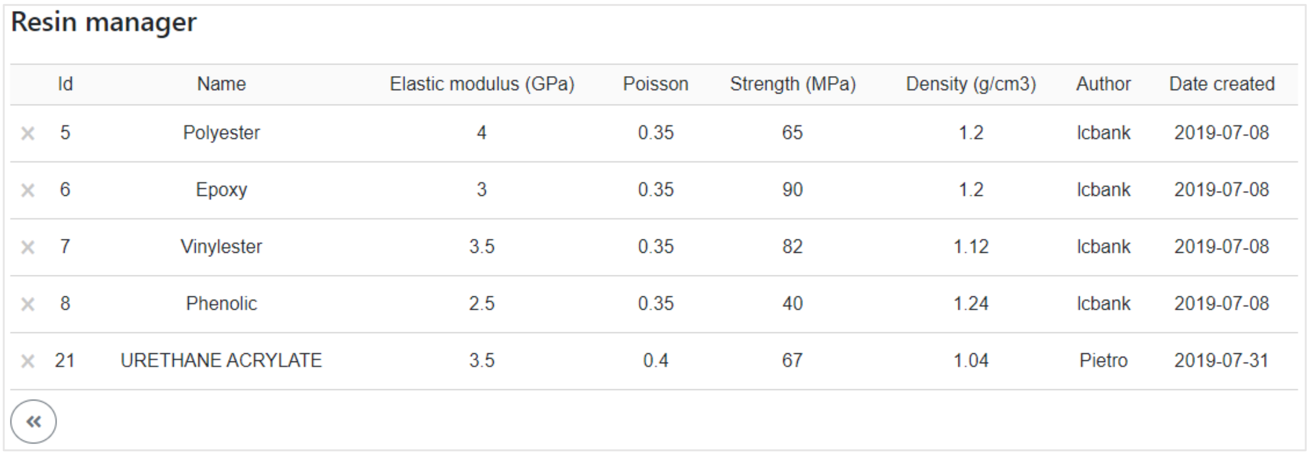
Strength (MPa)

Density (g/cm^3)

Author

Figure 15 – New material page dialog.

The Material Manager button, 🗁, prompts the user with the database of all resins and fibres stored in the app (Figure 16). Here one can see the list of all the properties for each resin and fibre and also delete materials. It should be highlighted that the deletion of a given material affects all users of the application and not just the current user. The Material Manager should be used with caution and mostly for information purposes. This issue should be fixed in a future update of the app in which a user-based specific login will be developed and implemented, so that each user has its own credentials and permissions. After choosing the materials desired for the composite analysis, the user should navigate to the Laminate tab.



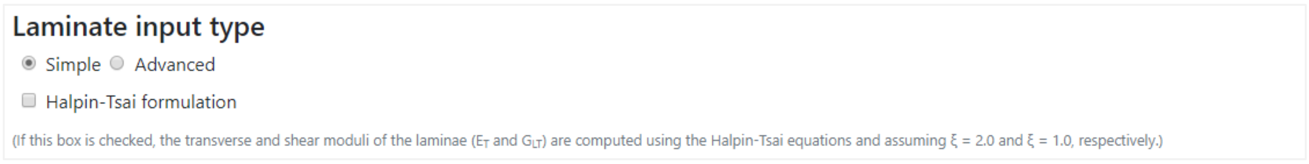
Resin manager

	Id	Name	Elastic modulus (GPa)	Poisson	Strength (MPa)	Density (g/cm3)	Author	Date created
×	5	Polyester	4	0.35	65	1.2	lcbank	2019-07-08
×	6	Epoxy	3	0.35	90	1.2	lcbank	2019-07-08
×	7	Vinylester	3.5	0.35	82	1.12	lcbank	2019-07-08
×	8	Phenolic	2.5	0.35	40	1.24	lcbank	2019-07-08
×	21	URETHANE ACRYLATE	3.5	0.4	67	1.04	Pietro	2019-07-31

Figure 16 – Material Manager data table.

4.3 Laminate

The Laminate tab prompts the user with two different types of data input: Simple and Advanced. Also, the user is allowed to choose between a basic rule of mixtures approach for the definition of all lamina properties or to use the Halpin-Tsai equations (see section 2.4) for the definition of the transverse elastic and shear moduli (Figure 17).



Laminate input type

[✓ Simple] Advanced

☐ Halpin-Tsai formulation

(If this box is checked, the transverse and shear moduli of the laminae (E_T and G_{LT}) are computed using the Halpin-Tsai equations and assuming ξ = 2.0 and ξ = 1.0, respectively.)

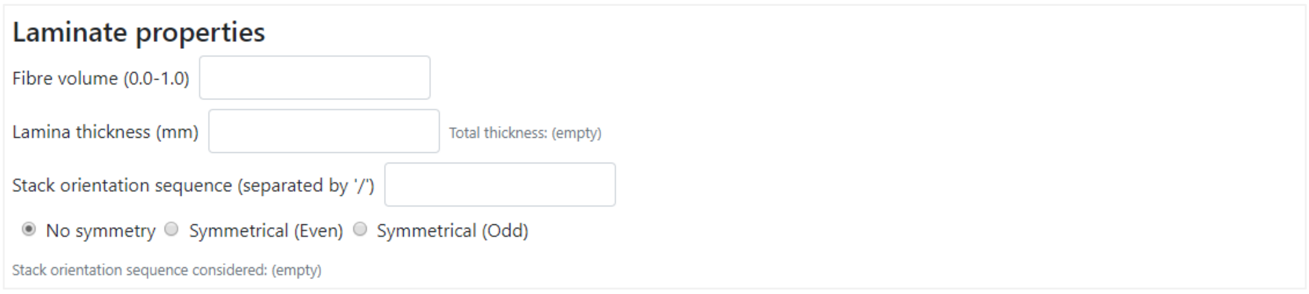
Figure 17 – Laminate input types and Halpin-Tsai formulation option.

4.3.1 Simple mode

The Simple mode consists of only four fields for data input (Figure 18):

- Fibre volume: the ratio between the volume of fibres to the volume of the entire laminate;
- Lamina thickness: the thickness of each lamina (all laminae are considered to be of equal thickness);
- Stack orientation sequence: the angle orientation of each lamina w.r.t. the laminate longitudinal axis. The user should input each angle separated by a slash (/). The stacking is considered to be from the bottom up;
- Symmetry options: the user must choose between **No symmetry** – the stacking is considered to be exactly the one given in the Stack orientation sequence field; **Symmetrical (Even)** – the stacking is

considered to be symmetrical, replicating all the laminae orientations in the mirrored sequence; **Symmetrical (Odd)** – the stacking is considered to be symmetrical, replicating all the laminae orientations in the mirrored sequence except for the last lamina in the user-input stacking sequence (which will become the middle lamina of the resulting laminate).


Laminate properties

Fibre volume (0.0-1.0)	
Lamina thickness (mm)	Total thickness: (empty)
Stack orientation sequence (separated by '/')	

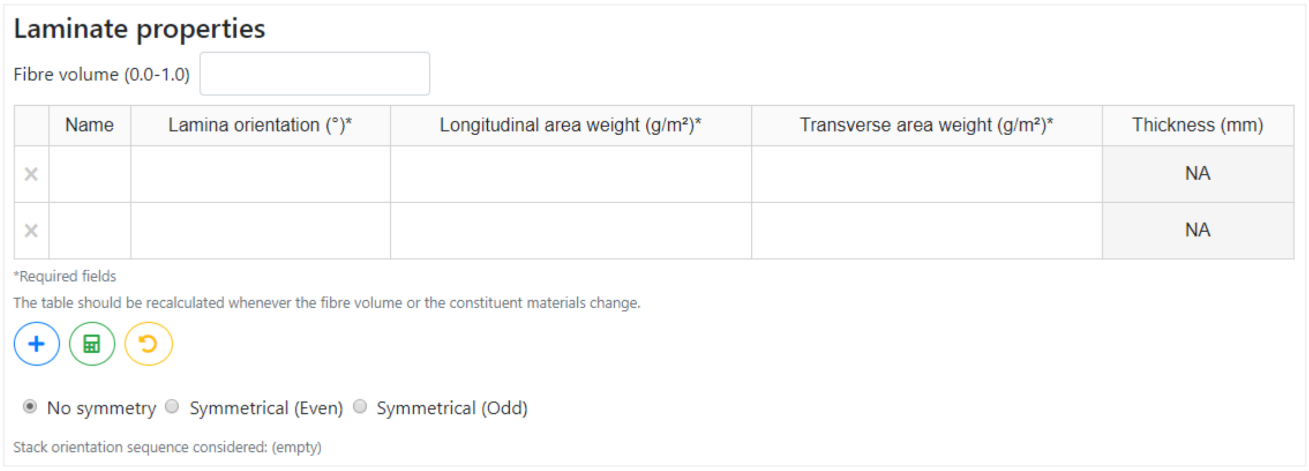
[✓ No symmetry] Symmetrical (Even) Symmetrical (Odd)

Stack orientation sequence considered: (empty)

Figure 18 – Laminate stacking options for simple mode.

4.3.2 Advanced mode

The <u>Advanced</u> mode should be used instead of the <u>Simple</u> mode if the user is required to use bidirectional laminae in the analysis and if it has access to the areal weight of each layer of fibre reinforcement. The user should fill in the cells of the data table presented in Figure 19 with the name, orientation, longitudinal and transverse areal weights of each lamina. Their thickness is then computed automatically according to the formulation presented in section 2.2. Once again, the stacking is considered to be from the bottom up (the first row of the data table corresponds to the bottommost row of the laminate and so on) and the symmetry options are the same as in the simple mode (section 4.3.1). The table is complemented with three buttons below. The first, ⊕, adds a new layer, the second, ▦, recalculates all the values if the fibre volume has changed, and the third, ↺, clears the whole table.


Laminate properties

Fibre volume (0.0-1.0)

	Name	Lamina orientation (°)*	Longitudinal area weight (g/m²)*	Transverse area weight (g/m²)*	Thickness (mm)
×					NA
×					NA

*Required fields

The table should be recalculated whenever the fibre volume or the constituent materials change.

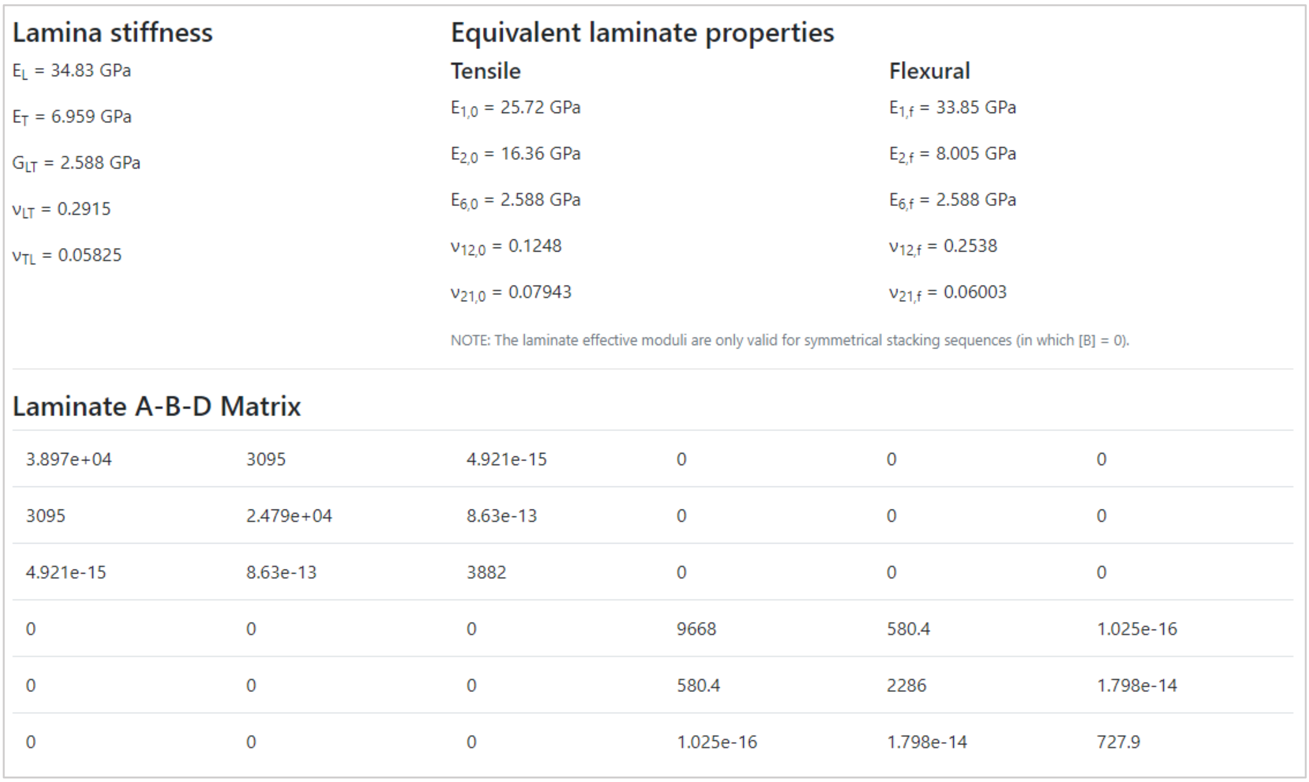
[✓ No symmetry] Symmetrical (Even) Symmetrical (Odd)

Stack orientation sequence considered: (empty)

Figure 19 – Laminate stacking options for advanced mode.

4.3.3 Laminate stiffness results

After choosing the appropriate mode of analysis and filling in all the required input fields, the user obtains the effective stiffness properties of the laminae and the laminate (Figure 20). On the top left the lamina elastic moduli and Poisson's ratios are shown. On the top right the user can obtain the effective tensile and flexural elastic properties of the laminate. Below these, the laminate ABD matrix is given.



Lamina stiffness

E_L = 34.83 GPa

E_T = 6.959 GPa

G_{LT} = 2.588 GPa

ν_{LT} = 0.2915

ν_{TL} = 0.05825

Equivalent laminate properties

Tensile	Flexural
$E_{1,0}$ = 25.72 GPa	$E_{1,f}$ = 33.85 GPa
$E_{2,0}$ = 16.36 GPa	$E_{2,f}$ = 8.005 GPa
$E_{6,0}$ = 2.588 GPa	$E_{6,f}$ = 2.588 GPa
$\nu_{12,0}$ = 0.1248	$\nu_{12,f}$ = 0.2538
$\nu_{21,0}$ = 0.07943	$\nu_{21,f}$ = 0.06003

NOTE: The laminate effective moduli are only valid for symmetrical stacking sequences (in which [B] = 0).

Laminate A-B-D Matrix

3.897e+04	3095	4.921e-15	0	0	0
3095	2.479e+04	8.63e-13	0	0	0
4.921e-15	8.63e-13	3882	0	0	0
0	0	0	9668	580.4	1.025e-16
0	0	0	580.4	2286	1.798e-14
0	0	0	1.025e-16	1.798e-14	727.9

Figure 20 – Example of lamina and laminate stiffness results.

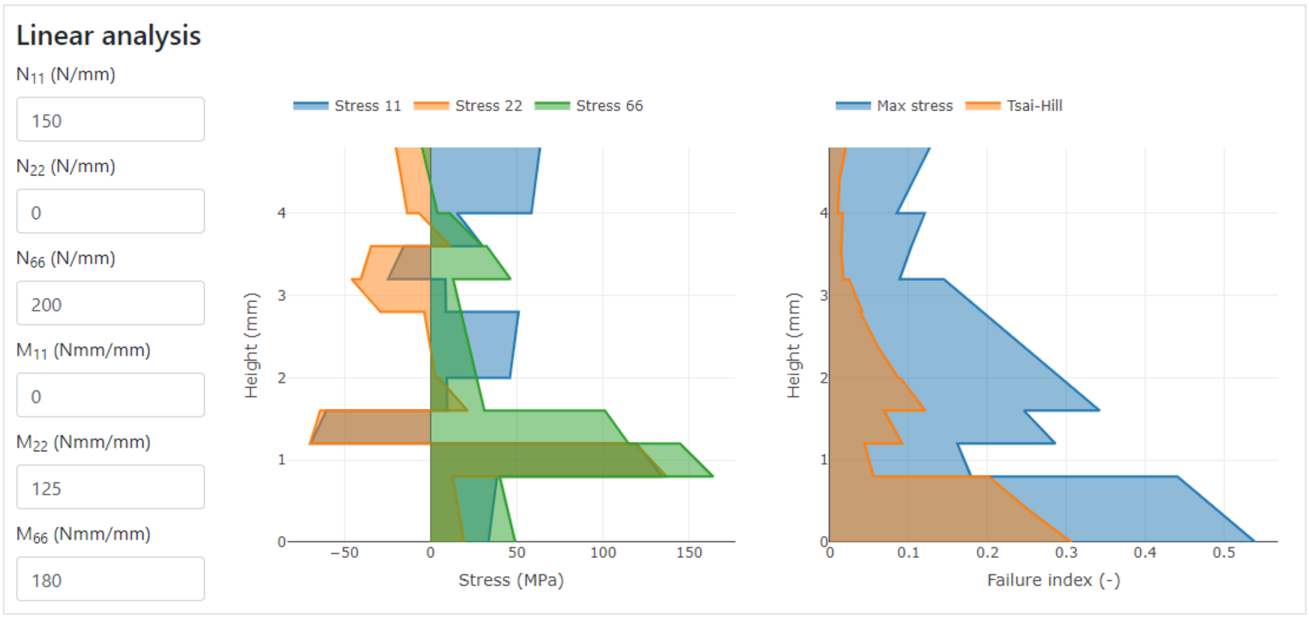
4.4 Analysis

The Analysis tab can only provide results if the previous tabs (Materials and Laminate) are duly filled.

4.4.1 Linear analysis

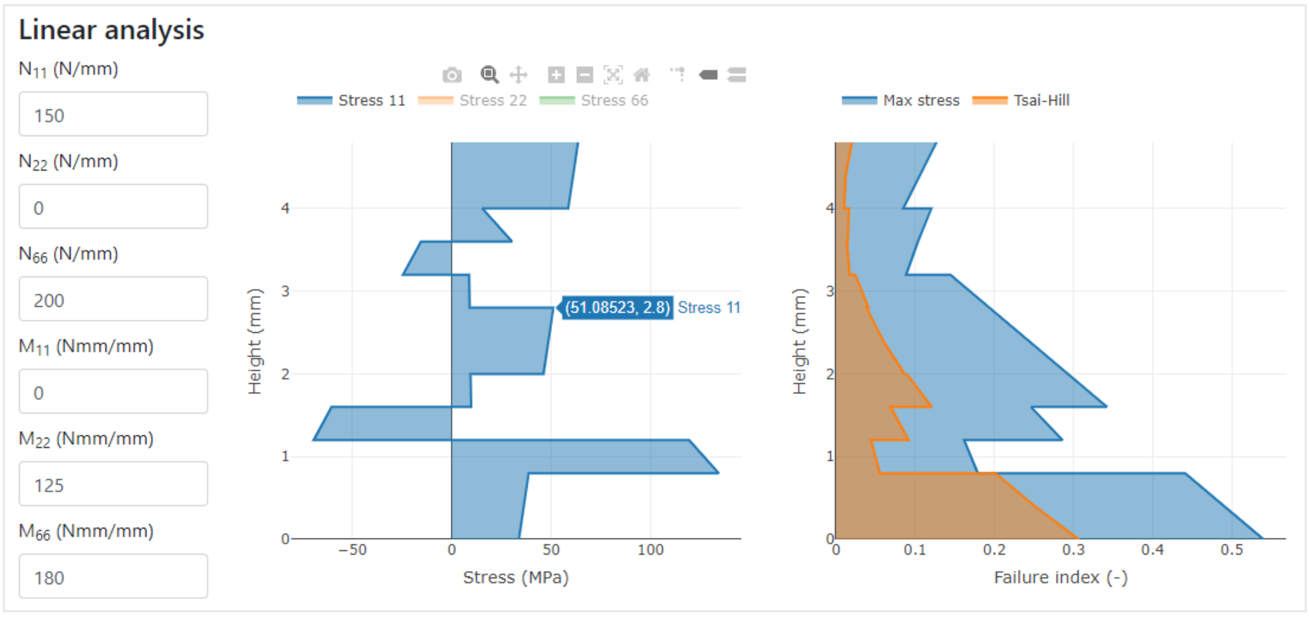
Firstly, a linear analysis panel is shown (Figure 21). The user can input the six different force values per unit width (N_{11}, N_{22}, N_{66}, M_{11}, M_{22} and M_{66}) and obtain the corresponding stresses across the thickness of the laminate (S_{11}, S_{22} and S_{66}), as well as the failure indices (Maximum stress and Tsai-Hill Index).

Upon hovering over the points on the charts the user is presented with the value of the stress for that given height (i.e., position along the thickness). The stress and index values are computed on the top and bottom of each lamina. Also, upon clicking on the legend, the user can toggle on or off one or the other curves for a clearer exhibition of the results (Figure 22). Also using the tool bar above the charts, the user may zoom in and out, pan and export an image file (in PNG format).


Linear analysis

N_{11} (N/mm)
150
N_{22} (N/mm)
0
N_{66} (N/mm)
200
M_{11} (Nmm/mm)
0
M_{22} (Nmm/mm)
125
M_{66} (Nmm/mm)
180

Figure 21 – Linear analysis input and results.

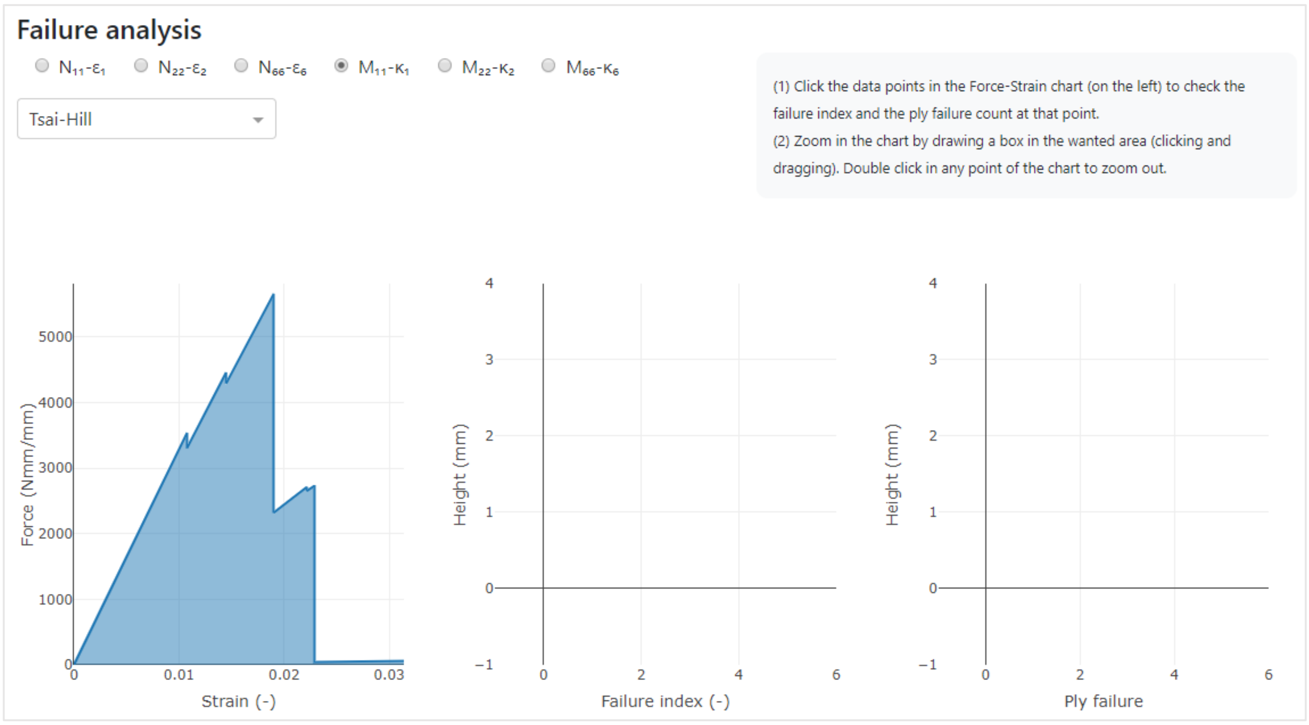

Linear analysis

N_{11} (N/mm)
150
N_{22} (N/mm)
0
N_{66} (N/mm)
200
M_{11} (Nmm/mm)
0
M_{22} (Nmm/mm)
125
M_{66} (Nmm/mm)
180

Figure 22 – Analysis chart features (toggling curves and hovering over data).

4.4.2 Failure analysis

Moving onto the bottom of the page, the user can obtain the progressive failure analyses results for each force-strain or moment-curvature pair (N_{11}-ε_1, N_{22}-ε_2, N_{66}-ε_6, M_{11}-κ_1, M_{22}-κ_2 and M_{66}-κ_6). For instance, Figure 23 shows the M_{11}-κ_1 moment-curvature path up to failure considering the Tsai-Hill failure index. As described in section 2.5, each peak point is computed by determining the maximum force of the laminate for that given stack configuration. Upon reaching the peak level there is a sudden drop which is related to the fact that some number of laminae have reached their respective failure indices and thus are discounted from the analysis. This evolution can be seen upon clicking the points on the force-strain path chart and analysing the two charts on the right. The middle chart computes the considered failure index (in this case the Tsai-Hill index) across the section thickness for all laminae that are still being considered in the analyses. The chart on the right shows the stacking of the laminate highlighting in green the laminae that are still being considered in the analysis and in red the laminae that have already reached failure and thus are discounted from the analysis (Figure 24).



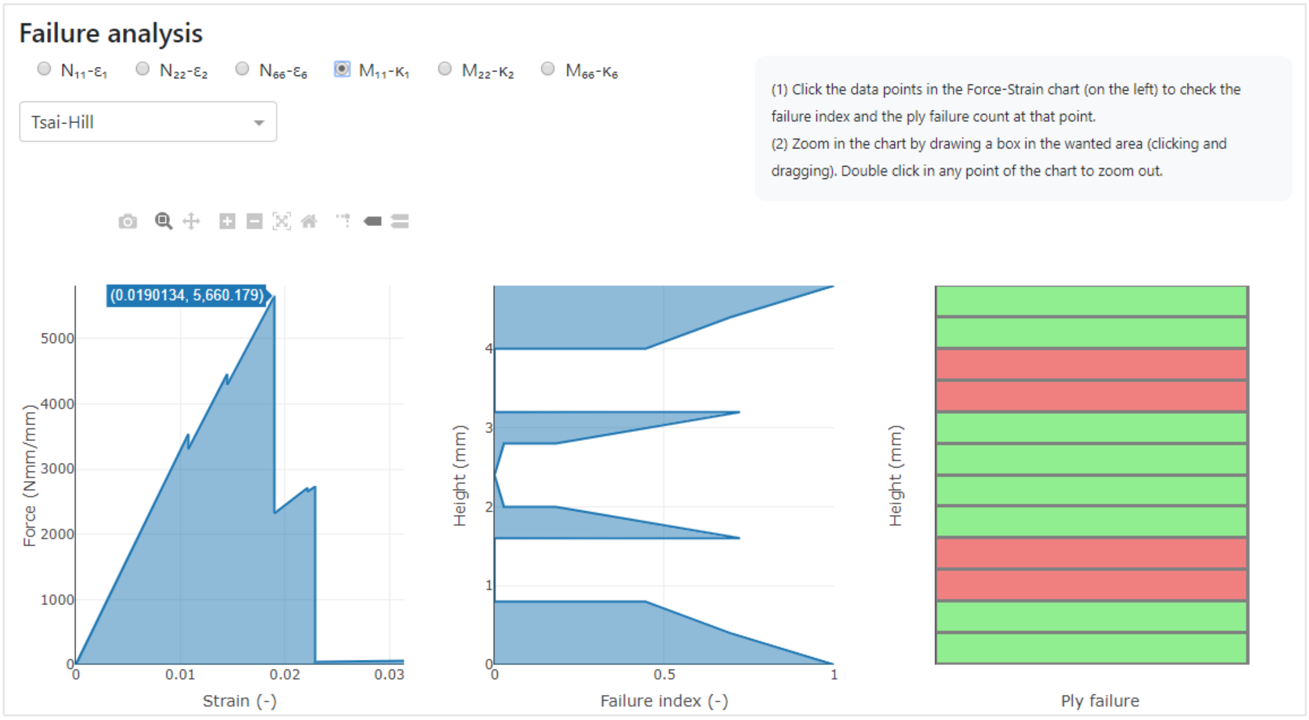
Failure analysis

N_{11}-ε_1 N_{22}-ε_2 N_{66}-ε_6 [✓ M_{11}-κ_1] M_{22}-κ_2 M_{66}-κ_6

Tsai-Hill

(1) Click the data points in the Force-Strain chart (on the left) to check the failure index and the ply failure count at that point.
(2) Zoom in the chart by drawing a box in the wanted area (clicking and dragging). Double click in any point of the chart to zoom out.

Figure 23 – Force-strain progressive failure path chart (in this illustrative example M_{11}-κ_1).



Failure analysis

N_{11}-ε_1 N_{22}-ε_2 N_{66}-ε_6 [✓ M_{11}-κ_1] M_{22}-κ_2 M_{66}-κ_6

Tsai-Hill

(1) Click the data points in the Force-Strain chart (on the left) to check the failure index and the ply failure count at that point.
(2) Zoom in the chart by drawing a box in the wanted area (clicking and dragging). Double click in any point of the chart to zoom out.

Figure 24 – Force-strain progressive failure path chart, failure index chart and lamina discount chart (in this illustrative example M_{11}-κ_1).

5 CONCLUSIONS

5.1 Main conclusions

This report described the theoretical background, the app development process and the user guide for the web-app LAYUP | FRP Laminate Analysis. This work constituted an application of new web-based technologies to the technical and scientific work in the field of FRP composites, consisting in the development of a user-friendly tool and interface that allows users to perform straightforward and easy designs and analyses of FRP layups. The fact that it is web-based allows for users to share their databases making these data accessible to everyone.

Within the EcoComposite project, this tool will help in the assessment of fibre-matrix compatibility, assessment of the applicability of traditional FRP theoretical and analytical formulations to the bio-based composites, and the development of layup optimization studies for different civil engineering applications of FRP.

5.2 Future developments

Despite the fact the LAYUP web-app is already online and ready to be used, there are some improvements that should be addressed in future updates. These improvements are listed below.

1. User login credentials – the objective would be to have each user with unique login credentials and different permissions in the app. Right now it is available for anyone with the same credentials, but this fact could compromise the goals of producing a high standard database of raw materials and laminates. Currently any user can add and delete materials (which might introduce irrelevant elements on the database or delete important ones).
2. Save laminate object – similarly to the fibre and resin database tables, the aim is to create a laminate database table with all the laminate inputs in order to use it in future analyses.
3. Property overwrites for bio-based resins – with the ongoing EcoComposite project one might conclude that the CLT is not appropriate for bio-based polymers and one might need to adapt those equations to take into account the fact that the resin is bio-based.
4. Migration to the IST servers – currently the web-app is hosted on a free-tier of a third-party server provider. Despite being able to be used without restrictions, the web-app takes a considerable amount of time to be loaded and also its computational resources are limited. Thus, it would be useful to migrate it to the IST servers since the whole project is being developed at IST and these constraints would be easily overcome.
5. Enabling exports – for some users, the results provided by LAYUP might not be enough and they may need some sort of post-processing. Thus, enabling exports as *.csv*, *.xls* or *.pdf* files might be useful.

6. Enabling the use of hybrid composites – hybrid laminates comprising different types of fibres (for instance carbon and glass) can be an interesting solution for some applications. Although this approach is already possible in the back-end (through the addition of laminae with different materials or thicknesses), it is not yet implemented in the front-end of the web-app.
7. Adding other data – cost, carbon footprint and weight of the laminate are some of the data that might be useful for the user and for future optimization studies and should be taken as an input for the materials and output for the resulting laminates.

6 REFERENCES

1. Bank, L.C. (2006). Composites for Construction (Hoboken, NJ, USA: John Wiley & Sons, Inc.) Available at: http://doi.wiley.com/10.1002/9780470121429.

2. Correia, J.R. (2008). GFRP pultruded profiles in civil engineering: hybrid solutions, bonded connections and fire behaviour. PhD Thesis in Civil Engineering. University of Lisbon.

3. Hewitt, R.L., and de Malherbe, M.C. (1970). An Approximation for the Longitudinal Shear Modulus of Continuous Fibre Composites. J. Compos. Mater. *4*, 280–282.

4. Loos, M. (2015). Chapter 5 - Fundamentals of Polymer Matrix Composites Containing CNTs. In Carbon Nanotube Reinforced Composites, M. Loos, ed. (Oxford: William Andrew Publishing), pp. 125–170. Available at: http://www.sciencedirect.com/science/article/pii/B9781455731954000059.

5. Jones, R.M. (1998). Mechanics of Composite Materials. Second Edi. (Philadelphia: Taylor & Francis).

6. Hashin, Z. (1980). Failure Criteria for Unidirectional Fiber Composites. J. Appl. Mech. *47*, 329–334. Available at: http://dx.doi.org/10.1115/1.3153664.

7. Hashin, Z., and Rotem, A. (1973). A Fatigue Failure Criterion for Fiber Reinforced Materials. J. Compos. Mater. *7*, 448–464. Available at: http://jcm.sagepub.com/content/7/4/448.abstract.

8. Python Software Foundation (2019). Python Language Reference, version 3.7. Available at: http://www.python.org.

9. Oliphant, T.E. (2006). A guide to NumPy. USA Trelgol Publ. Available at: https://docs.scipy.org/doc/numpy/index.html [Accessed October 21, 2019].

10. McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proc. 9th Python Sci. Conf., 51–56. Available at: https://pandas.pydata.org/pandas-docs/stable/.

11. Custódio, A.L., Madeira, J.F.A., Vaz, A.I.F., and Vicente, L.N. (2011). Direct Multisearch for Multiobjective Optimization. SIAM J. Optim. *21*, 1109–1140. Available at: http://epubs.siam.org/doi/abs/10.1137/10079731X.

12. Martín Abadi, Ashish Agarwal, Paul Barham, E.B., Zhifeng Chen, Craig Citro, Greg S. Corrado, A.D., Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, I.G., Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Y.J., Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, M.S., Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, J.S., Benoit Steiner, Ilya Sutskever, Kunal Talwar, P.T., Vincent Vanhoucke, Vijay Vasudevan, F.V., Oriol Vinyals, Pete Warden, Martin Wattenberg, M.W., and Yuan Yu, and X.Z. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Available at: tensorflow.org [Accessed October 30, 2019].

13. James, G., Witten, D., Hastie, T., and Tibshirani, R. (2017). Introduction to Statistical Learning with Applications in R (Springer).

14. Plotly Technologies Inc. (2015). Collaborative data science. Available at: https://plot.ly [Accessed October 30, 2019].

APPENDIX – BACK-END SCRIPT

```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import cos, sin, radians, degrees, sqrt
from numpy.linalg import inv
float_formatter = lambda x: "%.2f" % x
np.set_printoptions(formatter={float_kind:float_formatter})
import dash_core_components as dcc
import dash_html_components as html
from copy import deepcopy

"""
These analyses are based on formulations proposed by LC Bank on Composites for
Construction.

Changelog:
    - the value of the major poisson is considered to be 12 (or LT) and not 21
(or TL);
    - the third term of the transformation matrix is wrong: it now reads c**2-
s**2

"""

# defining the resin object
class Resin:

    """Creates a new Resin object"""

    def __init__(self, E, poisson, strength, density=1200):
        # density in kg/m**3
        self.density = density
        # Elastic modulus in GPa
        self.E = E
        # poisson ratio
        self.poisson = poisson
        # shear modulus in GPa
        self.G = self.E/(2*(1+self.poisson))
        # strength
        self.strength = strength
        # shear strength
        self.tau = self.strength/sqrt(3)
        # resin volume
        self.volume = None
        # resin name
        self.name = None

    def set_name(self, name):
        self.name = name
```

```
# defining the fibre object
class Fibre:

    """Creates a new Fibre object"""

    def __init__(self, E, poisson, strength, density=2500):
        # density in kg/m**3
        self.density = density
        # elastic modulus in GPa
        self.E = E
        # poisson ratio
        self.poisson = poisson
        # shear modulus in GPa
        self.G = self.E/(2*(1+self.poisson))
        # strength
        self.strength = strength
        # shear strength
        self.tau = self.strength/sqrt(3)
        # fibre volume
        self.volume = None
        # fibre name
        self.name = None
        # boolean variable indicating if it is fabric
        self.is_fabric = False
        # set the balance between the longitudinal and transverse direction
        self.balance = 1

    def set_name(self, name):
        self.name = name

# defining the fabric object
class Fabric(Fibre):

    """Creates a new Fabric object"""

    def __init__(self, fibre, area_weight_L, area_weight_T):
        if area_weight_L == 0:
            print(ERROR. The area weight in the longitudinal direction must not
be null. \
                  The fibre direction can be rotated upon assigning a stack
orientation to the laminate object)
        else:
            Fibre.__init__(self, fibre.E, fibre.poisson, fibre.strength,
fibre.density)
            # area weight in the longitudinal direction
            self.area_weight_L = area_weight_L
            # area weight in the transverse direction
            self.area_weight_T = area_weight_T
            # total area weight
            self.area_weight = area_weight_L + area_weight_T
            # assign boolean value speficying that it is a fabric
            self.is_fabric = True
            self.balance = self.area_weight_L/(self.area_weight)
            # assigning a fibre object
            self.fibre = fibre

# defining the lamina object
class Lamina:
    """Creates a new Lamina object"""

    def __init__(self, resin, fibre, volume, theta, halpin_tsai=False):
        # assigning the type of resin
        self.resin = resin
        # assigning the type of fibre
        self.fibre = fibre
        # defining the fibre volume (0-1)
        self.volume = volume
        fibre.volume = volume
        resin.volume = 1-volume
        # lamina orientation in radians
        theta = radians(theta)
        self.theta = theta
        # setting the halpin_tsai boolean variable
        self.halpin_tsai = halpin_tsai
        # assigning z_bot (height at the bottom) and z_top (height at the top)
        # variables (will only be filled in the laminate object)
        self.z_bot = None
        self.z_top = None
        # checking whether the fibres are unidirectional or bidirectional
        if fibre.balance == 1:
            self.is_bidirectional = False
            self.is_unidirectional = True
            # compute the elastic moduli and matrices
            self.compute_moduli()
            self.compute_matrices()
        else:
            self.is_bidirectional = True
            self.is_unidirectional = False
            self.set_bidirectional(fibre.balance)
        if self.fibre.is_fabric:
            # compute te automatic thickness according to the CLT formulation
            self.thickness = self.fibre.area_weight*10**-
3/(self.fibre.density*self.volume)
        else:
            self.thickness = None
        self.maximum_stress_index = None
        self.tsai_hill_index = None
        self.ply_failure = False

    def __str__(self):
        return .join([Lamina(resin,fibre,,str(self.volume),,,str(self.theta),)])

    def compute_moduli(self):
        # self properties
        resin = self.resin
        fibre = self.fibre
        volume = self.volume
        # longitudinal elastic modulus
        self.EL = resin.E*resin.volume + fibre.E*fibre.volume
        # longitudinal strength
        self.strengthL = resin.strength*resin.volume +
fibre.strength*fibre.volume
        # transverse elastic modulus
        self.ET = 1/(resin.volume/resin.E + fibre.volume/fibre.E)
        # transverse strength
        self.strengthT = 1/(resin.volume/resin.strength +
fibre.volume/fibre.strength)
```

```
        # poisson LT
        self.poissonLT = resin.poisson*resin.volume + fibre.poisson*fibre.volume
        # poisson TL
        self.poissonTL = self.ET/self.EL*self.poissonLT
        # shear modulus of the layer
        self.GLT = 1/(resin.volume/resin.G + fibre.volume/fibre.G)
        # shear strength
        self.strengthLT = 1/(resin.volume/resin.tau + fibre.volume/fibre.tau)
        # compute the transverse and shear modulus according to the Halpin-Tsai
equations
        if self.halpin_tsai:
            if volume < 0.65:
                # xi parameter according to the original halpin tsai formulation
                self.xi_ET = 2
                self.xi_GLT = 1
            else:
                # xi parameter according Hewitt and de Malherbe variation
                self.xi_ET = 2 + 40*volume**10
                self.xi_GLT = 1 + 40*volume**10
            self.eta_ET = ((fibre.E/resin.E)-1)/((fibre.E/resin.E)+self.xi_ET)
            self.ET = (1+self.xi_ET*self.eta_ET*volume)/(1-
self.eta_ET*volume)*resin.E
            self.eta_GLT = ((fibre.G/resin.G)-1)/((fibre.G/resin.G)+self.xi_GLT)
            self.GLT = (1+self.xi_GLT*self.eta_GLT*volume)/(1-
self.eta_GLT*volume)*resin.G

    def compute_matrices(self):
        # self properties
        theta = self.theta
        # lamina stiffness matrix
        self.QLT = np.array(
                [self.EL/(1-self.poissonLT*self.poissonTL),
                 self.EL*self.poissonTL/(1-self.poissonLT*self.poissonTL),
                 0,
                 self.ET*self.poissonLT/(1-self.poissonLT*self.poissonTL),
                 self.ET/(1-self.poissonLT*self.poissonTL),
                 0,
                 0,
                 0,
                 self.GLT]).reshape(3,3)*1000
        # lamina transformation matrix
        self.transform_matrix = np.array(
                [cos(theta)**4,
                 sin(theta)**4,
                 2*cos(theta)**2*sin(theta)**2,
                 4*cos(theta)**2*sin(theta)**2,
                 sin(theta)**4,
                 cos(theta)**4,
                 2*cos(theta)**2*sin(theta)**2,
                 4*cos(theta)**2*sin(theta)**2,
                 cos(theta)**2*sin(theta)**2,
                 cos(theta)**2*sin(theta)**2,
                 cos(theta)**4+sin(theta)**4,
                 -4*cos(theta)**2*sin(theta)**2,
                 cos(theta)**2*sin(theta)**2,
                 cos(theta)**2*sin(theta)**2,
                 -2*cos(theta)**2*sin(theta)**2,
                 (cos(theta)**2-sin(theta)**2)**2,
                 cos(theta)**3*sin(theta),
                  -cos(theta)*sin(theta)**3,
                   cos(theta)*sin(theta)**3-cos(theta)**3*sin(theta),
                   2*(cos(theta)*sin(theta)**3-cos(theta)**3*sin(theta)),
                   cos(theta)*sin(theta)**3,
                   -cos(theta)**3*sin(theta),
                   cos(theta)**3*sin(theta)-cos(theta)*sin(theta)**3,
                   2*(cos(theta)**3*sin(theta)-cos(theta)*sin(theta)**3),
                   ]).reshape(6,4)
        # lamina fibre alignment orientation properties
        self.Q_0 = np.array(
                [self.QLT[0,0],
                 self.QLT[1,1],
                 self.QLT[0,1],
                 self.QLT[2,2]])
        # lamina theta degree orientation properties
        self.Q_oriented = np.matmul(self.transform_matrix, self.Q_0)
        # lamina theta degree orientation properties
        self.Q_oriented_square = np.array(
                [self.Q_oriented[0],
                 self.Q_oriented[2],
                 self.Q_oriented[4],
                 self.Q_oriented[2],
                 self.Q_oriented[1],
                 self.Q_oriented[5],
                 self.Q_oriented[4],
                 self.Q_oriented[5],
                 self.Q_oriented[3]]).reshape(3,3)
        self.S = inv(self.Q_oriented_square)
        # compute the theta oriented elastic properties
        self.E1 = 1/self.S[0,0]/1000
        self.E2 = 1/self.S[1,1]/1000
        self.E6 = 1/self.S[2,2]/1000
        self.poisson12 = -self.S[1,0]/self.S[0,0]
        self.poisson21 = -self.S[0,1]/self.S[1,1]

    def set_bidirectional(self, balance):
        # assign the fraction of fibres that are aligned in the longitudinal
direction of the lamina
        self.balance = balance
        lamina_0 = Lamina(self.resin, self.fibre.fibre, self.volume, 0,
halpin_tsai=self.halpin_tsai)
        lamina_0.compute_moduli()
        lamina_0.compute_matrices()
        lamina_90 = Lamina(self.resin, self.fibre.fibre, self.volume, 90,
halpin_tsai=self.halpin_tsai)
        lamina_90.compute_moduli()
        lamina_0.compute_matrices()
        # longitudinal elastic modulus
        self.EL = lamina_0.E1*balance + lamina_90.E1*(1-balance)
        # longitudinal strength
        self.strengthL = lamina_0.strengthL*balance + lamina_90.strengthT*(1-
balance)
        # transverse elastic modulus
        self.ET = lamina_0.E2*balance + lamina_90.E2*(1-balance)
        # transverse strength
        self.strengthT = lamina_0.strengthT*balance + lamina_90.strengthL*(1-
balance)
        # poisson LT
        self.poissonLT = lamina_0.poisson12*balance + lamina_90.poisson12*(1-
balance)
        # poisson TL
        self.poissonTL = self.ET/self.EL*self.poissonLT
        # shear modulus of the lamina
        self.GLT = lamina_0.E6*balance + lamina_90.E6*(1-balance)
        # shear strength
        self.strengthLT = lamina_0.strengthLT*balance + lamina_90.strengthLT*(1-
balance)
        # compute lamina matrices
        self.compute_matrices()

# defining the laminate object
class Laminate:

    """Creates a new Laminate object"""

    def __init__(self):
        # list of all laminae in the laminate
        self.laminae = []
        # list of thicknesses for each lamina
        self.thicknesses = []
        # list of centroids for each lamina
        self.centroids = []
        # number of laminae in the laminate
        self.num_laminae = len(self.laminae)
        # total thickness
        self.total_thickness = sum(self.thicknesses)

    # add a single lamina to the laminate
    def add_lamina(self, lamina, thickness=None, compute_properties=True):

        """Add a single lamina to the laminate"""
        # flag an error to check if the thickness is specified or not
        error = False
        # check whether the fibre material is fabric or not
        if lamina.fibre.is_fabric:
            # compute te automatic thickness according to the CLT formulation
            thickness_auto = lamina.fibre.area_weight*10**-
3/(lamina.fibre.density*lamina.volume)
            # check if the thickness was provided and overwrite it
            if thickness != None:
                print(WARNING. For laminae with the fabric object, the thickness
is automatically computed \
                      according to the CLT formulation. The value of {} was
overwritten to {}.format(thickness, thickness_auto))
                thickness = thickness_auto
        # if the fibre object is not fabric the thickness must be provided
        else:
            # if no thickness is provided return an error and exit the analysis
            if thickness == None:
                print(ERROR. For laminae without the fabric object, the thickness
should be specified.)
                error = True
        # exit the analysis
        if error:
            pass
        # continue the analysis
        else:
            # add the lamina to the laminae list
            self.laminae.append(lamina)
            lamina.z_bot = self.total_thickness
            lamina.z_top = lamina.z_bot + thickness
            # add the thickness to the thicknesses list
            lamina.thickness = thickness
            self.thicknesses.append(thickness)
            # add the centroid to the centroids list
            centroid = self.total_thickness + thickness/2
            self.centroids.append(centroid)
            # number of laminae in the laminate
            self.num_laminae = len(self.laminae)
            # total thickness
            self.total_thickness = sum(self.thicknesses)
            if compute_properties:
                # compute ABD stiffness matrix
                self.compute_ABD_matrix()
                self.compute_effective_moduli()

    # add multiple laminae to the laminate
    def add_multiple_laminae(self, resin, fibre, volume, thickness_list,
angle_list, halpin_tsai=False):

        """Add multiple laminae to the laminate"""

        # check the length of each list
        if len(thickness_list)==len(angle_list):
            # create all the laminae
            for i in range(len(thickness_list)):
                # assign the lamina properties
                lamina = Lamina(resin, fibre, volume, angle_list[i],
halpin_tsai=halpin_tsai)
                self.add_lamina(lamina, thickness=thickness_list[i],
compute_properties=False)
            # number of laminae in the laminate
            self.num_laminae = len(self.laminae)
            # compute de ABD stiffness matrix
            self.compute_ABD_matrix()
            self.compute_effective_moduli()
        else:
            print(ERROR. Lists must be of the same length)

    def compute_ABD_matrix(self):

        """Computes the laminate A-B-D matrix"""

        #laminae properities data frame
        self.df = pd.DataFrame()
        if self.num_laminae > 0:
            # definition of matrix Q
            self.df[Q11] = [lamina.Q_oriented[0] for lamina in self.laminae]
            self.df[Q22] = [lamina.Q_oriented[1] for lamina in self.laminae]
            self.df[Q12] = [lamina.Q_oriented[2] for lamina in self.laminae]
            self.df[Q66] = [lamina.Q_oriented[3] for lamina in self.laminae]
            self.df[Q16] = [lamina.Q_oriented[4] for lamina in self.laminae]
            self.df[Q26] = [lamina.Q_oriented[5] for lamina in self.laminae]
            # thickness for each lamina
            self.df[thickness] = [thickness for thickness in self.thicknesses]
            # centroid for each lamina
            self.df[zk] = [-centroid+self.total_thickness/2 for centroid in 
self.centroids]
            # definition of matrix A
            self.df[A11_i] = self.df[Q11]*self.df[thickness]
            self.df[A22_i] = self.df[Q22]*self.df[thickness]
            self.df[A12_i] = self.df[Q12]*self.df[thickness]
            self.df[A66_i] = self.df[Q66]*self.df[thickness]
            self.df[A16_i] = self.df[Q16]*self.df[thickness]
            self.df[A26_i] = self.df[Q26]*self.df[thickness]
            self.A = np.zeros((3,3))
            self.A[0,0] = self.df[A11_i].sum()*1000
            self.A[1,1] = self.df[A22_i].sum()*1000
            self.A[0,1] = self.A[1,0] = self.df[A12_i].sum()*1000
            self.A[2,2] = self.df[A66_i].sum()*1000
            self.A[0,2] = self.A[2,0] = self.df[A16_i].sum()*1000
            self.A[1,2] = self.A[2,1] = self.df[A26_i].sum()*1000
            # definition of matrix B
            self.df[B11_i] = self.df[Q11]*(-self.df[zk]*self.df[thickness])
            self.df[B22_i] = self.df[Q22]*(-self.df[zk]*self.df[thickness])
            self.df[B12_i] = self.df[Q12]*(-self.df[zk]*self.df[thickness])
            self.df[B66_i] = self.df[Q66]*(-self.df[zk]*self.df[thickness])
            self.df[B16_i] = self.df[Q16]*(-self.df[zk]*self.df[thickness])
            self.df[B26_i] = self.df[Q26]*(-self.df[zk]*self.df[thickness])
            self.B = np.zeros((3,3))
            self.B[0,0] = self.df[B11_i].sum()*1000
            self.B[1,1] = self.df[B22_i].sum()*1000
            self.B[0,1] = self.B[1,0] = self.df[B12_i].sum()*1000
            self.B[2,2] = self.df[B66_i].sum()*1000
            self.B[0,2] = self.B[2,0] = self.df[B16_i].sum()*1000
            self.B[1,2] = self.B[2,1] = self.df[B26_i].sum()*1000
            # definition of matrix D
            self.df[D11_i] = 
self.df[Q11]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.df[D22_i] = 
self.df[Q22]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.df[D12_i] = 
self.df[Q12]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.df[D66_i] = 
self.df[Q66]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.df[D16_i] = 
self.df[Q16]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.df[D26_i] = 
self.df[Q26]*(self.df[thickness]*self.df[zk]**2+(self.df[thickness]**3)/12)
            self.D = np.zeros((3,3))
            self.D[0,0] = self.df[D11_i].sum()*1000**3
            self.D[1,1] = self.df[D22_i].sum()*1000**3
            self.D[0,1] = self.D[1,0] = self.df[D12_i].sum()*1000**3
            self.D[2,2] = self.df[D66_i].sum()*1000**3
            self.D[0,2] = self.D[2,0] = self.df[D16_i].sum()*1000**3
            self.D[1,2] = self.D[2,1] = self.df[D26_i].sum()*1000**3
            # ABD matrix definition
            AB = np.concatenate((self.A, self.B), axis = 1)
            BD = np.concatenate((self.B, self.D), axis = 1)
            self.ABD = np.concatenate((AB,BD), axis = 0)

    def compute_effective_moduli(self):
        self.a = inv(self.A)
        self.d = inv(self.D)
        # axial effective moduli
        self.E1_0 = (self.a[0,0]*self.total_thickness*1000)**(-1)
        self.E2_0 = (self.a[1,1]*self.total_thickness*1000)**(-1)
        self.E6_0 = (self.a[2,2]*self.total_thickness*1000)**(-1)
        self.poisson21_0 = -self.a[0,1]/self.a[1,1]
        self.poisson12_0 = -self.a[1,0]/self.a[0,0]
        # flexural effective moduli
        self.E1_f = 12/(self.d[0,0]*(self.total_thickness*1000)**3)
        self.E2_f = 12/(self.d[1,1]*(self.total_thickness*1000)**3)
        self.E6_f = 12/(self.d[2,2]*(self.total_thickness*1000)**3)
        self.poisson21_f = -self.d[0,1]/self.d[1,1]
        self.poisson12_f = -self.d[1,0]/self.d[0,0]

    def ABD_as_dataframe (self):

        """Converts a numpy array matrix into a pandas Data Frame"""

        df = pd.DataFrame(data=self.ABD)
        return df

    def compute_stress_state(self, N1, N2, N6, M1, M2, M6):

        """Computes a stress-strain state for given loads"""

        # instanciate a pandas data frame with the analysis
        df = pd.DataFrame(columns=[z (m), Angle (º),
                                   Strain 11 (m/m), Strain 22 (m/m), Strain 66
(m/m),
                                   Stress 11 (MPa), Stress 22 (MPa), Stress 66
(MPa),
                                   Stress L (MPa), Stress T (MPa), Stress LT
(MPa),
                                   Maximum stress, Tsai-Hill])
        # compute the inverse of matrix ABD
        ABD_inv = inv(self.ABD)
        # define the array of forces per unit width
        force_array = np.array([N1, N2, N6, M1, M2, M6])
        # compute the strain array
        strains = [e1, e2, e6, k1, k2, k6] = np.matmul(ABD_inv, force_array)
        # instanciate lists of strains, stresses and heights
        strain_list = []
        stress_list = []
        z_list = []
        # instanciate a counter for adding rows to the pandas data frame
        counter = 0
        # compute the strains and stresses in each lamina of the laminate
        for lamina in self.laminae:
            # transformation matrix from stress aligned along the laminate to
stresses aligned along the lamina
            transform_matrix = np.array([
                    cos(lamina.theta)**2,
                    sin(lamina.theta)**2,
                    2*cos(lamina.theta)*sin(lamina.theta),
                    sin(lamina.theta)**2,
                    cos(lamina.theta)**2,
                    -2*cos(lamina.theta)*sin(lamina.theta),
                    -cos(lamina.theta)*sin(lamina.theta),
                    cos(lamina.theta)*sin(lamina.theta),
                    cos(lamina.theta)**2-sin(lamina.theta)**2
                    ]).reshape(3,3)
            # bottom of the layer
            strain_bot_1 = e1 - k1*(lamina.z_bot-self.total_thickness/2)*1000
            strain_bot_2 = e2 - k2*(lamina.z_bot-self.total_thickness/2)*1000
            strain_bot_6 = e6 - k6*(lamina.z_bot-self.total_thickness/2)*1000
            # determination of the stresses for the bottom face
            [stress_bot_1, stress_bot_2, stress_bot_6] = \

np.matmul(lamina.Q_oriented_square,[strain_bot_1,strain_bot_2,strain_bot_6])
            # compute the lamina oriented stresses in the top face of the lamina
            [stress_bot_L, stress_bot_T, stress_bot_LT] = np.matmul(
                    transform_matrix, [stress_bot_1, stress_bot_2, stress_bot_6])
            # top of the lamina
            strain_top_1 = e1 - k1*(lamina.z_top-self.total_thickness/2)*1000
            strain_top_2 = e2 - k2*(lamina.z_top-self.total_thickness/2)*1000
            strain_top_6 = e6 - k6*(lamina.z_top-self.total_thickness/2)*1000
            # determination of the stresses for the top face
            [stress_top_1, stress_top_2, stress_top_6] = \

np.matmul(lamina.Q_oriented_square,[strain_top_1,strain_top_2,strain_top_6])
            # compute the lamina oriented stresses in the top face of the lamina
            [stress_top_L, stress_top_T, stress_top_LT] = np.matmul(
                    transform_matrix, [stress_top_1, stress_top_2, stress_top_6])
            # compute the maximum stress index
            max_stress_top = max(
                    abs(stress_top_L)/lamina.strengthL,
                    abs(stress_top_T)/lamina.strengthT,
                    abs(stress_top_LT)/lamina.strengthLT)
            max_stress_bot = max(
                    abs(stress_bot_L)/lamina.strengthL,
                    abs(stress_bot_T)/lamina.strengthT,
                    abs(stress_bot_LT)/lamina.strengthLT)
            lamina.maximum_stress_index = max(max_stress_top, max_stress_bot)
            # compute the Tsai-Hill index
            tsai_hill_top = stress_top_L**2/lamina.strengthL**2-
stress_top_L*stress_top_T/lamina.strengthL**2+\

stress_top_T**2/lamina.strengthT**2+stress_top_LT**2/lamina.strengthLT**2
            tsai_hill_bot = stress_bot_L**2/lamina.strengthL**2-
stress_bot_L*stress_bot_T/lamina.strengthL**2+\

stress_bot_T**2/lamina.strengthT**2+stress_bot_LT**2/lamina.strengthLT**2
            lamina.tsai_hill_index = max(tsai_hill_top, tsai_hill_bot)
            # add the information to the pandas data frame
            df.loc[counter] = [lamina.z_bot, degrees(lamina.theta),
                  strain_bot_1, strain_bot_2, strain_bot_6,
                  stress_bot_1, stress_bot_2, stress_bot_6,
                  stress_bot_L, stress_bot_T, stress_bot_LT,
                  max_stress_bot, tsai_hill_bot]
            df.loc[counter+1] = [lamina.z_top, degrees(lamina.theta),
                  strain_top_1, strain_top_2, strain_top_6,
                  stress_top_1, stress_top_2, stress_top_6,
                  stress_top_L, stress_top_T, stress_top_LT,
                  max_stress_top, tsai_hill_top]
            counter += 2
        return df

    def compute_maximum_stress_index(self, df):
        return max(df[Maximum stress])
    def compute_tsai_hill_index(self, df):
        return max(df[Tsai-Hill])

    def compute_hashin_criterion(self, df):
        pass

    def get_max_force(self, index_target=1, force_type=N1, force_sign=plus,
index_type=Tsai-Hill, tolerance=1e-3):

        # instanciate the variables
        index = 0
        if force_sign == plus:
            force = 1
        elif force_sign == minus:
            force=-1
        else:
            print(ERROR. The force_sign variable should be either plus or minus)
        i=0
        # find solution to the problem
        while (index-index_target)**2 > tolerance:
            # assign the force array according to the variable force_type
            if force_type == N1:
                [N1, N2, N6, M1, M2, M6] = [force, 0, 0, 0, 0, 0]
            elif force_type == N2:
                [N1, N2, N6, M1, M2, M6] = [0, force, 0, 0, 0, 0]
            elif force_type == N6:
                [N1, N2, N6, M1, M2, M6] = [0, 0, force, 0, 0, 0]
            elif force_type == M1:
                [N1, N2, N6, M1, M2, M6] = [0, 0, 0, force, 0, 0]
            elif force_type == M2:
                [N1, N2, N6, M1, M2, M6] = [0, 0, 0, 0, force, 0]
            elif force_type == M6:
                [N1, N2, N6, M1, M2, M6] = [0, 0, 0, 0, 0, force]
            else:
                print(ERROR. The force_type should be either N1, N2, N6, M1, M2
or M6)

            # calculate the step results
            df = self.compute_stress_state(N1, N2, N6, M1, M2, M6)
            if index_type == Tsai-Hill:
                index = self.compute_tsai_hill_index(df)
                # assign the new force value
                force = force/sqrt(index)
            elif index_type == Maximum stress:
                index = self.compute_maximum_stress_index(df)
                # assign the new force value
                force = force/index
            i+=1
        return force, i, [N1, N2, N6, M1, M2, M6]

    def get_max_force_list_factor(self, force_list, index_target=1,
index_type=Tsai-Hill, tolerance=1e-3):

        # instanciate the variables
        index = 0
        i=0
        [N1, N2, N6, M1, M2, M6] = force_list
        force_factor=1
        # find solution to the problem
        while (index-index_target)**2 > tolerance:
            # calculate the step results
            df = self.compute_stress_state(N1, N2, N6, M1, M2, M6)
            if index_type == Tsai-Hill:
                index = self.compute_tsai_hill_index(df)
            elif index_type == Maximum stress:
                index = self.compute_maximum_stress_index(df)
            # assign the new force value
            factor = 1/sqrt(index)
            force_factor = force_factor*factor
            force_list = [N1, N2, N6, M1, M2, M6] = np.array(force_list)*factor
            i+=1
        return force_factor, force_list, i
def compute_load_strain_path(laminate, force_type='N1', index_type='Tsai-Hill'):
    # assing a temporary laminate object to perform all calculations
    temp_laminate = deepcopy(laminate)
    # instanciate a boolean failure list
    step_bool_failure_list = [int(lamina.ply_failure) for lamina in
temp_laminate.laminae]
    bool_failure_list = []
    bool_failure_list.append(step_bool_failure_list)
    # instanciate lists of path forces and strains
    force_path = [0]
    strain_path = [0]
    # instantiating the lists for the maximum stress and the tsai-hill index
    maximum_stress_list = [[0 for i in range(len(temp_laminate.laminae)*2)]]
    tsai_hill_index_list = [[0 for i in range(len(temp_laminate.laminae)*2)]]
    # while there are laminae without failure
    while 0 in step_bool_failure_list:
        # get the maximum force attained
        step_force_array = [N1, N2, N6, M1, M2, M6] =
temp_laminate.get_max_force(force_type=force_type, index_type=index_type)[2]
        step_bool_failure_list = [int(lamina.ply_failure) for lamina in
temp_laminate.laminae]
        bool_failure_list.append(step_bool_failure_list)
        # compute the inverse ABD matrix
        ABD_inv = inv(temp_laminate.ABD)
        # compute the step strain array
        step_strain_array = [e1, e2, e6, k1, k2, k6] = np.matmul(ABD_inv,
step_force_array)
        df = temp_laminate.compute_stress_state(N1, N2, N6, M1, M2, M6)
        maximum_stress_list.append(list(df['Maximum stress']))
        tsai_hill_index_list.append(list(df['Tsai-Hill']))
        # append the force and strain pair to the path lists according to the
variable force_type
        if force_type == 'N1':
            force_path.append(N1)
            strain_path.append(e1)
        elif force_type == 'N2':
            force_path.append(N2)
            strain_path.append(e2)
        elif force_type == 'N6':
            force_path.append(N6)
            strain_path.append(e6)
        elif force_type == 'M1':
            force_path.append(M1)
            strain_path.append(k1)
        elif force_type == 'M2':
            force_path.append(M2)
            strain_path.append(k2)
        elif force_type == 'M6':
            force_path.append(M6)
            strain_path.append(k6)
        else:
            print('ERROR. The force_type should be either N1, N2, N6, M1, M2 or
M6')
        # check if the index is already achieved or not
        if index_type == 'Tsai-Hill':
            tsai_hill = temp_laminate.compute_tsai_hill_index(df)
            for lamina in temp_laminate.laminae:
                if round(lamina.tsai_hill_index*1000)/1000 == 1:
                    lamina.Q_oriented = lamina.Q_oriented/1e6
                    lamina.Q_oriented_square = lamina.Q_oriented_square/1e6
                    lamina.ply_failure = True
        elif index_type == 'Maximum stress':
            max_stress = temp_laminate.compute_maximum_stress_index(df)
            for lamina in temp_laminate.laminae:
                if round(lamina.maximum_stress_index*1000)/1000 == 1:
                    lamina.Q_oriented = lamina.Q_oriented/1e6
                    lamina.Q_oriented_square = lamina.Q_oriented_square/1e6
                    lamina.ply_failure = True
        step_bool_failure_list = [int(lamina.ply_failure) for lamina in
temp_laminate.laminae]
        bool_failure_list.append(step_bool_failure_list)
        temp_laminate.compute_ABD_matrix()
        temp_laminate.compute_effective_moduli()
        step_force_array = [N1, N2, N6, M1, M2, M6] =
np.matmul(temp_laminate.ABD, step_strain_array)
        df = temp_laminate.compute_stress_state(N1, N2, N6, M1, M2, M6)
        maximum_stress_list.append(list(df['Maximum stress']))
        tsai_hill_index_list.append(list(df['Tsai-Hill']))
        # append the force and strain pair to the path lists according to the
variable force_type
        if force_type == 'N1':
            force_path.append(N1)
            strain_path.append(e1)
        elif force_type == 'N2':
            force_path.append(N2)
            strain_path.append(e2)
        elif force_type == 'N6':
            force_path.append(N6)
            strain_path.append(e6)
        elif force_type == 'M1':
            force_path.append(M1)
            strain_path.append(k1)
        elif force_type == 'M2':
            force_path.append(M2)
            strain_path.append(k2)
        elif force_type == 'M6':
            force_path.append(M6)
            strain_path.append(k6)
        else:
            print('ERROR. The force_type should be either N1, N2, N6, M1, M2 or
M6')
    laminate_height_list = list(df['z (m)']*1000)

    return force_path, strain_path, bool_failure_list, laminate_height_list,
maximum_stress_list, tsai_hill_index_list
```

```
def run_random_stack_sequence_test(resin, fibre, volume, test_number,
layer_thickness, layer_number, angles_available=[0, 30, 45, 60, 90, -30, -45, -
60],force_array):

    laminates = []
    stacks = []

    for i in range(test_number):
        laminate = Laminate()
        stack = []
        for i in range(layer_number):
            angle = int(np.random.randint(len(angles_available), size=1))
            stack.append(angles_available[angle])
        laminate.add_multiple_laminae(polyester, glass, volume, [layer_thickness
for i in range(layer_number)], stack)
        stacks.append(stack)
        laminates.append(laminate)

    forces=[]
    for laminate in laminates:
        force = laminate.get_max_force_list_factor(force_array)
        forces.append(force[0])
    argmax = forces.index(max(forces))
    return max(forces), argmax
```