



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Estudi computacional del comportament mecànic de biocompostos per a aplicacions en aeroestructures

Document:

Annexes

Author:

Pol Roca Iruela

Director/Co-director:

Ester Comellas Sanfeliu
Fermín Enrique Otero Gruer
Miquel Guinart Garcia-Talavera

Degree:

Bachelor in Aerospace Technologies Engineering

Examination session:

Spring 2024

BACHELOR FINAL THESIS

Contents

A	Material Cards	1
A.1	Components	2
A.2	Products	7
A.3	Plies	12
A.4	Laminates and ply groups	18
B	Theoretical sections	25
B.1	FEM Theory	25
B.1.1	Introduction	25
B.1.2	Discretization and independent variables	26
B.1.3	Finite Element Based Functions	26
B.1.4	Obtaining the weak form of a simple structural example	27
B.1.5	Galerkin method	28
B.1.6	Matrix system of equations and system solving	29
B.1.7	Gaussian Quadrature and assembly process	30
B.2	Linear elasticity	30
B.2.1	Introduction	30
B.2.2	Constitutive equation: Hooke's law	31
B.2.3	Elastic properties	33
B.2.4	Material symmetries	34
B.2.5	Compliance and stiffness matrices for orthotropic materials definition given elastic properties	37
B.3	Classical Lamination Theory	39
B.3.1	Introduction	39
B.3.2	Model limitations	39
B.3.3	Lamina model in material coordinates	40

B.3.4	Thermal effect in material coordinates	42
B.3.5	Lamina model in laminate coordinates	43
B.3.6	Laminate displacement distribution	46
B.3.7	Strain and curvature distribution on the laminate	47
B.3.8	Laminate constitutive equations	48
B.4	Transverse Shear Theory	52
B.4.1	Introduction	52
B.4.2	Model limitations	53
B.4.3	Balance of forces in the longitudinal direction	54
B.4.4	Relation of the bending moment to the normal stresses	55
B.4.5	Balance of moments in the plate element	56
B.4.6	Integration of tangential stresses in a lamina	57
B.4.7	Bureau Veritas method	58
C	Project schedule	60
C.1	Introduction to the schedule	60
C.2	Task description	61
C.3	Gantt chart	64
C.3.1	Color coding of the Gantt chart	64
D	Auxiliary Python Code	66
D.1	BV dictionary	66
D.2	Composite components dictionary	72
D.3	Composite plies dictionary	82
D.4	Composite laminates dictionary	93
D.5	Main File	95
	Bibliography	141

List of Figures

B.1	Graphical exemplification of the coordinate systems of the material or sheet (1-2) and the laminate (x-y) together with the angle θ that separates them. Extracted from [4].	44
B.2	Diagram with the positive directions of the forces per unit length (F) and the moments per unit length (M) for an element of the laminate. Extracted from [7].	49
B.3	Diagram with the positive directions of the forces per unit length (F, Q) and the moments per unit length (M) for an element of the laminate. Extracted from [7] and modified.	52
C.1	Gantt chart. Task number vs week number.	64

Appendix A

Material Cards

In this chapter, the so-called Material Cards are presented. These are short custom data sheets designed to present the mechanical properties of all the materials used throughout the project in a standardized and visually pleasing way to quickly identify the values to consult. This approach comes as a result of the different sources of data having different formats and many values interesting values mixed up with non-interesting ones from the project point of view.

A custom colour coding and code name is assigned to every card. The code name is used to unequivocally define the card throughout the report and at the Python auxiliary code variables. The colour coding separates and identifies four different kinds of Material Cards according to their nature.

- **Green - Components:** Green Material Cards characterize individual components of the composite materials. For example, a glass fibre/epoxy ply is formed by 2 components, the glass fibres and the epoxy, with different individual homogeneous mechanical characteristics.
- **Black - Products:** Black Material Cards offer a description of the fibre component and how it is organized as it is provided by the fibre supplier. The commercial name of the reinforcement product is also mentioned to identify it on the plies that use the product. In this material card, there is also indicated which type of plies (blue Material Cards) are used to characterize the product.
- **Blue - Plies or Laminae:** Blue Material Cards characterize all the different types of individual plies which are included in the laminates. They indicate the Product (black Material Card) on which they are based and also the components (green Material Cards) included on the ply and their relative proportion. With this information, an output detailing the mechanical properties of the ply is obtained and also

included in the Material Card. A functioning example would be the modelling of a bidirectional fibre product (black colour) which is done by defining two consecutive unidirectional plies (blue colour). The unidirectional ply would have as inputs the green Material Card of the fibre component (glass fibre for instance) and also the one from the matrix component (epoxy for instance) along with its relative proportion in the ply.

- **Red - Laminates or Groups of Plies:** Red Material Cards characterize all the standalone laminates or ply groupings appearing in the project. Their inputs are all the different types of plies (blue Material Cards) which form the laminate, their order and their orientation with respect to the laminate system of coordinates. Additionally, the order and orientation of the products represented by the plies of the laminate are included to assist at the manufacturing stage. For instance, if two plies, one at 0 degrees and the other at 90 degrees, are used to characterize a bidirectional product which has to be oriented at a ± 45 degrees on the laminate coordinate system, then at the ply level the order would be a ply at 45 degrees and another ply at -45 degrees but from a manufacturing point of view it is only one product positioned at 45 degrees.

A.1 Components

A legend is offered here to identify the different parameters included on the material cards:

E: Young's modulus (isotropic component).

E1: Young's modulus in the fibre direction.

E2: Young's modulus in the transverse direction.

nu: Poisson ratio parameter (isotropic component).

nu12: Poisson ratio parameter (fibre direction).

nu23: Poisson ratio parameter (transverse).

G: Shear modulus (isotropic component).

G12: Shear modulus (fibre direction).

G23: Shear modulus (transverse).

rho: Volumetric mass density.

sigXt: Tensile breaking stress in fibre direction.

sigXc: Compression breaking stress in fibre direction.

sigYt: Tensile breaking stress in direction normal to fibre direction.

sigYc: Compression breaking stress in direction normal to fibre direction.

sigS: Shear breaking stress.

sigFlex: Maximum stress at bending.

epsXt: Tensile breaking strain in fibre direction.

epsXc: Compression breaking strain in fibre direction.

epsYt: Tensile breaking strain in direction normal to fibre direction.

epsYc: Compression breaking strain in direction normal to fibre direction.

epsS: Shear breaking strain.

epsFlex: Elongation at maximum bending stress.

mPart: Mass fraction of a subcomponent to the total mass of the component.

COMPONENT:		EGlass		
Name:		E-Class Glass Fibre - Generic fibre		
Classification (Mechanical properties):		ISOTROPIC		
E	nu	G	rho	
73100	0.238	3e4	2.57	
MPa		MPa	g/cm3	
Classification (Failure properties):		TRANSVERSELY ISOTROPIC		
sigXt	sigXc	sigYt	sigYc	sigS
2750	1750	1750		1700
MPa	Mpa	MPa	MPa	MPa
epsXt	epsXc	epsYt	epsYc	epsS
3.8	2.4	2.4		5.6
%	%	%	%	%

COMPONENT:

GP33_1

Name: Sicomin SR GreenPoxy 33 / SD4772 - Epoxy resin and hardener

Classification:

ISOTROPIC

Subcomponents: Sicomin SR GreenPoxy 33 - Epoxy resin. - **GP33**

Sicomin SD 4772 hardener - Epoxy resin hardener. - **SD4772**

GP33	mPart	rho	SD4473	mPart	rho
78.74	1.159		21.26	0.93	
%	g/cm3		%	g/cm3	
E	nu	G	rho		
3200	0.39	1241	1.101		
MPa		MPa	g/cm3		
sigXt	sigXc	sigS	sigFlex		
56	110	51	100		
MPa	Mpa	MPa	MPa		
epsXt	epsXc	epsS	epsFlex		
1.0	9.0		3.7		
%	%	%	%		

COMPONENT:**PVC_HR80****Name:**

Aircell HR80 - PVC Foam core

Classification:**ISOTROPIC**

E	nu	G	rho
90	0.41	30	0.08
MPa		MPa	g/cm3
sigXt	sigXc	sigS	
2	1.6	1.2	
MPa	Mpa	MPa	
epsXt	epsXc	epsS	
-	-	-	
%	%	%	

COMPONENT:**CFib-HS****Name:** HS Class Carbon Fibre - Generic fibre**Classification (Mechanical properties):****TRANSVERSELY ISOTROPIC**

E1	E2	nu12	nu23	G12	G23	rho
238000	15000	0.3	0.02	50000	7353	1.79
MPa	MPa			MPa	MPa	g/cm3

Classification (Failure properties):**TRANSVERSELY ISOTROPIC**

sigXt	sigXc	sigYt	sigYc	sigS
3600	2140	135		1200
MPa	Mpa	MPa	MPa	MPa
epsXt	epsXc	epsYt	epsYc	epsS
1.5	0.9	0.9		2.4
%	%	%	%	%

A.2 Products**PRODUCT:****BXE446****Name:**BXE 446 1270 STD01 - E-Glass bidirectional ply $\pm 45^\circ$ **Description:**

1270 mm wide roll of E-Class type glass fibre. Its main components are two identical layers of unidirectional continuous glass fibres arranged next to each other forming a plane. Each layer has a surface density of $217g/m^2$. The 2 layers are stacked one on top of the other perpendicularly, one at a 45° angle between the roll longitudinal direction and the layer fibre direction and the other at a -45° angle following the same criterion. Additionally, there are fibers dedicated to maintain the cohesion of the main layers which do not contribute significantly to the mechanical behaviour of the product. These are E-Class glass fibre threads of stabilization weighting $2g/m^2$ and polyethersulfone (PES) knit yarn threads weighting $5g/m^2$.

Overall, the surface density of the product is $441g/m^2 \pm 5\%$.

Materials:E-Class Glass Fibre - Generic fibre. - **EGlass****Ply modelization:**

1. **P_UD_1** - $+45^\circ$ - Unidirectional, $\rho_{Fib} = 217g/m^2$
2. **P_UD_1** - -45° - Unidirectional, $\rho_{Fib} = 217g/m^2$

PRODUCT:**BXE300****Name:**BXE 300 1270 STD - E-Glass bidirectional ply $\pm 45^\circ$ **Description:**

1270 mm wide roll of E-Class type glass fibre. Its main components are two identical layers of unidirectional continuous glass fibres arranged next to each other forming a plane. Each layer has a surface density of $142g/m^2$. The 2 layers are stacked one on top of the other perpendicularly, one at a 45° angle between the roll longitudinal direction and the layer fibre direction and the other at a -45° angle following the same criterion. Additionally, there are fibers dedicated to maintain the cohesion of the main layers which do not contribute significantly to the mechanical behaviour of the product. These are E-Class glass fibre threads of stabilization weighting $6g/m^2$ and polyethersulfone (PES) knit yarn threads weighting $5g/m^2$.

Overall, the surface density of the product is $295g/m^2 \pm 5\%$.

Materials:E-Class Glass Fibre - Generic fibre. - **EGlass****Ply modelization:**

1. **P_UD_2** - $+45^\circ$ - Unidirectional, $\rho_{Fib} = 142g/m^2$
2. **P_UD_2** - -45° - Unidirectional, $\rho_{Fib} = 142g/m^2$

PRODUCT:**VV-29****Name:**

VV-29 - E-Glass woven roving fibre, plain weave

Description:

Roll of E-Class type glass fibre. It is a woven roving fabric made out of long continuous glass fibres. The weave pattern used is a plain weave, defined by warp tows in a 0° orientation interlacing perpendicularly with fill tows (90° orientation) alternatively. The warp tows follow the principal roll direction, 0° , aligned with the length of the roll.

Overall, the surface density of the product is $130g/m^2$.

The woven balance coefficient is a ratio of the mass of the warp (principal direction) orientated fabric to the total mass of the fabric. As this product is a plain weave fabric, the woven balance coefficient (C_{eq}) is $C_{eq} = 0.5$.

Materials:E-Class Glass Fibre - Generic fibre. - **EGlass****Ply modelization:**

1. **P_WR_1** - 0° - Woven roven, $\rho_{Fib} = 130g/m^2$, $C_{eq} = 0.5$

PRODUCT:**HR80****Name:**

Aircell HR80 - PVC Foam core 3mm thick

Description:

3 mm thick PVC foam flexible sheet. Its material is a partially cross-linked, structural cellular PVC foam, ideal for composite sandwich structures.

Overall, its density is $0.08g/cm^3$.

Materials:Aircell HR80 - PVC Foam core. - **PVC_HR80****Ply modelization:**

1. **Core1** - 0° - PVC Foam core, thickness = $3mm$, $\rho = 0.08g/cm^3$

PRODUCT:**GV220U****Name:**

CV 220 U - HS-Carbon Fibre unidirectional ply

Description:

1000 mm or 500 mm wide roll of HS type carbon fibre. The product is one layer of unidirectional continuous carbon fibres (6K, 400 tex) arranged next to each other forming a plane. Additionally, there are perpendicular unidirectional weft fibers dedicated to maintain the cohesion of the carbon threads which do not contribute significantly to the mechanical behaviour of the product. These are E-Class glass fibre threads (glass EC 9 68). The mass rate between carbon fibres and weft glass fibers is 88% to 12% respectively.

Overall, the surface density of the product is $220g/m^2$ and its thickness is $0.2mm$.

Materials:HS Class Carbon Fibre - Generic fibre. - **CFib-HS****Ply modelization:**

1. **P_CU_2** - 0° - Unidirectional, $\rho_{Fib} = 193.6g/m^2$

A.3 Plies

A legend is offered here to identify the different parameters included on the material cards:

VPart: Volume fraction of a subcomponent to the total volume of the ply.

mPart: Mass fraction of a subcomponent to the total mass of the ply.

rho: Volumetric mass density.

E1: Young's modulus in ply principal (fibre) direction.

E2: Young's modulus in ply secondary direction.

nu12: Poisson ratio parameter between principal and secondary directions.

nu1z: Poisson ratio parameter between principal and plane-normal directions.

nu2z: Poisson ratio parameter between secondary and plane-normal directions.

G12: In-plane shear modulus.

G1z: Shear modulus on the plane defined by the principal and plane-normal directions.

G2z: Shear modulus on the plane defined by the secondary and plane-normal directions.

rhoSup: Planar mass density.

sigXt: Tensile breaking stress in principal direction.

sigXc: Compression breaking stress in principal direction.

sigYt: Tensile breaking stress in secondary direction.

sigYc: Compression breaking stress in secondary direction.

sigS: Shear breaking stress.

epsXt: Tensile breaking strain in principal direction.

epsYt: Tensile breaking strain in secondary direction.

epsYc: Compression breaking strain in secondary direction.

epsS: Shear breaking strain.

SHELL PLY:

P_UD_1

Name:

BXE446 unidirectional subcomponent
 - E-Glass / Epoxy unidirectional ply

Subcomponents:

E-Class Glass Fibre - Generic fibre. - **EGlass**

Sicomin SR GreenPoxy 33 / SD4772 - Epoxy resin and hardener. - **GP33_1**

EGlass	VPart	mPart	rho	GP33_1	VPart	mPart	rho
44.63	65.0	2.57		55.37	35.0	1.12	
%	%	g/cm3		%	%	g/cm3	

Classification:

TRANSVERSELY ISOTROPIC

E1	E2	E3	nu12	nu1z	nu2z	nu21	nuz1	nuz2
34538.0	7576.0	7576.0	0.29	0.29	0.29	0.06	0.06	0.29
MPa	MPa	MPa						
G12	G1z	G2z	rho	rhoSup	rhoFib	thknss		
2671.0	2671.0	1870.0	1.76	337.35	217	0.19		
MPa	MPa	MPa	g/cm3	g/m2	g/m2	mm		
sigXt	sigXc	sigYt	sigYc	sigS	sigIL1	sigIL2		
933.0	622.0	40.0	117.0	48.0	47.0	48.0		
MPa	MPa	MPa	MPa	MPa	MPa	MPa		
epsXt	epsXc	epsYt	epsYc	epsS	epsIL1	epsIL2		
2.7	1.8	0.53	1.55	1.8	2.5	1.8		
%	%	%	%	%	%	%		
R =	35186.0	2238.0	0.0	S =	2.9e-05	-8e-06	0.0	
[MPa]	2238.0	7718.0	0.0	[1/MPa]	-8e-06	0.0001	0.0	
	0.0	0.0	2671.0		0.0	0.0	0.000374	

SHELL PLY:**P_UD_2****Name:**

BXE300 unidirectional subcomponent
- E-Glass / Epoxy unidirectional ply

Subcomponents:

E-Class Glass Fibre - Generic fibre. - EGlass

Sicomin SR GreenPoxy 33 / SD4772 - Epoxy resin and hardener. - GP33_1

EGlass	VPart	mPart	rho	GP33_1	VPart	mPart	rho
44.63	65.0	2.57		55.37	35.0	1.12	
%	%	g/cm3		%	%	g/cm3	

Classification:**TRANSVERSELY ISOTROPIC**

E1	E2	E3	nu12	nu1z	nu2z	nu21	nuz1	nuz2
34538.0	7576.0	7576.0	0.29	0.29	0.29	0.06	0.06	0.29
MPa	MPa	MPa						
G12	G1z	G2z	rho	rhoSup	rhoFib	thknss		
2671.0	2671.0	1870.0	1.76	226.92	142	0.13		
MPa	MPa	MPa	g/cm3	g/m2	g/m2	mm		
sigXt	sigXc	sigYt	sigYc	sigS	sigIL1	sigIL2		
933.0	622.0	40.0	117.0	48.0	47.0	48.0		
MPa	MPa	MPa	MPa	MPa	MPa	MPa		
epsXt	epsXc	epsYt	epsYc	epsS	epsIL1	epsIL2		
2.7	1.8	0.53	1.55	1.8	2.5	1.8		
%	%	%	%	%	%	%		
R =	35186.0	2238.0	0.0	S =	2.9e-05	-8e-06	0.0	
[MPa]	2238.0	7718.0	0.0	[1/MPa]	-8e-06	0.0001	0.0	
	0.0	0.0	2671.0		0.0	0.0	0.000374	

SHELL PLY:**P_WR_1****Name:** VV-29 plain woven roven - E-Glass / Epoxy woven ply**Subcomponents:** E-Class Glass Fibre - Generic fibre. - **EGlass**Sicomin SR GreenPoxy 33 / SD4772 - Epoxy resin and hardener. - **GP33_1**

EGlass	VPart	mPart	rho	GP33_1	VPart	mPart	rho
44.63	65.0	2.57		55.37	35.0	1.12	
%	%	g/cm3		%	%	g/cm3	

Classification:**TRANSVERSELY ISOTROPIC**

E1	E2	E3	nu12	nu1z	nu2z	nu21	nuz1	nuz2
21219.0	21219.0	7576.0	0.1	0.29	0.29	0.1	0.18	0.18
MPa	MPa	MPa						
G12	G1z	G2z	rho	rhoSup	rhoFib	thknss		
2675.0	2407.0	2407.0	1.76	200.0	130	0.11		
MPa	MPa	MPa	g/cm3	g/m2	g/m2	mm		
sigXt	sigXc	sigYt	sigYc	sigS	sigIL1	sigIL2		
382.0	382.0	382.0	382.0	40.0	43.0	43.0		
MPa	MPa	MPa	MPa	MPa	MPa	MPa		
epsXt	epsXc	epsYt	epsYc	epsS	epsIL1	epsIL2		
1.8	1.8	1.8	1.8	1.5	1.8	1.8		
%	%	%	%	%	%	%		
R =	21452.0	2238.0	0.0	S =	4.7e-05	-5e-06	0.0	
[MPa]	2238.0	21452.0	0.0	[1/MPa]	-5e-06	0.0	0.0	
	0.0	0.0	2675.0		0.0	0.0	0.000374	

SHELL PLY:**Core1****Name:** Aircell HR80 3mm thick - PVC Foam core**Subcomponents:** Aircell HR80 - PVC Foam core. - **PVC_HR80**

PVCHR80	VPart	mPart	rho
100	100	0.08	
%	%	g/cm3	

Classification:**ISOTROPIC**

E1	E2	E3	nu12	nu1z	nu2z	nu21	nuz1	nuz2
90	90	90	0.41	0.41	0.41	0.41	0.41	0.41
MPa	MPa	MPa						

G12	G1z	G2z	rho	rhoSup	rhoFib	thknss
30	30	30	0.08	240.0	240.0	3.0
MPa	MPa	MPa	g/cm3	g/m2	g/m2	mm

sigXt	sigXc	sigYt	sigYc	sigS	sigIL1	sigIL2
2.0	2.0	2.0	2.0	1.0	1.0	1.0
MPa	MPa	MPa	MPa	MPa	MPa	MPa

epsXt	epsXc	epsYt	epsYc	epsS	epsIL1	epsIL2
-	-	-	-	-	-	-
%	%	%	%	%	%	%

R =	108.0	44.0	0.0	S =	0.01111	-0.0046	0.0
[MPa]	44.0	108.0	0.0	[1/MPa]	-0.0046	0.01111	0.0
	0.0	0.0	30.0		0.0	0.0	0.033333

SHELL PLY:

P_CU_2

Name:

GV220U unidirectional component - HS-Carbon / Epoxy ply

Subcomponents: HS-Class Carbon Fibre - Generic fibre. - **CFib-HS**

Sicomin SR GreenPoxy 33 / SD4772 - Epoxy resin and hardener. - **GP33_1**

CFib-HS	VPart	mPart	rho	GP33_1	VPart	mPart	rho
53.34	65.0	1.79		46.66	35.0	1.1	
%	%	g/cm3		%	%	g/cm3	

Classification:

TRANSVERSELY ISOTROPIC

E1	E2	E3	nu12	nu1z	nu2z	nu21	nuz1	nuz2
128438	6311.0	6311.0	0.27	0.27	0.15	0.01	0.01	0.15
MPa	MPa	MPa						
G12	G1z	G2z	rho	rhoSup	rhoFib	thknss		
3187.0	3187.0	2231.0	1.47	324.25	193.6	0.2		
MPa	MPa	MPa	g/cm3	g/m2	g/m2	mm		
sigXt	sigXc	sigYt	sigYc	sigS	sigIL1	sigIL2		
1541.0	1092.0	63.0	145.0	51.0	42.0	51.0		
MPa	MPa	MPa	MPa	MPa	MPa	MPa		
epsXt	epsXc	epsYt	epsYc	epsS	epsIL1	epsIL2		
1.2	0.85	1.0	2.3	1.6	1.9	1.6		
%	%	%	%	%	%	%		
R =	128912	1733.0	0.0	S =	8e-06	-2e-06	0.0	
[MPa]	1733.0	6334.0	0.0	[1/MPa]	-2e-06	0.0002	0.0	
	0.0	0.0	3187.0		0.0	0.0	0.000314	

A.4 Laminates and ply groups

LAMINATE:

L1

Name: Laminate 1 - E-Glass / Epoxy laminate

Subcomponents:

BXE446 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_1**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Classification: ANISOTROPIC

Product disposition:

1. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)
2. **BXE446** - 0° - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
3. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)

Model:

Number	Code	Angle	Type	Thknss	rhoSup
4	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
3	P_UD_1	45	E-G/Ep UDirect	0.19	337.35
2	P_UD_1	-45	E-G/Ep UDirect	0.19	337.35
1	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
Total	L1			0.61	1074.69
		deg		mm	g/m2

LAMINATE:**L2****Name:** Laminate 2 - E-Glass / Epoxy laminate**Subcomponents:**

BXE300 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_2**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Classification: **ANISOTROPIC****Product disposition:**

1. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)
2. **BXE300** - 0° - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
3. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)

Model:

Number	Code	Angle	Type	Thknss	rhoSup
4	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
3	P_UD_2	45	E-G/Ep UDirect	0.12	223.96
2	P_UD_2	-45	E-G/Ep UDirect	0.12	223.96
1	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
Total	L2			0.47	847.92
		deg		mm	g/m2

LAMINATE:**L3****Name:** Laminate 3 - E-Glass / Epoxy and PVC Foam laminate**Subcomponents:**

BXE446 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_1**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Aircell HR80 3mm thick - PVC Foam core. - **Core1**

Classification:**ANISOTROPIC****Product disposition:**

1. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)
2. **HR80** - 0° - PVC Foam core, 3mm thickness
3. **BXE446** - 0° - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
4. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)

Model:

Number	Code	Angle	Type	Thknss	rhoSup
5	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
4	P_UD_1	45	E-G/Ep UDirect	0.19	337.35
3	P_UD_1	-45	E-G/Ep UDirect	0.19	337.35
2	Core1	0	PVC/ Foam	3.0	240.0
1	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
Total	L3			3.61	1314.69
		deg		mm	g/m2

LAMINATE:**L4****Name:** Laminate 4 - E-Glass / Epoxy and PVC Foam laminate**Subcomponents:**

BXE300 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_2**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Aircell HR80 3mm thick - PVC Foam core. - **Core1**

Classification:**ANISOTROPIC****Product disposition:**

1. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)
2. **HR80** - 0° - PVC Foam core, 3mm thickness
3. **BXE300** - 0° - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
4. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)

Model:

Number	Code	Angle	Type	Thknss	rhoSup
5	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
4	P_UD_2	45	E-G/Ep UDirect	0.13	226.92
3	P_UD_2	-45	E-G/Ep UDirect	0.13	226.92
2	Core1	0	PVC/ Foam	3.0	240.0
1	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
Total	L4			3.48	1093.85
		deg		mm	g/m2

LAMINATE:**L5****Name:** Laminate 5 - E-Glass / Epoxy and PVC Foam laminate**Subcomponents:**

BXE300 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_2**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Aircell HR80 3mm thick - PVC Foam core. - **Core1**

Classification:**ANISOTROPIC****Product disposition:**

1. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)
2. **BXE300** - 0° (Upside down) - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
3. **HR80** - 0° - PVC Foam core, 3mm thickness
4. **BXE300** - 0° - E-Glass/Epoxy Bidirectional ($\pm 45^\circ$), two layers (no weave)
5. **VV-29** - 0° - E-Glass/Epoxy Woven Roven, plain weave (0° - 90°)

LAMINATE:**L5****Name:** Laminate 5 - E-Glass / Epoxy and PVC Foam laminate**Subcomponents:**

BXE300 unidirectional subcomponent - E-Glass / Epoxy unidirectional ply. - **P_UD_2**

VV-29 plain woven roven - E-Glass / Epoxy woven ply. - **P_WR_1**

Aircell HR80 3mm thick - PVC Foam core. - **Core1**

Classification:**ANISOTROPIC****Model:**

Number	Code	Angle	Type	Thknss	rhoSup
7	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
6	P_UD_2	45	E-G/Ep UDirect	0.12	223.96
5	P_UD_2	-45	E-G/Ep UDirect	0.12	223.96
4	Core1	0	PVC/ Foam	3.0	240.0
3	P_UD_2	-45	E-G/Ep UDirect	0.12	223.96
2	P_UD_2	45	E-G/Ep UDirect	0.12	223.96
1	P_WR_1	0	E-G/Ep WovenR	0.11	200.0
Total	L5			3.73	1535.85
		deg		mm	g/m2

LAMINATE:**LB****Name:** Ply group LB - HS-Carbon / Epoxy laminate**Subcomponents:**GV220U unidirectional component - HS-Carbon / Epoxy unidirectional ply. - **P_CU_2****Classification:** **ANISOTROPIC****Product disposition:**

1. **GV220U** - 0° - HS-Carbon/Epoxy Unidirectional
2. **GV220U** - 45° - HS-Carbon/Epoxy Unidirectional
3. **GV220U** - -45° - HS-Carbon/Epoxy Unidirectional
4. **GV220U** - 90° - HS-Carbon/Epoxy Unidirectional

Model:

Number	Code	Angle	Type	Thknss	rhoSup
4	P_CU_2	90	C-HS/Ep UDirect	0.2	324.25
3	P_CU_2	-45	C-HS/Ep UDirect	0.2	324.25
2	P_CU_2	45	C-HS/Ep UDirect	0.2	324.25
1	P_CU_2	0	C-HS/Ep UDirect	0.2	324.25
Total	LB			0.81	1296.98
		deg		mm	g/m2

Appendix B

Methodology and theory developments

This chapter contains a few fragments of the report with a high concentration of theoretical content which are not fully developed in the Report document to not extend it more than it is required. Nonetheless, the theory included in this chapter has been directly applied in the development of the auxiliary Python code or has been learned to understand the operations performed in MSC Nastran. For this reason, it should appear in the Bachelor thesis as it constitutes an important part of the development of the whole project.

B.1 Theoretical overview of the FEM

In this section, the reader can find the full extension of the Theoretical overview of the FEM section in the “NASTRAN as a Finite Element Method structural solver” chapter of the Report document. It is advisable to read said chapter to acquire the context of the methodology described here.

B.1.1 Introduction

A brief and mathematically shallow explanation of the mathematical foundation of FEM is presented here. The objective of this explanation is not to prove or formulate the method but introduce key concepts which characterize the method, mainly because the FEM is used in this work at a user level and steps taken beyond proving a certain level of understanding of it fall out of scope. The contents on this section have been learned from the lectures [1]. An available source around the same method in detail can be found at [2].

B.1.2 Discretization and independent variables

Given a static structural problem, a certain geometry is made out of certain materials with different behaviours and subjected to some force and displacement constraints (called boundary conditions). Solving the problem involves finding the final positions of the points in the geometry and afterwards, internal stress can be computed via each material constitutive model. As the FEM is a numerical method, the geometry must be discretized into a sufficiently large number of small finite elements, which will be adjacent polygons or polyhedrons (not necessarily regular) whose shared vertex will be called nodes. Overall, the set of nodes and defined finite elements is called the mesh. The nodes are located in precise locations of the space or plane and the numerical problem will be considered solved when the desired properties, such as displacements or stresses, are found at every node of the mesh.

The independent variables of the problem are decided to be the displacements (u) of each node in the mesh. Those are vectors that point to the final location of a node when applied to the original location of the node. Their spatial derivative (u') is the strain at the corresponding node (the spatial direction on which u is derived determines the specific component of the strain matrix). When applying the constitutive model of the material containing a node, a stress state at that node is typically obtained given the strain state at the node. Finally, the stress state can be translated into a force when considering a certain surface (usually related to a finite element face, delimited by nodes). This means that forces can be related to spatial derivatives of the displacement (u'), and therefore, a boundary condition consisting of some kind of force applied to the geometry will be translated to a u' boundary condition of the nodes involved. This type of boundary condition is called the Neumann boundary condition. On the other hand, a displacement constraint boundary condition, called the Dirichlet boundary condition, is a direct declaration of the displacement u of the involved nodes.

B.1.3 Finite Element Based Functions

As the exact function describing all the displacements u of the problem spacial domain Ω is unknown, the following assumption is made. The general solution function describing the displacements u is a linear combination of several functions.

$$u(\vec{x}) = \vec{N}(\vec{x}) \cdot \vec{d} \quad (\text{B.1})$$

where $u(\vec{x})$ is the displacement depending on the position \vec{x} , $\vec{N}(\vec{x})$ is a vector containing every function depending on the position \vec{x} that makes $u(\vec{x})$ through linear combination and \vec{d} is a vector containing all the coefficients multiplying each of the functions in $\vec{N}(\vec{x})$.

Now some special functions will be proposed to construct the vector $\vec{N}(\vec{x})$, called Finite Element Base Functions. These functions are piece-wise functions characterized for being equal to 0 at every node of a geometry's mesh except one single node, in which the value at that exact point is equal to 1. The vector $\vec{N}(\vec{x})$ will have as many functions as nodes in the considered mesh, each one pointing at a certain distinct node in which the image value is 1 instead of zero. Following these definitions, the independent variables to be solved are all the elements of the vector \vec{d} , as they have to be equal to the actual displacement of each node in order to comply with (B.1).

B.1.4 Obtaining the weak form of a simple structural example

The next step would be to have the problem governing equation in weak form. For instance, in a very simple structural unidirectional beam problem we would have the following strong form governing equation coming from an equilibrium of forces:

$$\frac{d\sigma}{dx} + E \cdot q(x) = 0 \quad (\text{B.2})$$

where σ is the stress present at a point x , and $E \cdot q$ is the external distributed force per unit area, all in the unique axial dimension.

The constitutive equation could be expressed as:

$$\sigma = E \cdot \varepsilon = E \cdot \frac{du}{dx} = E \cdot u' \quad (\text{B.3})$$

where E is a constant value Young's modulus and u is the displacement at a point.

The yet strong form governing equation could be put in terms of displacement u as:

$$\frac{d^2u}{dx^2} + q(x) = u'' + q(x) = 0 \quad (\text{B.4})$$

If we impose the definition in (B.1) using Finite Element Base Functions, most probably the governing equation would be unsolvable. No 2nd order derivatives can be currently obtained at any domain point on the functions in \vec{N} , but also the exact solution is not given in terms of the \vec{N} functions. A numerical approach

has to be taken, in which a good enough approximation is considered satisfactory. For this reason, a residual function r is defined which equals zero when the solution is exact and should be minimized to improve precision.

$$r(x) = u''(x) + q(x) \approx 0 \quad (\text{B.5})$$

It can be proved that a minimization technique of the residual r is achieved when imposing:

$$\int_{\Omega} v(x) \cdot r(x) dx = \int_{\Omega} v(x) \cdot [u''(x) + q(x)] dx = 0 \quad (\text{B.6})$$

where Ω represents the integration domain comprised of all the space covering the geometry of the problem, $r(x)$ is the aforementioned residual function and $v(x)$ is the so-called virtual displacement function or test function.

Mathematically, it can be proved that the previous expression can be reformulated to prevent the appearance of the 2nd derivative $u''(x)$ by making use of the Dirichlet boundary conditions of the problem. Those dictate (by the necessary mathematical properties of the function v) that at each point x_D where a Dirichlet condition is applied, then $v(x_D) = 0$.

$$\int_{\Omega} \frac{dv}{dx} \cdot \frac{du}{dx} dx = \int_{\Omega} v(x) \cdot q(x) dx + \left. \frac{du}{dx} \right|_{x=L} \cdot v(x=L) \quad (\text{B.7})$$

where $\left. \frac{du}{dx} \right|_{x=L}$ can be obtained through the present Neumann boundary conditions at $x = L$.

The expression in (B.7) is called the weak form of the governing equation in (B.5) and depends on the test function v and the displacement function u which is called the trial function in general terms. It is compatible with Finite Element Base Functions as both the integrals of u and u' are defined at any point. Incidentally, it is also a form of the Principle of Virtual Work.

B.1.5 Galerkin method

Using the Galerkin method, the virtual displacements or test functions (v) are defined using the exact same base functions as the trial function but with independent coefficients (elements in vector \vec{c}):

$$u(\vec{x}) = \vec{N}(\vec{x}) \cdot \vec{d} \quad \rightarrow \quad v(\vec{x}) = \vec{N}(\vec{x}) \cdot \vec{c} \quad (\text{B.8})$$

The weak equation in (B.7) is then written as:

$$\vec{c}^T \cdot \left[\int_{\Omega} \frac{d\vec{N}^T}{dx} \cdot \frac{d\vec{N}}{dx} dx \right] \cdot \vec{d} = \vec{c}^T \cdot \left[\int_{\Omega} \vec{N}^T \cdot q(x) dx \right] + \vec{c}^T \cdot \left[\frac{du}{dx} \Big|_{x=L} \cdot \vec{N}^T(x=L) \right] \quad (\text{B.9})$$

B.1.6 Matrix system of equations and system solving

The weak form equation in the form of (B.9) can be contracted into a matrix form to simplify its writing:

$$\vec{c}^T \cdot \mathbf{K} \cdot \vec{d} = \vec{c}^T \cdot \vec{F} \quad (\text{B.10})$$

$$\vec{c}^T \cdot \mathbf{K} \cdot \vec{d} - \vec{c}^T \cdot \vec{F} = 0 \quad (\text{B.11})$$

The previous expression is a system of equations in which nodes with Dirichlet boundary conditions are treated differently as other nodes because $c = 0$ at those points and the equation holds for any value of d (or physically speaking, the solution is already imposed and therefore known). The matrices and vectors are then split to isolate nodes associated with imposed displacements or Dirichlet conditions (subscript r) or free-moving nodes (subscript l).

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{rr} & \mathbf{K}_{rl} \\ \mathbf{K}_{lr} & \mathbf{K}_{ll} \end{bmatrix} \quad \vec{F} = \begin{bmatrix} \vec{F}_r \\ \vec{F}_l \end{bmatrix} \quad \vec{d} = \begin{bmatrix} \vec{d}_r \\ \vec{d}_l \end{bmatrix} \quad (\text{B.12})$$

The system of equations in (B.11) is then:

$$\begin{cases} \vec{c}_r^T [\mathbf{K}_{rr} \vec{d}_r + \mathbf{K}_{rl} \vec{d}_l - \vec{F}_r] = 0 \\ \vec{c}_l^T [\mathbf{K}_{lr} \vec{d}_r + \mathbf{K}_{ll} \vec{d}_l - \vec{F}_l] = 0 \end{cases} \quad (\text{B.13})$$

where the first row is true regardless of the values at \vec{d}_r as mathematically $\vec{c}_r = \vec{0}$.

Finally, the solution at all the unknown displacement nodes is given when solving the next system of equations:

$$\mathbf{K}_{ll} \vec{d}_l = \vec{F}_l - \mathbf{K}_{lr} \vec{d}_r \quad (\text{B.14})$$

B.1.7 Gaussian Quadrature and assembly process

Crucially, the matrix \mathbf{K} and vector \vec{F} are made out of mostly geometric parameters ($\vec{N}(\vec{x})$ depends on the positions of the nodes in the mesh) and are integral terms over the geometry domain space. Integration can be done individually for each finite element of the mesh and then added to the result.

If the type of finite element is well defined and studied, the value of the integral can be estimated to various degrees of accuracy by using the Gaussian Quadrature Rule. This requires a change of space to turn the real domain into another with a definite regular shape in which Gaussian Quadrature is doable. In this specific space, the integral can be computed by adding the value of its interior contents calculated at specific points (called Gauss Points) in the domain, which are weighted according to specific tabulated values. The Jacobian of the change of space is needed to correct the result of the integral to match that of the real domain and to track the position of the Gauss Points at the real domain coordinates. This whole procedure can be effectively generalized for each type of discretization element and allows the integration to be approximated for an element of any size and shape.

The values from the integral of individual elements have to be added into the whole matrix or vector containing information of every node in the problem, so a process called assembly is done to add the local element matrix values into the specific element position on the global problem matrix and finally form the system of equations necessary to solve the problem.

B.2 Linear elasticity and elastic properties

In this section, the reader can find the full extension of the Linear elasticity and elastic properties section in the Methodology chapter of the Report document. It is advisable to read said chapter to acquire the context of the methodology described here.

B.2.1 Introduction

When a general material is referred to as a linear elastic material, it is understood that said material presents strains (displacements) which are a linear combination of the loads acting on the material, along with the property of returning to the same strain state for a particular stress state no matter the history of states that

could have happened in between the two equal states. In practice, this behaviour is valid only for some materials (typically metals) and only when low strains are induced.

B.2.2 Constitutive equation: Hooke's law

The constitutive equations of linear elasticity are characterized by imposing a general constant proportional relation between stress present in the material and the resulting strain observed at the same point.

Both the stress and strain state of a point in a material are defined as two-dimensional tensors (3x3 matrices for 3 spatial coordinates). They are also symmetrical tensors because if it were not the case, then the static equilibrium of moments at an infinitesimal point of a solid is not maintained.

$$\bar{\bar{\sigma}} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_{22} & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{pmatrix} = \begin{pmatrix} \sigma_1 & \tau_{12} & \tau_{13} \\ \tau_{12} & \sigma_2 & \tau_{23} \\ \tau_{13} & \tau_{23} & \sigma_3 \end{pmatrix} \quad \bar{\bar{\varepsilon}} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{12} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{13} & \varepsilon_{23} & \varepsilon_{33} \end{pmatrix} = \begin{pmatrix} \varepsilon_1 & \frac{\gamma_{12}}{2} & \frac{\gamma_{13}}{2} \\ \frac{\gamma_{12}}{2} & \varepsilon_2 & \frac{\gamma_{23}}{2} \\ \frac{\gamma_{13}}{2} & \frac{\gamma_{23}}{2} & \varepsilon_3 \end{pmatrix} \quad (\text{B.15})$$

In order to relate every stress element with every strain element with a distinct proportionality constant, a four-dimensional tensor is needed, called the stiffness tensor $\bar{\bar{\bar{C}}}$ [3]. In (B.16) the stiffness tensor is written in tensor form and also in index form.

$$\bar{\bar{\sigma}} = \bar{\bar{\bar{C}}} \bar{\bar{\varepsilon}} \quad ; \quad \sigma_{ij} = C_{ijkl} \cdot \varepsilon_{kl} \quad (\text{B.16})$$

By making use of Voigt notation, the order of the tensors can decrease and the expression becomes easier to operate with [3]. This reduction is possible when the stress and strain matrices are symmetric, like in this case. The stress and strain vectors in Voigt notation will be written with a tilde like $\tilde{\sigma}$ and $\tilde{\varepsilon}$.

$$\begin{aligned}
\bar{\bar{\sigma}} \iff \tilde{\sigma} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{13} \\ \tau_{12} \end{pmatrix} & \quad \bar{\bar{\varepsilon}} \iff \tilde{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \gamma_{23} \\ \gamma_{13} \\ \gamma_{12} \end{pmatrix} & \quad \bar{\bar{\bar{C}}} \iff \bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix}
\end{aligned} \tag{B.17}$$

Then Hooke's law in Voigt notation ends up being written like this:

$$\tilde{\sigma} = \bar{\bar{C}} \tilde{\varepsilon} \quad ; \quad \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{13} \\ \tau_{12} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix} \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \gamma_{23} \\ \gamma_{13} \\ \gamma_{12} \end{pmatrix} \tag{B.18}$$

This formulation then implies that each material can be characterized by its unique stiffness matrix. Two materials with the same stiffness matrix will behave in the same manner.

Finally, an inverse relation is usually used too [3].

$$\tilde{\varepsilon} = \bar{\bar{S}} \tilde{\sigma} \quad ; \quad \bar{\bar{S}} = \bar{\bar{C}}^{-1} \tag{B.19}$$

The newly defined matrix $\bar{\bar{S}}$ is called the compliance matrix, and it is characterized as being the inverse of the stiffness matrix $\bar{\bar{C}}$.

B.2.3 Elastic properties

Material elastic properties, also called engineering properties, are some parameters easily measurable in standard material tests which can fully define the constitutive equation of a material. They are not necessarily the elements of the stiffness matrix, but they can be obtained given the elements of the stiffness matrix and vice versa.

One of the tests used for the determination of the elastic properties is the traction test. In this test, all tensions in every point of the material are forced to be 0 except one tension σ , controlled by a traction force. The magnitude of the strains during the test can be measured, and the proportionality parameters between stresses and strains can be used to define some elastic properties of the tested material.

On the other hand, a shear test is used in the same manner as the traction test, only that in this case the isolated applied load is a shear stress τ .

Young's modulus (E)

Young's modulus or elastic modulus (E) is an elastic property obtained in a traction test. It relates stress in one direction with the strain in the same direction. When the traction direction is labeled as x its definition is:

$$E_1 = \frac{\sigma_1}{\varepsilon_1} \quad (\text{B.20})$$

It can also be found from the elements of the compliance matrix \bar{S} [3]:

$$E_i = \frac{1}{S_{ii}} \text{ s.t. } i \in \{1, 2, 3\} \quad (\text{B.21})$$

Poisson ratio (ν)

The Poisson ratio (ν) is an elastic property obtainable in a traction test. It is described as the ratio of strains in two orthogonal directions when a stress is applied following one of those directions. If the traction (stress) direction is labelled as i and its measured orthogonal direction is labelled as j then its definition follows the next formula:

$$\nu_{ij} = -\frac{\varepsilon_j}{\varepsilon_i} \text{ s.t. } i, j \in \{1, 2, 3\} \quad (\text{B.22})$$

It can also be found from the elements of the compliance matrix $\overline{\overline{S}}$ [3]:

$$\nu_{ij} = -\frac{S_{ji}}{S_{ii}} \text{ s.t. } i, j \in \{1, 2, 3\} \quad (\text{B.23})$$

Shear modulus (G)

The shear modulus (G) is an elastic property obtainable in a shear test. Similar to the concept of Young's modulus, it is defined as the ratio between one of the three possible tangential stresses (τ) and its corresponding shear strain elements (γ). It is defined by the following formula, where 3 distinct moduli appear for 3 dimensions (G_{12} , G_{23} , G_{13}).

$$G_{ij} = \frac{\tau_{ij}}{\gamma_{ij}} \text{ s.t. } i, j \in \{1, 2, 3\} \quad (\text{B.24})$$

It can also be found from the elements of the compliance matrix $\overline{\overline{S}}$ [3]:

$$G_{12} = \frac{1}{S_{66}} \quad G_{13} = \frac{1}{S_{55}} \quad G_{23} = \frac{1}{S_{44}} \quad (\text{B.25})$$

B.2.4 Material symmetries

If the overall structure of a material has some kind of geometrical symmetry, then it can be represented in the stiffness matrix as a simplification of its formulation when the axis defined for the stress and strain tensors are aligned with these symmetric features.

Anisotropy

For a generic non-symmetric material, that is, an anisotropic material, the stiffness matrix remains untouched. Nonetheless, it can be proved by computing the strain energy that the stiffness matrix is symmetric.

This type of matrix is a general matrix for any kind of material and orientation. This structure can also present itself for materials with some kind of symmetry but expressed with coordinate directions not aligned with the symmetry directions of the material [4].

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ & & C_{33} & C_{34} & C_{35} & C_{36} \\ & & & C_{44} & C_{45} & C_{46} \\ & & & & C_{55} & C_{56} \\ SYM & & & & & C_{66} \end{pmatrix} \quad (\text{B.26})$$

Therefore, an anisotropic material can be fully defined given 21 independent parameters.

Orthotropy

An orthotropic material is characterized by having symmetrical properties with respect to planes which are orthogonal between each other. In other words, its properties are different along each coordinate direction (and Cartesian coordinate directions are orthogonal between each other). The stiffness matrix aligned with the different variation directions results as [4]:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ & C_{22} & C_{23} & 0 & 0 & 0 \\ & & C_{33} & 0 & 0 & 0 \\ & & & C_{44} & 0 & 0 \\ & & & & C_{55} & 0 \\ SYM & & & & & C_{66} \end{pmatrix} \quad (\text{B.27})$$

Therefore, an orthotropic material can be fully defined given 9 independent parameters.

The elements of the stiffness matrix can be found by inverting the compliance matrix. The formulation for the inversion ends up being [3]:

$$C_{ii} = \frac{S_{jj} \cdot S_{kk} - S_{jk}^2}{S_{CONSTANT}} \text{ s.t. } i, j, k \in \{1, 2, 3\} \quad C_{ii} = \frac{1}{S_{ii}} \text{ s.t. } i \in \{4, 5, 6\} \quad (\text{B.28})$$

$$C_{ij} = \frac{S_{ik} \cdot S_{jk} - S_{ij} \cdot S_{kk}}{S_{CONSTANT}} \text{ s.t. } i, j, k \in \{1, 2, 3\}$$

$$S_{CONSTANT} = S_{11} \cdot S_{22} \cdot S_{33} - S_{11} \cdot S_{23}^2 - S_{22} \cdot S_{13}^2 - S_{33} \cdot S_{12}^2 + 2 \cdot S_{12} \cdot S_{23} \cdot S_{13}$$

Transverse isotropy

A transversely isotropic material is an orthotropic material which maintains the same properties along two of the three orthogonal axes. That is, it has the same properties along a distinct direction and also along a plane perpendicular to it. The stiffness matrix aligned with the distinct direction (direction 3) results as [4]:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ & C_{11} & C_{13} & 0 & 0 & 0 \\ & & C_{33} & 0 & 0 & 0 \\ & & & C_{44} & 0 & 0 \\ & & & & C_{44} & 0 \\ \text{SYM} & & & & & \frac{C_{11} - C_{12}}{2} \end{pmatrix} \quad (\text{B.29})$$

Therefore, a transverse isotropic material can be fully defined given 5 independent parameters.

Isotropy

An isotropic material maintains the same properties regardless of the direction chosen to measure them. Its stiffness matrix results as [4]:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{12} & 0 & 0 & 0 \\ & C_{11} & C_{12} & 0 & 0 & 0 \\ & & C_{11} & 0 & 0 & 0 \\ & & & \frac{C_{11} - C_{12}}{2} & 0 & 0 \\ & & & & \frac{C_{11} - C_{12}}{2} & 0 \\ SYM & & & & & \frac{C_{11} - C_{12}}{2} \end{pmatrix} \quad (\text{B.30})$$

Therefore, an isotropic material can be fully defined given 2 independent parameters.

B.2.5 Compliance and stiffness matrices for orthotropic materials definition given elastic properties

As the studied laminates of this project are made out of orthogonal plies (it is usually the case for common composites), an insight is made into orthotropic materials.

As seen earlier, an orthotropic material stiffness and compliance matrix can be fully defined using 9 independent parameters. These 9 parameters have to be measured in the 3 orthogonal directions which define the matrices and are also directions of symmetry of the material.

By recalling the relations (B.21), (B.23) and (B.25) it is obtained [3]:

$$S_{11} = \frac{1}{E_1} \quad S_{22} = \frac{1}{E_2} \quad S_{33} = \frac{1}{E_3} \quad (\text{B.31})$$

$$S_{12} = -\frac{\nu_{21}}{E_2} = -\frac{\nu_{12}}{E_1} \quad S_{13} = -\frac{\nu_{31}}{E_3} = -\frac{\nu_{13}}{E_1} \quad S_{23} = -\frac{\nu_{32}}{E_3} = -\frac{\nu_{23}}{E_2} \quad (\text{B.32})$$

$$S_{44} = \frac{1}{G_{23}} \quad S_{55} = \frac{1}{G_{13}} \quad S_{66} = \frac{1}{G_{12}} \quad (\text{B.33})$$

$$\bar{\bar{S}} = \begin{pmatrix} \frac{1}{E_1} & -\frac{\nu_{21}}{E_2} & -\frac{\nu_{31}}{E_3} & 0 & 0 & 0 \\ & \frac{1}{E_2} & -\frac{\nu_{32}}{E_3} & 0 & 0 & 0 \\ & & \frac{1}{E_3} & 0 & 0 & 0 \\ & & & \frac{1}{G_{23}} & 0 & 0 \\ & & & & \frac{1}{G_{13}} & 0 \\ \text{SYM} & & & & & \frac{1}{G_{12}} \end{pmatrix} \quad (\text{B.34})$$

Isotropic material particular case

If a material is isotropic the same formulation applies, as any isotropic material is also orthotropic, but it can be greatly simplified as only 2 different properties define the material. As the properties are constant for any direction chosen, there is a single Young's modulus value, Poisson ratio value and shear modulus value for the material [3].

$$E = E_1 = E_2 = E_3 \quad \nu = \nu_{ij} \quad s.t. \quad i, j = \{1, 2, 3\} \quad G = G_{12} = G_{13} = G_{23} = \frac{E}{2 \cdot (1 + \nu)} \quad (\text{B.35})$$

$$\bar{\bar{S}} = \frac{1}{E} \cdot \begin{pmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ & 1 & -\nu & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 2 \cdot (1 + \nu) & 0 & 0 \\ & & & & 2 \cdot (1 + \nu) & 0 \\ \text{SYM} & & & & & 2 \cdot (1 + \nu) \end{pmatrix} \quad (\text{B.36})$$

$$\bar{\bar{C}} = \frac{E}{(1 + \nu) \cdot (1 - 2 \cdot \nu)} \cdot \begin{pmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ & 1 - \nu & \nu & 0 & 0 & 0 \\ & & 1 - \nu & 0 & 0 & 0 \\ & & & \frac{1 - 2 \cdot \nu}{2} & 0 & 0 \\ & & & & \frac{1 - 2 \cdot \nu}{2} & 0 \\ SYM & & & & & \frac{1 - 2 \cdot \nu}{2} \end{pmatrix} \quad (\text{B.37})$$

B.3 Classical Lamination Theory

In this section, the reader can find the full extension of the Classical Lamination Theory section in the Methodology chapter of the Report document. It is advisable to read said chapter to acquire the context of the methodology described here.

B.3.1 Introduction

Classical Lamination Theory (CLT) is a basic constitutive model used to analyze the behaviour of thin-walled or thin-profile solid parts made by stacking laminae of known materials. The set of laminae, called a laminate, acts in the elastic regime and under plane stresses, using a model equivalent to the Euler-Bernoulli theory for prismatic parts. This model is useful for making a first representation of the behaviour of fibre-matrix composite materials organized into stacked laminae forming a laminate that usually has a very small thickness.

References to all the proofs used to vertebrate this theory are found in many of the bibliographic sources [5, 4, 6, 7, 8, 3].

B.3.2 Model limitations

The Classical Lamination Theory (CLT) makes the following assumptions in its formulation.

- The laminate is under a state of plane stress.
- The laminae exhibit elastic behaviour.
- The laminae exhibit linear behaviour.

- The strain distribution follows a linear distribution through the laminate thickness.
- The laminae are perfectly bonded to each other.
- The interlaminar bonds are infinitesimally thin.
- There is no relative slip between the laminae.

In general, the most significant simplification is the elastic and linear behaviour of each lamina. This implies that the fibres of the laminae maintain a constant relationship between stresses and strains, and that plasticity will not occur before rupture or failure.

The next simplification of the model postulates that the laminate as a whole will be sufficiently thin compared to the other two longitudinal dimensions to approximate the assembly as a two-dimensional element under a state of plane stress. This implies that there will be no significant stresses out of the plane tangent (locally, at any point) to the part. Also, the stresses in this same direction will be small enough compared to the others to be neglected without introducing a significant error.

As a consequence of the above assumptions, it can be understood that the material strains must be small, since a material with behaviour close to the ideal elastic and linear range will be only found when enduring small strains. The linear distribution of the strains over the thickness is reasonable since if it were assumed to be of higher order, the increase in precision would not justify the increase in complexity. At the same time, it allows the modelling of the element's bending, which would not be seen if the distribution was constant.

The criteria related to the bonding between laminae are necessary to avoid complicating the model beyond its intended scope. If the laminae are not perfectly bonded, there will be irregularly arranged distortions in the stress and strain fields that are difficult to model. The bonds must be considered of infinitesimal thickness since otherwise, in the context of a thin part, they would probably have to be modelled as an additional lamina. If the laminae allow slippage between them, the problem will move away from the potential elastic treatment, violating the precept of elasticity.

B.3.3 Lamina model in material coordinates

The CLT model starts by evaluating the individual properties of each distinct lamina that will form the laminate. In the first instance, it is convenient to define the stiffness matrix $\overline{\overline{C}}$ (or its inverse, the strain matrix $\overline{\overline{S}}$) according to a coordinate system that will normally coincide with the material geometry or the direction of the mechanical tests. Of the 3 directions of the coordinate axes, one will be normal to the plane that

defines the lamina (along the thickness, the dimension shorter than the others) and the other 2 will be oriented following the principal direction of the material if it has any, intending to obtain the most simplified stiffness matrix possible.

These axes will be called material or lamina axes since they are oriented according to the characteristics of each lamina and can be identified by the subscripts 1-2-3 (in contrast to x-y-z).

In a general anisotropic case (and following Voigt notation) the stiffness matrix of the lamina according to the material axes 1-2-3 will be:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ & & C_{33} & C_{34} & C_{35} & C_{36} \\ & & & C_{44} & C_{45} & C_{46} \\ & & & & C_{55} & C_{56} \\ SYM & & & & & C_{66} \end{pmatrix} \quad (\text{B.38})$$

Since the theory considers a state of plane stress, the matrix degenerates to a smaller matrix which considers the components other from the direction normal to the plane of the lamina (direction 3). The terms that disappear are zero under this assumption and therefore the notation can be contracted.

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{16} \\ C_{21} & C_{22} & C_{26} \\ C_{61} & C_{62} & C_{66} \end{pmatrix} \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{pmatrix} \quad (\text{B.39})$$

Here, the 3rd dimension of the matrix has been labelled 6 for continuity with the general 6x6 matrix. However, it can be labelled 3 as long as it is not confused with the direction normal to the lamina, out of scope.

The stiffness matrix $\bar{\bar{C}}$ and strain matrix $\bar{\bar{S}}$ are, therefore:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{16} \\ & C_{22} & C_{26} \\ SYM & & C_{66} \end{pmatrix} \quad \bar{\bar{S}} = \begin{pmatrix} S_{11} & S_{12} & S_{16} \\ & S_{22} & S_{26} \\ SYM & & S_{66} \end{pmatrix} \quad (\text{B.40})$$

Typically, laminae are not strictly anisotropic but can be considered orthotropic or transversely isotropic (with respect to some direction contained in the plane of the lamina). For these two cases, the stiffness matrix $\bar{\bar{C}}$ and strain matrix $\bar{\bar{S}}$ are:

$$\bar{\bar{C}} = \begin{pmatrix} C_{11} & C_{12} & 0 \\ & C_{22} & 0 \\ SYM & & C_{66} \end{pmatrix} \quad \bar{\bar{S}} = \begin{pmatrix} S_{11} & S_{12} & 0 \\ & S_{22} & 0 \\ SYM & & S_{66} \end{pmatrix} \quad (\text{B.41})$$

B.3.4 Thermal effect in material coordinates

If the effects of thermal expansion or contraction are taken into account, the formulation must include an additional strain term that depends on the temperature. The expansion model is as follows:

$$\varepsilon_{123}^T = \begin{pmatrix} \varepsilon_1^T \\ \varepsilon_2^T \\ \gamma_{12}^T \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ 0 \end{pmatrix} \cdot \Delta T \quad \Delta T = T - T_{ref} \quad (\text{B.42})$$

where ε_{123}^T are the components of the thermal strains in material axes, α_1 and α_2 are the coefficients of thermal expansion in directions 1 and 2, T is the temperature at the point at issue, and T_{ref} is the reference temperature for which there are no thermal strains.

It can be seen that there is no cross-term of the coefficient of thermal expansion since the material axes are principal axes, such that thermal expansion does not cause shear strains ($\gamma_{12}^T = 0$).

Other phenomena such as hygroscopic expansion can be modeled similarly. Specifically, hygroscopic expansion can be calculated following the same structure as the case of thermal expansion but changing the coefficient of thermal expansion α by the hygroscopic expansion coefficient β and the temperature increment term ΔT by the increment of the absorbed moisture weight over the weight of the lamina ΔC .

The total or perceptible strain is the sum, therefore, of the component caused by the internal stresses of the material plus the component of thermal effects. If there were other sources of strain (such as hygroscopic), they could be added at this point.

$$\tilde{\varepsilon} = \bar{S}\tilde{\sigma} + \tilde{\varepsilon}^T \quad (\text{B.43})$$

Isolating the stresses from the expression it is found:

$$\tilde{\sigma} = \bar{C}(\tilde{\varepsilon} - \tilde{\varepsilon}^T) = \bar{C}\tilde{\varepsilon} - \bar{C}\tilde{\varepsilon}^T = \bar{C}\tilde{\varepsilon} - \tilde{\sigma}^T \quad \tilde{\sigma}_{123}^T = \bar{C}_{123} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ 0 \end{pmatrix} \cdot \Delta T \quad (\text{B.44})$$

Therefore, thermal effects can be summarized as additional thermal stresses ($\tilde{\sigma}^T$) that depend on the temperature at each point.

B.3.5 Lamina model in laminate coordinates

Since the laminae that form the laminate must be stacked in different directions to improve the material response in directions other than the longitudinal one, it is convenient to reference the stresses and strains in laminate axes, which will be called x-y-z, and can be associated with the finite element mesh or geometry alignment to facilitate the design or understanding of the model. These axes, however, keep the z-axis parallel to the 3-axis so that it points in the direction normal to the tangent plane at each point, being the direction that does not appear in the plane stress formulation. Therefore, the two coordinate axes (1-2-3 and x-y-z) only differ in rotation about the z or 3 axis. This rotation will be identified with an angle θ between the x and 1 axes (or y and 2 axes), following the positive direction marked by the z or 3 vector.

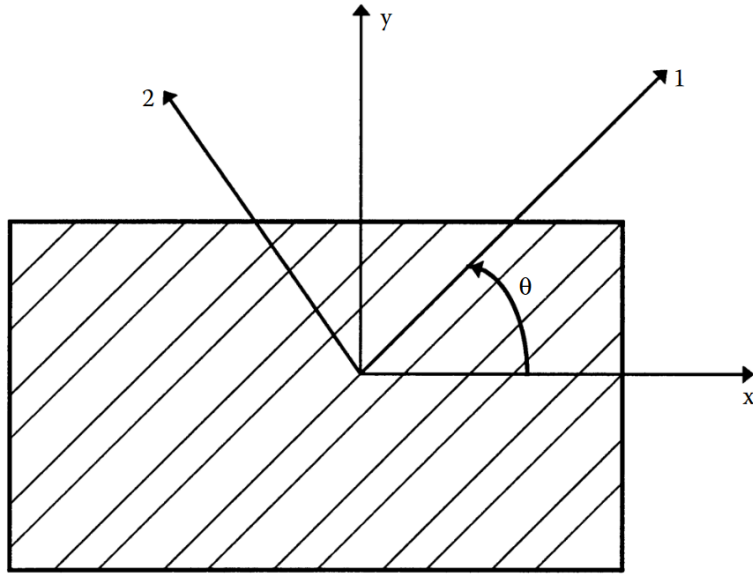


Figure B.1: Graphical exemplification of the coordinate systems of the material or sheet (1-2) and the laminate (x-y) together with the angle θ that separates them. Extracted from [4].

According to the plane stress theory, the stresses and strains are transformed from one coordinate system to another as follows:

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \overline{\overline{T(\theta)}}^{-1} \cdot \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{pmatrix} \quad \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \frac{\gamma_{xy}}{2} \end{pmatrix} = \overline{\overline{T(\theta)}}^{-1} \cdot \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \frac{\gamma_{12}}{2} \end{pmatrix} \quad (\text{B.45})$$

where matrix $\overline{\overline{T(\theta)}}$ is the change-of-basis matrix, function of the angle θ .

$$\overline{\overline{T(\theta)}} = \begin{pmatrix} \cos^2 \theta & \sin^2 \theta & 2 \sin \theta \cos \theta \\ \sin^2 \theta & \cos^2 \theta & -2 \sin \theta \cos \theta \\ -\sin \theta \cos \theta & \sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta \end{pmatrix} \quad (\text{B.46})$$

$$\overline{\overline{T(\theta)}}^{-1} = \begin{pmatrix} \cos^2 \theta & \sin^2 \theta & -2 \sin \theta \cos \theta \\ \sin^2 \theta & \cos^2 \theta & 2 \sin \theta \cos \theta \\ \sin \theta \cos \theta & -\sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta \end{pmatrix} \quad (\text{B.47})$$

The strain vectors must be modified with a special matrix $\overline{\overline{R}}$ so that the last cross term does not appear multiplied by 0.5.

$$\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix} = \overline{\overline{R}} \overline{\overline{T(\theta)}}^{-1} \overline{\overline{R}}^{-1} \cdot \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{pmatrix} \quad (\text{B.48})$$

$$\overline{\overline{R}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix} \quad \overline{\overline{R}}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix} \quad (\text{B.49})$$

$$\overline{\overline{R}} \overline{\overline{T(\theta)}} \overline{\overline{R}}^{-1} = \begin{pmatrix} \cos^2 \theta & \sin^2 \theta & \sin \theta \cos \theta \\ \sin^2 \theta & \cos^2 \theta & -\sin \theta \cos \theta \\ -2 \sin \theta \cos \theta & 2 \sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta \end{pmatrix} \quad (\text{B.50})$$

$$\overline{\overline{R}} \overline{\overline{T(\theta)}}^{-1} \overline{\overline{R}}^{-1} = \begin{pmatrix} \cos^2 \theta & \sin^2 \theta & -\sin \theta \cos \theta \\ \sin^2 \theta & \cos^2 \theta & \sin \theta \cos \theta \\ 2 \sin \theta \cos \theta & -2 \sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta \end{pmatrix} \quad (\text{B.51})$$

With the new laminate coordinate system, the structure of equation (B.44) can be reproduced with a new matrix $\overline{\overline{G}}$ which is the lamina stiffness matrix in the laminate x-y-z coordinates.

$$\tilde{\sigma}_{xyz} = \overline{\overline{G}} (\tilde{\varepsilon}_{xyz} - \tilde{\varepsilon}_{xyz}^T) = \overline{\overline{G}} \tilde{\varepsilon}_{xyz} - \tilde{\sigma}_{xyz}^T \quad \overline{\overline{G}} = \overline{\overline{C}}_{xyz} = \overline{\overline{T}}^{-1} \overline{\overline{C}}_{123} \overline{\overline{R}} \overline{\overline{T}} \overline{\overline{R}}^{-1} \quad (\text{B.52})$$

$$\tilde{\sigma}_{xyz}^T = \bar{\bar{T}}^{-1} \bar{\bar{C}}_{123} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ 0 \end{pmatrix} \cdot \Delta T = \bar{\bar{G}} \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_{xy} \end{pmatrix} \cdot \Delta T = \bar{\bar{G}} \bar{\bar{R}} \bar{\bar{T}}^{-1} \bar{\bar{R}}^{-1} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ 0 \end{pmatrix} \cdot \Delta T \quad (\text{B.53})$$

It can be seen, in the laminate coordinate system, that the influence of thermal expansion in each direction can be determined, where now there may be a non-negative cross term α_{xy} .

B.3.6 Laminate displacement distribution

Classical lamination theory imposes a linear strain profile. In order to define it, it is necessary to locate a reference system on the laminate.

First, it is important to remember that the thickness of the laminate is relatively small to represent plane stresses. For this reason, the laminate can be represented as a two-dimensional surface to which forces and boundary conditions are applied tangentially at each point. This surface, from now on called the reference surface, will normally be located following the midpoint of the laminate thickness (it would be equidistant from the two outer surfaces) but could be defined according to some other criterion. Since the system forces will be applied on the reference plane, it should be located in the most representative position of application of the mentioned forces.

At each point on the reference surface, an intrinsic reference system x-y-z can be established such that the z-axis coincides with the normal direction of the reference plane at that point. Thus, the x and y axes will define a reference plane tangent at each point to the reference surface. In addition, $z = 0$ can be defined as the condition of all the points that form the reference surface. The x and y axes will be aligned following the chosen laminate's global axes. Summarizing, all points on the laminate can be identified by their position (x, y) on the reference surface and by their position normal to the reference surface z.

The displacement vector at a point on the laminate will be called \vec{U} , where U , V and W are the components of the vector \vec{U} on the respective axes x, y, and z.

$$\vec{U} = (U, V, W) \quad (\text{B.54})$$

Considering the position \vec{P} of a generic point P that is contained on the reference surface, its coordinates are characterized by $z = 0$.

$$\vec{P} = (x, y, 0) \quad (\text{B.55})$$

At this point P, its displacements (\vec{U}_P) will have the following structure:

$$\vec{U}_P = (U_P, V_P, W_P) \quad (\text{B.56})$$

Since the displacement (and strain) profile over the laminate thickness is imposed as linear, as the coordinate system has been defined, the linear dependence is precisely on the position on the z-axis, the direction normal to the reference plane.

$$U = A \cdot z + B \quad (\text{B.57})$$

Where U is the displacement of a generic point of the laminate in one of the directions tangent to the reference plane, z corresponds to the position on the z-axis (measured from the reference plane and in the “ascending” direction, or defined according to the x and y axes) and A and B are constant parameters.

Therefore, in this model, B corresponds to the displacement experienced by the projection of the study point on the reference surface along the z-axis. The constant parameter A corresponds to a rotation about the point $z = 0$, at least for small rotations where the described arc of circumference approximates the rectilinear displacement.

$$U(\vec{x}) = U_P + z \cdot \theta_y \quad V(\vec{x}) = V_P - z \cdot \theta_x \quad (\text{B.58})$$

θ_x and θ_y correspond to rotations of the reference plane about the x and y axes, respectively, following their positive directions.

B.3.7 Strain and curvature distribution on the laminate

Knowing the displacement distribution, the strain distribution can be found. The strain ε is defined as:

$$\varepsilon_x = \frac{\partial U}{\partial x} \quad \varepsilon_y = \frac{\partial V}{\partial y} \quad \gamma_{xy} = \frac{\partial U}{\partial y} + \frac{\partial V}{\partial x} \quad (\text{B.59})$$

Therefore, substituting the displacement distribution found, we obtain:

$$\tilde{\varepsilon} = \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial U_P}{\partial x} + z \cdot \frac{\partial \theta_y}{\partial x} \\ \frac{\partial V_P}{\partial y} - z \cdot \frac{\partial \theta_x}{\partial y} \\ \frac{\partial U_P}{\partial y} + \frac{\partial V_P}{\partial x} + z \cdot \left(\frac{\partial \theta_y}{\partial y} - \frac{\partial \theta_x}{\partial x} \right) \end{pmatrix} = \tilde{\varepsilon}^0 - z \cdot \vec{\chi} \quad (\text{B.60})$$

$$\tilde{\varepsilon}^0 = \begin{pmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{pmatrix} = \begin{pmatrix} \frac{\partial U_P}{\partial x} \\ \frac{\partial V_P}{\partial y} \\ \frac{\partial U_P}{\partial y} + \frac{\partial V_P}{\partial x} \end{pmatrix} \quad \vec{\chi} = \begin{pmatrix} \chi_x \\ \chi_y \\ \chi_{xy} \end{pmatrix} = \begin{pmatrix} -\frac{\partial \theta_y}{\partial x} \\ \frac{\partial \theta_x}{\partial y} \\ -\left(\frac{\partial \theta_y}{\partial y} - \frac{\partial \theta_x}{\partial x} \right) \end{pmatrix} \quad (\text{B.61})$$

Where $\tilde{\varepsilon}^0$ has been defined as the vector of mid-plane strains and $\vec{\chi}$ as the vector of reference surface curvatures.

It can be observed that knowing the position of a point with respect to the reference surface and also the strain and curvature of the reference surface, the strain of the point can be calculated.

B.3.8 Laminate constitutive equations

The reduction of the laminate to a two-dimensional reference surface (plate treatment) causes the stresses to be considered as forces per unit length and moments per unit length instead of forces and moments.

The forces per unit length of a lamina k of the N laminae that make up the laminate correspond to the integration over the z -axis of the stresses in the lamina. To obtain the value for the laminate, the contributions of each of the N laminae must be added.

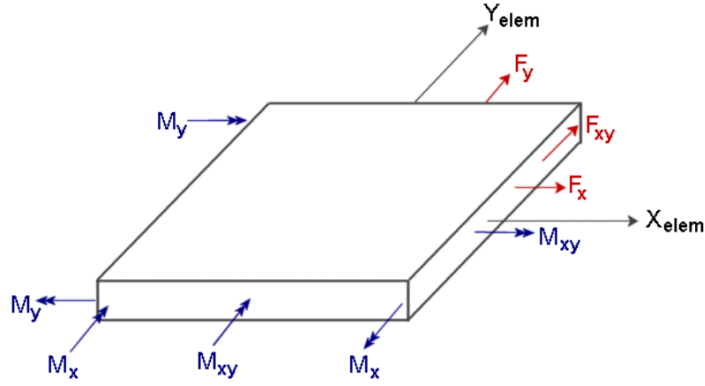


Figure B.2: Diagram with the positive directions of the forces per unit length (F) and the moments per unit length (M) for an element of the laminate. Extracted from [7].

Due to the laminate geometry, the integration limits on the z -axis correspond to the z -position values of the contact surfaces between contiguous laminae. These surfaces can be called interfaces [9]. For a generic lamina k , its contact surfaces with the contiguous laminae are defined as $z = z_k$ for the upper surface and $z = z_{k-1}$ for the lower surface. As a result, the laminate of N laminae has a thickness comprised between $z = z_0$ and $z = z_N$.

$$\begin{pmatrix} N_x \\ N_y \\ N_{xy} \end{pmatrix} = \sum_{k=1}^N \int_{z_{k-1}}^{z_k} \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix}_k dz = \sum_{k=1}^N \int_{z_{k-1}}^{z_k} \overline{\overline{G}}_k \cdot \left[\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix}_k - \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_{xy} \end{pmatrix}_k \cdot \Delta T_k \right] dz \quad (\text{B.62})$$

Where the stress vector of each lamina k has been defined as a function of the strain vector $\tilde{\varepsilon}_k$, the lamina stiffness matrix $\overline{\overline{G}}_k$, the thermal expansion coefficients $\vec{\alpha}_k$ and the increment of temperature ΔT_k . N_x , N_y , N_{xy} are components of the applied forces per unit of length (refer to Figure B.2 where they are labeled as F_x , F_y , F_{xy}).

The stiffness matrix $\overline{\overline{G}}_k$ and the thermal expansion coefficients $\vec{\alpha}_k$ are constant for each lamina k (they do not depend on z inside the lamina), so they can be brought outside the integral as a result.

If the strain vector $\tilde{\varepsilon}_k$ is developed as the linear distribution of equation (B.60), a result is obtained that depends on z (integration variable) and on $\tilde{\varepsilon}^0$ and $\vec{\chi}$, which are functions only of the reference surface and not of the position z . Therefore, the terms can be taken out of the integral.

$$\begin{pmatrix} N_x \\ N_y \\ N_{xy} \end{pmatrix} = \sum_{k=1}^N \bar{\bar{G}}_k \cdot \left[\int_{z_{k-1}}^{z_k} \begin{pmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{pmatrix} dz - \int_{z_{k-1}}^{z_k} z \cdot \begin{pmatrix} \chi_x \\ \chi_y \\ \chi_{xy} \end{pmatrix} dz - \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_{xy} \end{pmatrix}_k \cdot \int_{z_{k-1}}^{z_k} \Delta T_k dz \right] \quad (\text{B.63})$$

For convenience, an average temperature $\Delta T'_k$ is defined that can represent the result of the temperature integral.

$$\Delta T'_k = \frac{\int_{z_{k-1}}^{z_k} \Delta T_k dz}{\int_{z_{k-1}}^{z_k} dz} \quad (\text{B.64})$$

If the temperature is constant along the whole thickness of the lamina, then $\Delta T'_k = \Delta T_k$.

Finally, and in reduced notation, the forces per unit length \vec{N} are:

$$\vec{N} = \sum_{k=1}^N \left(\bar{\bar{G}}_k \cdot \vec{\varepsilon}_k^0 \cdot (z_k - z_{k-1}) - \bar{\bar{G}}_k \cdot \vec{\chi}_k \cdot \frac{(z_k^2 - z_{k-1}^2)}{2} - \bar{\bar{G}}_k \cdot \vec{\alpha}_k \cdot \Delta T'_k \cdot (z_k - z_{k-1}) \right) \quad (\text{B.65})$$

In the case of the moments per unit of length (M_x , M_y , M_{xy}), the methodology is the same but considering the initial z term inside the integral.

$$\begin{pmatrix} M_x \\ M_y \\ M_{xy} \end{pmatrix} = - \sum_{k=1}^N \int_{z_{k-1}}^{z_k} z \cdot \begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix}_k dz = - \sum_{k=1}^N \int_{z_{k-1}}^{z_k} \bar{\bar{G}}_k \cdot \left[\begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix}_k - \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_{xy} \end{pmatrix}_k \cdot \Delta T_k \right] \cdot z dz \quad (\text{B.66})$$

$$\begin{pmatrix} M_x \\ M_y \\ M_{xy} \end{pmatrix} = \sum_{k=1}^N \bar{\bar{G}}_k \cdot \left[- \int_{z_{k-1}}^{z_k} \begin{pmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{pmatrix} \cdot z \, dz + \int_{z_{k-1}}^{z_k} z^2 \cdot \begin{pmatrix} \chi_x \\ \chi_y \\ \chi_{xy} \end{pmatrix} \, dz + \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_{xy} \end{pmatrix}_k \cdot \int_{z_{k-1}}^{z_k} z \cdot \Delta T_k \, dz \right] \quad (\text{B.67})$$

Now a temperature $\Delta T_k''$ is defined such that can represent the result of the temperature integral.

$$\Delta T_k'' = \frac{\int_{z_{k-1}}^{z_k} z \cdot \Delta T_k \, dz}{\int_{z_{k-1}}^{z_k} z \, dz} \quad (\text{B.68})$$

If the temperature is constant along the whole thickness of the lamina, then $\Delta T_k'' = \Delta T_k$.

$$\vec{M} = \sum_{k=1}^N \left(-\bar{\bar{G}}_k \cdot \vec{\varepsilon}_k^0 \cdot \frac{(z_k^2 - z_{k-1}^2)}{2} + \bar{\bar{G}}_k \cdot \vec{\chi}_k \cdot \frac{(z_k^3 - z_{k-1}^3)}{3} + \bar{\bar{G}}_k \cdot \vec{\alpha}_k \cdot \Delta T_k'' \cdot \frac{(z_k^2 - z_{k-1}^2)}{2} \right) \quad (\text{B.69})$$

The temperature map is considered known, and therefore the thermal contribution can be grouped as a force and moment per unit length.

$$\vec{N}^T = \sum_{k=1}^N \bar{\bar{G}}_k \cdot \vec{\alpha}_k \cdot \Delta T_k'' \cdot (z_k - z_{k-1}) \quad \vec{M}^T = \sum_{k=1}^N \bar{\bar{G}}_k \cdot \vec{\alpha}_k \cdot \Delta T_k'' \cdot \frac{(z_k^2 - z_{k-1}^2)}{2} \quad (\text{B.70})$$

Finally, if the two expressions are grouped in matrix form, the system results in:

$$\begin{pmatrix} \vec{N} \\ \vec{M} \end{pmatrix} = \begin{bmatrix} \bar{\bar{A}} & \bar{\bar{B}} \\ \bar{\bar{B}} & \bar{\bar{D}} \end{bmatrix} \begin{pmatrix} \vec{\varepsilon}^0 \\ \vec{\chi} \end{pmatrix} + \begin{pmatrix} -\vec{N}^T \\ \vec{M}^T \end{pmatrix} \quad (\text{B.71})$$

The submatrixes which compose the general matrix are called the membrane matrix ($\bar{\bar{A}}$), the membrane coupling matrix ($\bar{\bar{B}}$) and the bending matrix ($\bar{\bar{D}}$).

$$\bar{\bar{A}} = \sum_{k=1}^N \bar{\bar{G}}_k \cdot (z_k - z_{k-1}) \quad \bar{\bar{B}} = -\frac{1}{2} \sum_{k=1}^N \bar{\bar{G}}_k \cdot (z_k^2 - z_{k-1}^2) \quad \bar{\bar{D}} = \frac{1}{3} \sum_{k=1}^N \bar{\bar{G}}_k \cdot (z_k^3 - z_{k-1}^3) \quad (\text{B.72})$$

B.4 Transverse Shear Theory

In this section, the reader can find the full extension of the Transverse Shear Theory section in the Methodology chapter of the Report document. It is advisable to read said chapter to acquire the context of the methodology described here.

B.4.1 Introduction

The Transverse Shear Theory (TST) is a complementary addition to the CLT used to solve the problem involving the assumption of plane stress in the laminate. Imposing plane stress means no stress is considered in the direction normal to the laminate plane and therefore no out-of-plane forces or loads are taken into account.

MSC Nastran uses this theory to obtain an approximate value of the transverse shear stress elements τ_{xz} and τ_{yz} , and ultimately obtain an equivalent constitutive relation to the CLT (A , B and D matrices) for the transverse shear model.

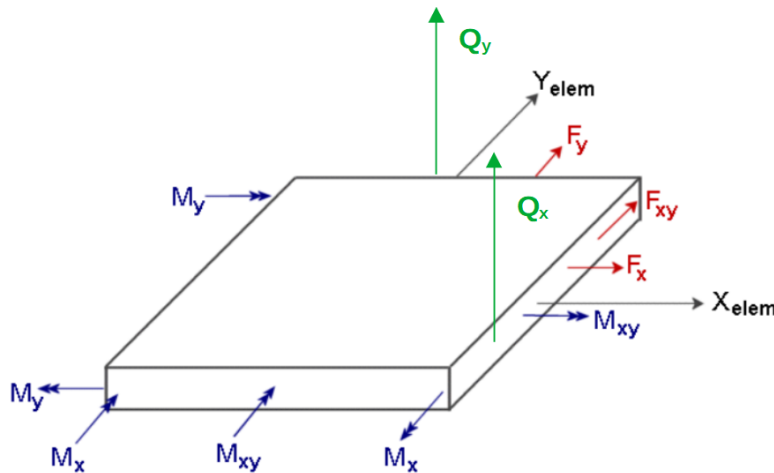


Figure B.3: Diagram with the positive directions of the forces per unit length (F , Q) and the moments per unit length (M) for an element of the laminate. Extracted from [7] and modified.

$$\vec{Q} = \overline{\overline{H}} \vec{\gamma} \rightarrow \begin{pmatrix} Q_x \\ Q_y \end{pmatrix} = \begin{pmatrix} H_{xx} & H_{xy} \\ H_{xy} & H_{yy} \end{pmatrix} \begin{pmatrix} \gamma_{xz} \\ \gamma_{yz} \end{pmatrix} \quad (\text{B.73})$$

where $\overline{\overline{H}}$ is called the transverse shear stiffness matrix, $\vec{\gamma}$ contains the out-of-plane shear strains and \vec{Q} is

the vector with the two components of force per unit length load acting in the z axis. See Figure B.3 for reference.

In [5] the same transverse shear stiffness matrix is defined following the first-order shear deformation theory (FSDT). Its expression needs a shear correction coefficient K dependent on lamina properties and lamination scheme [3]. This is because in reality the shear stresses and strains are not constant as FSDT suggest but have at least a quadratic distribution profile along the thickness of a general laminate [3]. The shear correction coefficient K corrects the first-order output to the mean stress value of the real non-constant distribution. The real distribution has to be solved using alternative methods as it is the solution of a system of second-order partial differential equations which does not have a general closed-form solution [8].

With the use of equation (B.73), one can have an approximation of the mean value of transverse shear stresses on a point of the laminate and a corresponding artificial mean value strain which couples well with the general loads Q_x and Q_y applied at the laminate, but that might not be enough to predict the failure point of the laminate if it is significantly dependent on the shear stresses. Specifically, there is a phenomenon called delamination where two adjacent plies' bonding fails and a surface slip becomes possible between them. The so-called interlaminar or intralaminar stresses usually define whether or not this failure is achieved, and they are precisely the transverse shear stress measured at the interface between implicated neighbour laminae.

As a result, knowing the values of the transverse shear stress at each interface between plies is the aim of the TST at MSC Nastran, which will be exposed in the following sections. Note that the formulation will not include the steps to transition from the resulting transverse shear stress values along the laminate thickness to the formulation of the transverse shear stiffness matrix of equation (B.73) because it will not be used in this project, where interest is put on establishing real intralaminar stresses rather than an equivalent unified transverse shear strain for the laminate.

The source of the TST formulation can be found at [6] or at [7].

B.4.2 Model limitations

The first limitations, shared with the CLT, are the following. Crucially, the plane stress consideration is no longer valid.

- The laminae exhibit elastic behaviour.

- The laminae exhibit linear behaviour.
- The laminae are perfectly bonded to each other.
- The interlaminar bonds are infinitesimally thin.
- There is no relative slip between the laminae.

The remaining limitation of the theory is fundamentally one:

- The transverse shear deformation phenomenon for each in-plane axis (1 and 2 or x and y depending on the case) is uncoupled from the other.

The previous statement is not true in general except for the case of only one orthotropic lamina in its oriented material axis (1 and 2), where the shear strain deformation in the fibre direction is considered to be independent of the transverse direction deformation (the ply local transverse shear stiffness matrix is diagonal).

In practice, the following procedures are based on finding a suitable differential equation, arising from the Kirchhoff-Love plate theory, which relates transverse shear loads Q to shear stresses τ_{xz} or τ_{yz} , and then simplify the equation enough to be analytically solvable. This simplification involves disregarding coupling terms with the other axis transverse shear parameters, therefore uncoupling the phenomenon altogether.

B.4.3 Balance of forces in the longitudinal direction

To approximate the values of the shear stresses inside the laminate, it is necessary to consider equilibrium situations on representative elements of the laminae, taking into account the shear forces and stresses acting on them.

First, a force equilibrium is performed in the longitudinal direction (1 or 2) of the material. In this case, the calculation is performed for direction 1, and the same procedure is followed for direction 2 but changing the associated values. The normal coordinate axis 3 is usually called z to improve understanding. Although they belong to different coordinate systems, both share the same direction and sense.

The force equilibrium in the 1-axis on an infinitesimal volume element of the lamina yields the following result that relates the shear stresses to the normal stresses in the 1-direction:

$$\frac{\partial \sigma_1}{\partial x_1} + \frac{\partial \tau_{z1}}{\partial z} = 0 \quad (\text{B.74})$$

From now on, variables x_1 and x_2 are used to describe the spacial coordinates of the ply local coordinate system with axis 1 (x_1 , spacial first axis) and axis 2 (x_2 , spacial second axis), just as it is done with the laminate coordinate system with x and y .

B.4.4 Relation of the bending moment to the normal stresses

In order to expand the normal stress term in the 1-axis, the Kirchhoff-Love plate theory for simple bending is used. This theory can be considered analogous to the Euler-Bernoulli theory for prismatic beams applied to plates, which are elements with one dimension much smaller than the others. This is very suitable for the study of laminates. The assumptions made in this theory are, in general, elastic behaviour and small deformations.

The definition of the moment per unit length (M_1) with respect to the rotation of the plate at a point (w) in the direction of the moment is:

$$M_1 = -D_1 \cdot \left(\frac{\partial^2 w}{\partial x_1^2} + \nu_{12} \cdot \frac{\partial^2 w}{\partial x_2^2} \right) \quad (\text{B.75})$$

The analogous definition for the case of normal stresses in the 1-direction (σ_1) and considering the origin of the z-axis at the middle of the lamina is:

$$\sigma_1 = \frac{12 \cdot (z_n - z)}{T^3} \cdot D_1 \cdot \left(\frac{\partial^2 w}{\partial x_1^2} + \nu_{12} \cdot \frac{\partial^2 w}{\partial x_2^2} \right) \quad (\text{B.76})$$

Where z_n is the position of the neutral axis (the directional bending center) [6], T is the thickness of the sheet (about the z-axis) and D is called the stiffness at bending.

The combination of the two previous expressions results in the expression of the normal stresses as a function of the equivalent moment and other material or geometrical parameters.

$$\sigma_1 = \frac{\left[\frac{E_1}{1 - \nu_{12} \cdot \nu_{21}} \right] \cdot (z_n - z)}{D_1} \cdot M_1 \quad D_1 = \frac{E_1}{1 - \nu_{12} \cdot \nu_{21}} \cdot \frac{T^3}{12} \quad (\text{B.77})$$

Where E_1 is Young's modulus in direction 1, ν_{12} is the Poisson's ratio with respect to directions 1 and 2, z is the position for which the stress σ_1 is obtained, z_n is the position of the neutral axis (such that the normal stresses are zero), T is the thickness of the sheet (about the z-axis) and D is called flexural rigidity, which

depends on the material of the sheet and the inertia of the section, which in this case is calculated taking into account that the origin of the z -axis is in the middle of the section.

Substituting the previous equation into equation (B.74), the shear stresses can be found as a function of the position z of the point and the applied equivalent moment M_1 .

$$\frac{\partial \tau_{z1}}{\partial z} = -\frac{\partial \sigma_1}{\partial x_1} = -\frac{\left[\frac{E_1}{1-\nu_{12}\nu_{21}} \right] \cdot (z_n - z)}{D_1} \cdot \frac{\partial M_1}{\partial x_1} \quad (\text{B.78})$$

B.4.5 Balance of moments in the plate element

To relate the shear stresses to the shear forces, the latter can be related to the moments per unit length in the 1-direction by considering the equilibrium of moments per unit length of a flat plate in the 2-direction.

Ignoring the effect of the twisting moment M_{12} and the shear force Q_2 as they are supposed to be independent (uncoupled phenomena), the equilibrium results in:

$$Q_1 + \frac{\partial M_1}{\partial x_1} = 0 \quad (\text{B.79})$$

The problem here is that the laminate is formed by several plies of different orientations, meaning the direction of the axis 1 of a ply is not necessarily the same as the direction of the axis 1 of another ply in the laminate. However, the shear force Q and the moment M are common for every ply as they are applied on the theoretical laminate surface affecting every ply at the same time. Then, equation (B.79) can only be valid for every ply in the laminate if expressed in global coordinates (x and y axis).

$$Q_x + \frac{\partial M_x}{\partial x} = 0 \quad (\text{B.80})$$

To be able to substitute $\frac{\partial M_x}{\partial x}$ into equation (B.78) and finally relate the shear forces Q_x to the shear stresses τ_{z1} we now have to express equation (B.78) in global coordinates (x and y axis), then having shear stresses τ_{zx} .

$$\frac{\partial \tau_{zx}}{\partial z} = -\frac{\partial \sigma_x}{\partial x} = -\frac{\left[\frac{E_x}{1-\nu_{xy}\nu_{yx}} \right] \cdot (z_n - z)}{D_x} \cdot \frac{\partial M_x}{\partial x} \quad (\text{B.81})$$

Now substituting both equations, the result is:

$$\frac{\partial \tau_{zx}}{\partial z} = -\frac{\partial \sigma_x}{\partial x} = \frac{\left[\frac{E_x}{1-\nu_{xy}\nu_{yx}} \right] \cdot (z_n - z)}{D_x} \cdot Q_x \quad (\text{B.82})$$

B.4.6 Integration of tangential stresses in a lamina

The shear stresses can be isolated by integrating equation (B.82) over the thickness of a lamina, assuming that the material properties (E_x , ν_{yx} and ν_{xy}) are constant in z :

$$\tau_{zx} = \tau_{xz} = \int_{lamina} \frac{\left[\frac{E_x}{1-\nu_{xy}\nu_{yx}} \right] \cdot (z_n - z)}{D_x} \cdot Q_x dz \quad (\text{B.83})$$

$$\tau_{xz} = \frac{\left[\frac{E_x}{1-\nu_{xy}\nu_{yx}} \right] \cdot Q_x}{D_x} \cdot \left(z_n \cdot z - \frac{z^2}{2} \right) + C \quad (\text{B.84})$$

The integration of the previous equation generates a constant C that can be determined by the boundary conditions of each lamina separately. Therefore, the equation will be generalized for each lamina of the laminate and specifically at the position of the bonding planes between laminae. This is because, firstly, it is at this point where the values of the unknown constants C can be determined. Secondly, the value of the stresses between the laminae is also the value of the interlaminar shear stresses, which can be an interesting factor when calculating the failure by delamination.

At this point, it can be useful to simplify the expression of τ_{xz} to be a function of elements on the constitutive equations matrices of laminae and laminate where possible.

$$\frac{E_x}{1 - \nu_{xy} \cdot \nu_{yx}} = G_{xx} = G_{11} \quad (\text{B.85})$$

where \overline{G} is the corresponding lamina stiffness matrix in the lamina x-y-z coordinates.

Alternatively, the constant D is defined to be a flexural rigidity acting on the laminate as a whole. Therefore it has to be treated as an average flexural rigidity of the plies, or else simply the flexural rigidity of the whole laminate in laminate coordinates. By recalling its definition a relation with the bending matrix of the laminate can be found.

$$D_x = \frac{E_x}{1 - \nu_{xy} \cdot \nu_{yx}} \cdot \frac{T^3}{12} = D_{xx} = D_{11} \quad (\text{B.86})$$

where D_{xx} (or D_{11} also) is the first row and first column element of the bending matrix $\overline{\overline{D}}$ of the laminate (not to be confused with the actual defined constant D_x).

As a result (B.82) in laminate coordinates ends up being:

$$\tau_{xz} = \frac{G_{xx} \cdot Q_x}{D_{xx}} \cdot \left(z_n \cdot z - \frac{z^2}{2} \right) + C \quad (\text{B.87})$$

For a lamina i of the laminate, located between the positions $z = z_{i-1}$ and $z = z_i$, the stress $\tau_{1z i}$ is the interlaminar stress at $z = z_i$ between laminae i and $i + 1$ in the 1-direction. On the other hand, the constant C_i is unique and valid for each lamina i evaluated.

$$\tau_{xz i} = \frac{[G_{xx}]_i \cdot Q_x}{D_{xx}} \cdot \left(z_n \cdot z_i - \frac{z_i^2}{2} \right) + C_i \quad (\text{B.88})$$

$$C_i = -\frac{[G_{xx}]_i \cdot Q_x}{D_{xx}} \cdot \left(z_n \cdot z_{i-1} - \frac{z_{i-1}^2}{2} \right) + \tau_{xz i-1} \quad (\text{B.89})$$

Solving the system of equations for all the interlaminar stresses in the laminate involves equating the interlaminar stress of two adjacent laminae to find a constant C and then using the value of the constant to find the next stress. Starting from the bottom position of the laminate $z = z_0$, lamina $i = 1$ has no adjacent lamina, and therefore the stress at this point is $\tau_{1z 0} = 0$. Knowing this data, C_1 can be calculated and then $\tau_{1z 1}$ and so on until $\tau_{1z N}$ is found, the value of which must be zero since the boundary condition is the same as on the other side of the laminate.

$$\tau_{xz 0} = 0 \quad \tau_{xz N} = 0 \quad (\text{B.90})$$

B.4.7 Bureau Veritas method

At the same Bureau Veritas document from which procedures in micromechanics have been extracted (see the Report document, the Methodology chapter) an algorithm to provide approximate interlaminar stress values is proposed [9]. The transverse shear calculation method is not the same as the procedure is different from that used in MSC Nastran, but similar results are obtained. A closer comparison between the two methods reveals some similarities as the presence of constants computed using previous ply properties or the dependence on a thickness position z and a quadratic thickness position z^2 , mainly resembling in structure. Both versions are implemented in the Python code in the Annexes of the present document and its results can

be compared for a particular laminate.

Appendix C

Project schedule

An assessment of the time duration of the tasks in the project was made at an early stage and included in the Project Charter document of the project. Here, the Schedule section appearing on the Project Charter is included to provide some insight into the time management involved in this Bachelor's Final Thesis.

Because of the early stage at which the assessment was written, some objectives or tasks did not survive the final project. The most important change has been the substitution of the task called "User materials" where custom methods would be implemented in MSC Nastran to the final design of a laminate for the Singular Aircraft wing beam. Also, the Manufacturing Process chapter was not expected at the time, as the possibility of learning the processes employed at Singular Aircraft was offered later. Apart from that, the planning is a decent approximation of the time burden of the tasks.

C.1 Introduction to the schedule

There are 15 full weeks until the written memory presentation deadline (June 19th), starting to count from the week that starts on March 4th (week 1).

From these 15 weeks, the final 2 weeks are kept free as a security margin. Ideally, they will be used for revision and giving the advisors time to provide final feedback.

Tasks are defined following a predetermined structure:

1. **Title of the task** - *Duration of the task in weeks* - Dep.: [Task numbers of the depending tasks]
Description of the task.

In this case, the task would be identified as task number 1 (number next to the title). If the task would need another previous task to be completed before beginning it, the number of said previous task would be in the dependency task number section.

C.2 Task description

1. **Familiarization with MSC Nastran** - 3 weeks - Dep.: None

Acquisition of knowledge about the inner workings of MSC Nastran as a finite element solver and its default capabilities regarding constitutive material models. In parallel, this knowledge has to be applied to obtain results. This acquired skill must be enough to properly execute a structural analysis using the necessary constitutive relations developed.

2. **Composite micromechanics models and theories** - 2 weeks - Dep.: None

Documenting used mathematical models and theories needed to obtain an equivalent homogeneous anisotropic material from the real heterogeneous combination of constituents conforming the composite material.

3. **Experimental data on fiber and matrix** - 1 week - Dep.: None

Acquiring mechanical data of fiber and matrix materials used in the studied composite.

4. **Experimental data on laminae** - 1 week - Dep.: None

Acquiring mechanical data about similar laminates tested experimentally.

5. **Legal constraints on composite design** - 2 weeks - Dep.: None

Documenting the existing laws and regulations affecting composite material design on the aeronautical unmanned vehicle sector, where Singular Aircraft's Flyox airplane operates.

6. **General introduction paragraphs on the written report** - 2 weeks - Dep.: None

Writing general introductory sections which do not depend on final results like objective, justification...

7. **Project charter** - 1 week - Dep.: None

Writing and submitting the project charter

8. First results Nastran - 1 week - Dep.: 1

Obtaining results from studied composites using simple Nastran laminate models already implemented by default.

9. Laminate constitutive models and theories - 2 weeks - Dep.: None

Documenting used mathematical models and theories needed to model shell-like laminates in MSC Nastran from separate anisotropic laminae.

10. Experimental data on failure - 1 week - Dep.: 8

Acquiring data on similar laminate fracture from literature.

11. Validation of the model - 1 week - Dep.: 8

Broad picture of results and its error in front of experimental data to assess improvements to be made in User Materials.

12. Failure models and theories - 2 weeks - Dep.: 11

Documenting material failure modeling methods implemented in MSC Nastran .

13. User materials - 3 weeks - Dep.: 11

Big family of tasks that include familiarization of necessary type of user material in Nastran, search in literature for suitable models to include in user material definition and implementation in Nastran.

14. Environmental impact and biocomposite convenience - 2 weeks - Dep.: None

Explanation about the environmental impact of the present thesis and the composite materials studied on its application environment. Explanation of the difference and convenience of studied biocomposites with respect to conventional composites.

15. Documentation of used User Materials - 2 weeks - Dep.: 12, 13

Documenting in the report the type of model implemented as a user material, characteristics, objectives, etc.

16. Comparison of obtained results with experimental sources - 2 weeks - Dep.: 13

Final comparison between obtained results and interesting experimental data from other sources in order to validate the results.

17. **Selection and presentation of relevant results** - 2 weeks - Dep.: 13, 17

Selecting and presenting graphical results and data of interest to include in the report.

18. **Discussion and conclusions on obtained data** - 2 weeks - Dep.: 13, 17

Description and discussion of obtained data.

19. **Budget** - 1 week - Dep.: 13

Writing the budget (assignment).

20. **Finalizing writing** - 1 week - Dep.: 16

Writing conclusions and unfinished parts overall.

C.3 Gantt chart

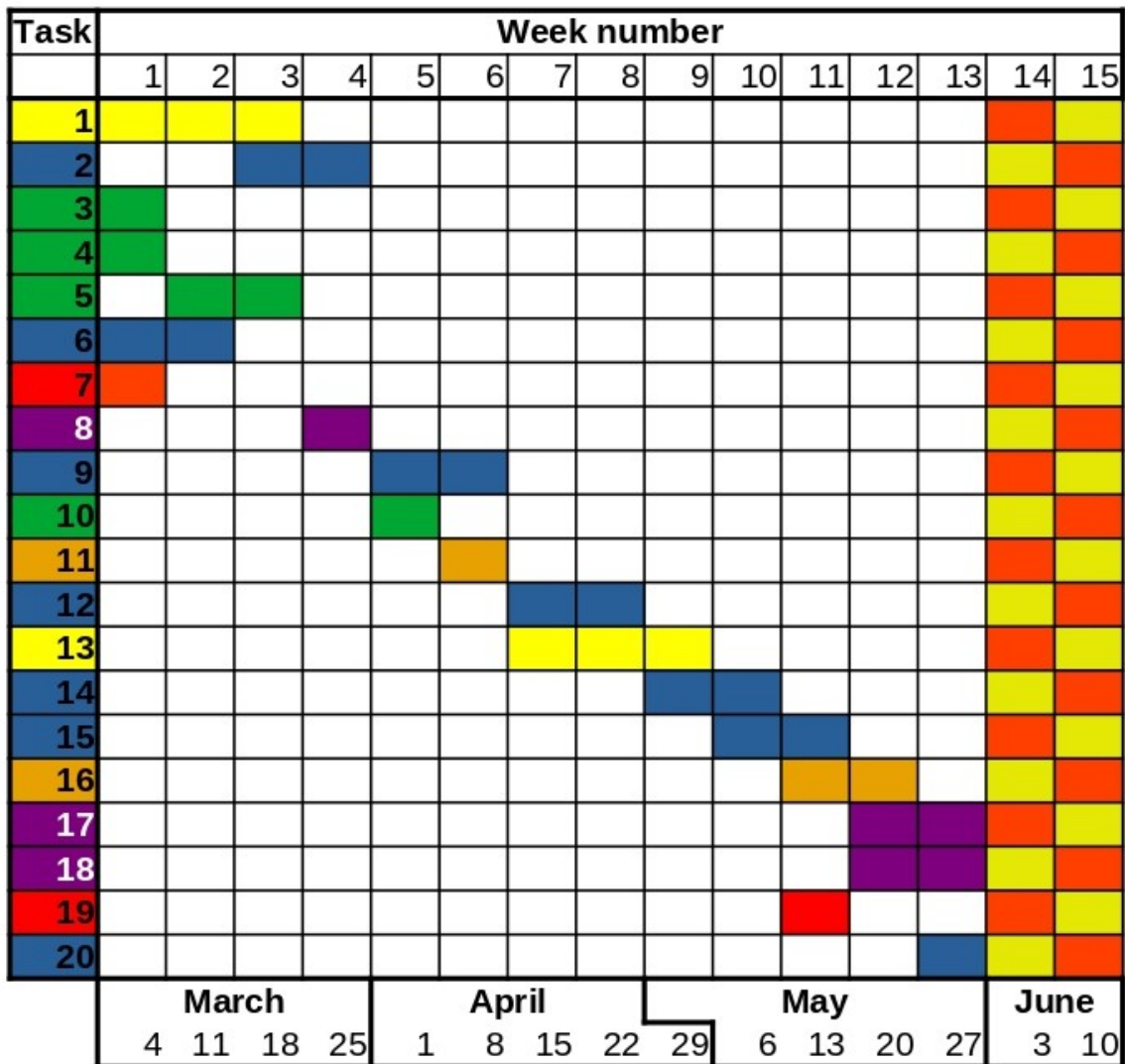


Figure C.1: Gantt chart. Task number vs week number.

C.3.1 Color coding of the Gantt chart

Yellow: Informatic Nastran interface related work

Blue: Written memory writing

Green: Literature data search and classification

Red: Explicit deliverables

Purple: Internal delivery of numerical results

Orange: Analysis and/or comparison of results and literature data

Red-Yellow: 2 final weeks of margin for delays

As a general rule, the tasks that generate dependencies are purple and orange given their nature of analysis of previous work and decision on where to advance. Yellow tasks also generate dependencies as they end up reporting practical results from MSC Nastran. The other ones have a considerable degree of independence.

Appendix D

Auxiliary Python Code

In this chapter, the Python written auxiliary code is included. Each necessary file is shown in a different section with the executable being the Main Code section (the longest section by extension).

D.1 Bureau Veritas method parameter dictionary

This file defines a Python dictionary containing the specific parameters needed by the Bureau Veritas method of obtaining ply properties from individual components. Its author is Alfons Borràs, and it is used at his Master's Thesis [10].

The file name can be found commented at the beginning of the code.

```
# -*- coding: utf-8 -*-  
"""  
  
Created on Fri Mar  3 12:37:45 2023  
  
@author: Alfons Borràs  
"""  
  
#FILE NAME:  BV_materials_dictionary.py
```

```

#Dictionary of BV Coefficients to be applied with UD fibers and
↪ different fiber materials

#individual dictionaries for each material
E_Glass_c={'CUD1':1.0, 'CUD2':0.8, 'CUD12':0.9, 'CUDnu': 0.9}
R_Glass_c={'CUD1':0.9, 'CUD2':1.2, 'CUD12':1.2, 'CUDnu': 0.9}
Carbon_HS_c={'CUD1':1, 'CUD2':0.7, 'CUD12':0.9, 'CUDnu': 0.8}
Carbon_IM_c={'CUD1':0.85, 'CUD2':0.8, 'CUD12':0.9, 'CUDnu': 0.75}
Carbon_HM_c={'CUD1':0.9, 'CUD2':0.85, 'CUD12':1, 'CUDnu': 0.7}
Para_amid_c={'CUD1':0.95, 'CUD2':0.9, 'CUD12':0.55, 'CUDnu': 0.9}

#Main UD Coefficients dictionary
CoefDict={
'E-Glass': E_Glass_c,
'R-Glass': R_Glass_c,
'Carbon-HS': Carbon_HS_c,
'Carbon-IM': Carbon_IM_c,
'Carbon-HM': Carbon_HM_c,
'Para-amid': Para_amid_c
}

#Material data charts

#Individual elastic and physical properties for each fiber material

E_Glass_BV={'Density':2.57, 'Ef0':73100, 'Ef90':73100, 'Gf': 30000,
↪ 'nuf0': 0.238, 'nuf90':0.238}
R_Glass_BV={'Density':2.52, 'Ef0':86000, 'Ef90':86000, 'Gf': 34600,
↪ 'nuf0': 0.2, 'nuf90':0.2}
Carbon_HS_BV={'Density':1.79, 'Ef0':238000, 'Ef90':15000, 'Gf': 50000,
↪ 'nuf0': 0.3, 'nuf90':0.02}
Carbon_IM_BV={'Density':1.75, 'Ef0':350000, 'Ef90':10000, 'Gf': 35000,
↪ 'nuf0': 0.32, 'nuf90':0.01}

```

```
Carbon_HM_BV={'Density':1.88, 'Ef0':410000, 'Ef90':13800, 'Gf': 27000,
↪ 'nuf0': 0.35, 'nuf90':0.01}
```

```
Para_amid_BV={'Density':1.45, 'Ef0':129000, 'Ef90':5400, 'Gf': 12000,
↪ 'nuf0': 0.38, 'nuf90':0.015}
```

```
#Main fiber material dictionary
```

```
FiberDict={
'E-Glass': E_Glass_BV,
'R-Glass': R_Glass_BV,
'Carbon-HS': Carbon_HS_BV,
'Carbon-IM': Carbon_IM_BV,
'Carbon-HM': Carbon_HM_BV,
'Para-amid': Para_amid_BV
}
```

```
#Individual breaking strains for fibers
```

```
E_Glass_UD_st={'epsbrt1':2.7, 'epsbrt2':0.53, 'epsbrc1':1.80,
↪ 'epsbrc2':1.55, 'gammabr12':1.80,
           'gammabr13':1.80, 'gammabr23':2.50, 'gammaIL2':1.80,
           ↪ 'gammaIL1':2.50
}
```

```
R_Glass_UD_st={'epsbrt1':3.1, 'epsbrt2':0.44, 'epsbrc1':1.80,
↪ 'epsbrc2':1.10, 'gammabr12':1.50,
           'gammabr13':1.50, 'gammabr23':1.80, 'gammaIL2':1.50,
           ↪ 'gammaIL1':1.80
}
```

```
Carbon_HS_UD_st={'epsbrt1':1.20, 'epsbrt2':1.0, 'epsbrc1':0.85,
↪ 'epsbrc2':2.30, 'gammabr12':1.60,
           'gammabr13':1.60, 'gammabr23':1.90, 'gammaIL2':1.60,
           ↪ 'gammaIL1':1.90
}
```

```
Carbon_IM_UD_st={'epsbrt1':1.15, 'epsbrt2':0.8, 'epsbrc1':0.65,  
↪ 'epsbrc2':2.30, 'gammabr12':1.70,  
    'gammabr13':1.70, 'gammabr23':1.85, 'gammaIL2':1.70,  
    ↪ 'gammaIL1':1.85  
    }  
Carbon_HM_UD_st={'epsbrt1':0.7, 'epsbrt2':0.5, 'epsbrc1':0.45,  
↪ 'epsbrc2':2.10, 'gammabr12':1.80,  
    'gammabr13':1.80, 'gammabr23':1.80, 'gammaIL2':1.80,  
    ↪ 'gammaIL1':1.80  
    }  
Para_amid_UD_st={'epsbrt1':1.70, 'epsbrt2':0.8, 'epsbrc1':0.35,  
↪ 'epsbrc2':2.00, 'gammabr12':2.0,  
    'gammabr13':2.0, 'gammabr23':2.90, 'gammaIL2':2.0,  
    ↪ 'gammaIL1':2.90  
    }  
E_Glass_WR_st={'epsbrt1':1.8, 'epsbrt2':1.8, 'epsbrc1':1.80,  
↪ 'epsbrc2':1.80, 'gammabr12':1.50,  
    'gammabr13':1.80, 'gammabr23':1.80, 'gammaIL2':1.80,  
    ↪ 'gammaIL1':1.80  
    }  
R_Glass_WR_st={'epsbrt1':2.30, 'epsbrt2':2.30, 'epsbrc1':2.50,  
↪ 'epsbrc2':2.50, 'gammabr12':1.50,  
    'gammabr13':1.80, 'gammabr23':1.80, 'gammaIL2':1.80,  
    ↪ 'gammaIL1':1.80  
    }  
Carbon_HS_WR_st={'epsbrt1':1.00, 'epsbrt2':1.0, 'epsbrc1':0.85,  
↪ 'epsbrc2':0.85, 'gammabr12':1.55,  
    'gammabr13':1.55, 'gammabr23':1.55, 'gammaIL2':1.55,  
    ↪ 'gammaIL1':1.55  
    }  
Carbon_IM_WR_st={'epsbrt1':0.80, 'epsbrt2':0.8, 'epsbrc1':0.80,  
↪ 'epsbrc2':0.80, 'gammabr12':1.60,  
    'gammabr13':1.60, 'gammabr23':1.60, 'gammaIL2':1.60,  
    ↪ 'gammaIL1':1.60
```

```

    }
Carbon_HM_WR_st={'epsbrt1':0.45, 'epsbrt2':0.45, 'epsbrc1':0.50,
↪ 'epsbrc2':0.50, 'gammabr12':1.85,
    'gammabr13':1.85, 'gammabr23':1.85, 'gammaIL2':1.85,
↪ 'gammaIL1':1.85
    }
Para_amid_WR_st={'epsbrt1':1.40, 'epsbrt2':1.40, 'epsbrc1':0.42,
↪ 'epsbrc2':0.42, 'gammabr12':2.30,
    'gammabr13':2.90, 'gammabr23':2.90, 'gammaIL2':2.90,
↪ 'gammaIL1':2.90
    }
E_Glass_CSM_st={'epsbrt1':1.55, 'epsbrt2':1.55, 'epsbrc1':1.55,
↪ 'epsbrc2':1.55, 'gammabr12':2.0,
    'gammabr13':2.15, 'gammabr23':2.15, 'gammaIL2':2.15,
↪ 'gammaIL1':2.15
    }
R_Glass_CSM_st={'epsbrt1':0, 'epsbrt2':0, 'epsbrc1':0, 'epsbrc2':0,
↪ 'gammabr12':0,
    'gammabr13':0, 'gammabr23':0, 'gammaIL2':0, 'gammaIL1':0
    }
Carbon_HS_CSM_st={'epsbrt1':0, 'epsbrt2':0, 'epsbrc1':0, 'epsbrc2':0,
↪ 'gammabr12':0,
    'gammabr13':0, 'gammabr23':0, 'gammaIL2':0, 'gammaIL1':0
    }
Carbon_IM_CSM_st={'epsbrt1':0, 'epsbrt2':0, 'epsbrc1':0, 'epsbrc2':0,
↪ 'gammabr12':0,
    'gammabr13':0, 'gammabr23':0, 'gammaIL2':0, 'gammaIL1':0
    }
Carbon_HM_CSM_st={'epsbrt1':0, 'epsbrt2':0, 'epsbrc1':0, 'epsbrc2':0,
↪ 'gammabr12':0,
    'gammabr13':0, 'gammabr23':0, 'gammaIL2':0, 'gammaIL1':0
    }
Para_amid_CSM_st={'epsbrt1':0, 'epsbrt2':0, 'epsbrc1':0, 'epsbrc2':0,
↪ 'gammabr12':0,

```

```

        'gammabr13':0, 'gammabr23':0, 'gammaIL2':0, 'gammaIL1':0
    }

#dictionaries per type of material
E_Glass_fab={'UD':E_Glass_UD_st, 'WR':E_Glass_WR_st,
    ↪ 'CSM':E_Glass_CSM_st}
R_Glass_fab={'UD':R_Glass_UD_st, 'WR':R_Glass_WR_st,
    ↪ 'CSM':R_Glass_CSM_st}
Carbon_HS_fab={'UD':Carbon_HS_UD_st, 'WR':Carbon_HS_WR_st,
    ↪ 'CSM':Carbon_HS_CSM_st}
Carbon_IM_fab={'UD':Carbon_IM_UD_st, 'WR':Carbon_IM_WR_st,
    ↪ 'CSM':Carbon_IM_CSM_st}
Carbon_HM_fab={'UD':Carbon_HM_UD_st, 'WR':Carbon_HM_WR_st,
    ↪ 'CSM':Carbon_HM_CSM_st}
Para_amid_fab={'UD':Para_amid_UD_st, 'WR':Para_amid_WR_st,
    ↪ 'CSM':Para_amid_CSM_st}

#distionary of breaking strains
StrainDict={
    'E-Glass': E_Glass_fab,
    'R-Glass': R_Glass_fab,
    'Carbon-HS': Carbon_HS_fab,
    'Carbon-IM': Carbon_IM_fab,
    'Carbon-HM': Carbon_HM_fab,
    'Para-amid': Para_amid_fab
}

#Individual material foer each matrix material

Polyester_BV={'Density':1.20, 'Er':3550, 'Gr': 1350, 'nur': 0.38,
    ↪ 'Coefres': 0.8}
Vinilester_BV={'Density':1.10, 'Er':3350, 'Gr': 1400, 'nur': 0.26,
    ↪ 'Coefres': 0.9 }

```

```
Epoxy_BV={'Density':1.25, 'Er':3100, 'Gr': 1500, 'nur': 0.39,  
↪ 'Coefres': 1}
```

```
#Main matrix material dictionary
```

```
MatrixDict={  
    'Polyester':Polyester_BV,  
    'Vinilester':Vinilester_BV,  
    'Epoxy':Epoxy_BV  
}
```

```
if __name__ == "__main__":
```

```
    print(type(FiberDict['E-Glass']['Ef0']))  
    print(FiberDict['E-Glass']['Ef0'])  
    print(type(MatrixDict['Polyester']['Er']))  
    print(MatrixDict['Polyester']['Er'])
```

D.2 Composite components dictionary

This Python dictionary stores the information related to each component of the plies used.

The file name can be found commented at the beginning of the code.

```
"""
```

```
June 2024
```

```
@author: Pol Roca Iruela
```

```
"""
```

```
#FILE NAME: MaterialsCustom1.py
```

```

def DensityVolMix(d1, d2, v1): #Compute general density of two
↳ components (resin and hardener) given their densities and volume
↳ fractions
    return v1*d1 + (1-v1)*d2

def DensityMassMix(d1, d2, w1): #Compute general density of two
↳ components (resin and hardener) given their densities and mass
↳ fractions
    return 1/(w1/d1 + (1-w1)/d2)

#### CUSTOM MATERIAL DICTIONARY ####

#List of different Matrices

#Matrix code: GP33_SD4772_A
#Sicom GreenPoxo GP33 / SD 4772    Ambient + 24 hrs 40 °C
GP33_1 = {
'Name': "GP33_1",
'Mat_inp': "EP", #Epoxy -- EP    Polyester -- P    Vinilester -- V
'd_r': DensityVolMix(1.159, 0.93, 3/4), #Density [g/cm^3] Empty for
↳ default value
'Er': 3200, #Young Modulus [MPa] Empty for default value
'Gr': "ISO", #Shear Modulus [MPa] Empty for default value. "ISO"
↳ computes isotropic value Gr = Er/(2*(1+nur))
'nur': '' #Poisson ratio Empty for default value
}

#Matrix code: GP33_SD4773_A
#Sicom GreenPoxo GP33 / SD 4773    Ambient + 24 hrs 40 °C
GP33_2 = {
'Name': "GP33_2",
'Mat_inp': "EP", #Epoxy -- EP    Polyester -- P    Vinilester -- V
'd_r': DensityVolMix(1.159, 0.98, 100/132), #Density [g/cm^3] Empty for
↳ default value

```

```

'Er': 3450, #Young Modulus [MPa] Empty for default value
'Gr': "ISO", #Shear Modulus [MPa] Empty for default value. "ISO"
↪ computes isotropic value  $Gr = Er / (2 * (1 + \nu_r))$ 
'nur': '' #Poisson ratio Empty for default value
}

#Matrix code: PVC Foam HR80 (Core material)
#Dummy matrix, isotropic material
PVC_HR80 = {
'Name': "PVCHR80",
'Mat_inp': "PVCF", #Epoxy -- EP      Polyester -- P      Vinilester -- V
↪ PVC Foam -- PVCF
'd_r': 80/1000, #Density [g/cm3] Empty for default value
'Er': 66, #Young Modulus [MPa] Empty for default value
'Gr': 30, #Shear Modulus [MPa] Empty for default value. "ISO" computes
↪ isotropic value  $Gr = Er / (2 * (1 + \nu_r))$ 
'nur': 0.08 #Poisson ratio Empty for default value
}
#PVC_HR80['nur'] = 0.5*PVC_HR80['Er']/PVC_HR80['Gr'] -1 #Poisson ratio
↪ Empty for default value

#Matrix code: Al 6082-T6 (Isotropic mat)
#Dummy matrix, isotropic material
Al6082_mat = {
'Name': "Al6082",
'Mat_inp': "PVCF", #Epoxy -- EP      Polyester -- P      Vinilester -- V
↪ PVC Foam -- PVCF
'd_r': 2.7, #Density [g/cm3] Empty for default value
'Er': 70000, #Young Modulus [MPa] Empty for default value
'Gr': 26.32, #Shear Modulus [MPa] Empty for default value. "ISO"
↪ computes isotropic value  $Gr = Er / (2 * (1 + \nu_r))$ 
'nur': 0.33 #Poisson ratio Empty for default value
}

```

```

#PVC_HR80['nur'] = 0.5*PVC_HR80['Er']/PVC_HR80['Gr'] -1 #Poisson ratio
↳ Empty for default value

#Matrix code: Al 6068-T6 (Isotropic mat)
#Dummy matrix, isotropic material
Al6068_mat = {
'Name': "Al6068",
'Mat_inp': "PVCF", #Epoxy -- EP      Polyester -- P      Vinilester -- V
↳ PVC Foam -- PVCF
'd_r': 2.73, #Density [g/cm^3] Empty for default value
'Er': 70000, #Young Modulus [MPa] Empty for default value
'Gr': 26.32, #Shear Modulus [MPa] Empty for default value. "ISO"
↳ computes isotropic value Gr = Er/(2*(1+nur))
'nur': 0.33 #Poisson ratio Empty for default value
}
#PVC_HR80['nur'] = 0.5*PVC_HR80['Er']/PVC_HR80['Gr'] -1 #Poisson ratio
↳ Empty for default value

#List of different Fibers

#Fiber code: BXE446
#BXE 446 +-45° (ONE layer)
EGlass_446 = {
'Name': "EGlass",
'Fib_inp': "E", #E-Glass ="E"    R-Glass ="R"    Carbon HS="CHS"    Carbon
↳ IM="CIM"    Carbon HM="CHM"    Para-Amid="PA"
'Fab': "UD", #Fabric Type: UniDirectional Fabric = "UD"    Woven Roven =
↳ "WR"    Chopped Strand Mat = "CSM"
'Ceq': 217*2/441, #Woven Coefficient. Mass fraction of 0° fiber with
↳ respect to all fibers. Only used for Woven Roven or when it is
↳ defined on the Fiber, aka. Aux_Fab = "WR"
'Pf': 217, #Surface density [g/m^2]

```

```

'Pextra': (5+2)/2, #Additional surface density [g/m^2] due to
↳ additional weight that could exist in form of (for instance)
↳ stabilization threads
'd_f': '', #Density [g/cm^3] Empty for default value
'Ef0': '', #Young Modulus, fiber direction [MPa] Empty for default
↳ value
'Ef90': "SAME", #Young Modulus, transversal direction [MPa] Empty for
↳ default value. "SAME" assigns Ef90 = Ef0
'Gf': "ISO", #Shear Modulus [MPa] Empty for default value. "ISO"
↳ computes isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and  $\nu f$  are
↳ the mean values between  $Ef0 / \nu f0$  and  $Ef90 / \nu f90$ 
'nuf0': '', #Poisson ratio, fiber direction Empty for default value
'nuf90': "SAME", #Poisson ratio, transversal direction Empty for
↳ default value. "SAME" assigns  $\nu f90 = \nu f0$ 
}

#Fiber code: BXE300
#BXE 300 +-45° (ONE layer)
EGlass_300 = {
'Name': "EGlass",
'Fib_inp': "E", #E-Glass ="E" R-Glass ="R" Carbon HS="CHS" Carbon
↳ IM="CIM" Carbon HM="CHM" Para-Amid="PA"
'Fab': "UD", #Fabric Type: UniDirectional Fabric = "UD" Woven Roven =
↳ "WR" Chopped Strand Mat = "CSM"
'Ceq': 142*2/295, #Woven Coefficient. Mass fraction of 0° fiber with
↳ respect to all fibers. Only used for Woven Roven or when it is
↳ defined on the Fiber, aka. Aux_Fab = "WR"
'Pf': 142, #Surface density [g/m^2]
'Pextra': (5+6)/2, #Additional surface density [g/m^2] due to
↳ additional weight that could exist in form of (for instance)
↳ stabilization threads
'd_f': '', #Density [g/cm^3] Empty for default value
'Ef0': '', #Young Modulus, fiber direction [MPa] Empty for default
↳ value

```

```

'Ef90': "SAME", #Young Modulus, transversal direction [MPa] Empty for
↳ default value. "SAME" assigns Ef90 = Ef0
'Gf': "ISO", #Shear Modulus [MPa] Empty for default value. "ISO"
↳ computes isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and nuf are
↳ the mean values between Ef0/nuf0 and Ef90/nuf90
'nuf0': '', #Poisson ratio, fiber direction Empty for default value
'nuf90': "SAME", #Poisson ratio, transversal direction Empty for
↳ default value. "SAME" assigns nuf90 = nuf0
}

#Fiber code: VV29
#VV-29 Silionne plain
EGlass_130 = {
'Name': "EGlass",
'Fib_inp': "E", #E-Glass ="E" R-Glass ="R" Carbon HS="CHS" Carbon
↳ IM="CIM" Carbon HM="CHM" Para-Amid="PA"
'Fab': "WR", #Fabric Type: UniDirectional Fabric = "UD" Woven Roven =
↳ "WR" Chopped Strand Mat = "CSM"
'Ceq': 0.5, #Woven Coefficient. Mass fraction of 0° fiber with respect
↳ to all fibers. Only used for Woven Roven or when it is defined on
↳ the Fiber, aka. Aux_Fab = "WR"
'Pf': 130, #Surface density [g/m^2]
'Pextra': 0, #Additional surface density [g/m^2] due to additional
↳ weight that could exist in form of (for instance) stabilization
↳ threads
'd_f': '', #Density [g/cm^3] Empty for default value
'Ef0': '', #Young Modulus, fiber direction [MPa] Empty for default
↳ value
'Ef90': '', #Young Modulus, transversal direction [MPa] Empty for
↳ default value. "SAME" assigns Ef90 = Ef0
'Gf': '', #Shear Modulus [MPa] Empty for default value. "ISO" computes
↳ isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and nuf are the mean
↳ values between Ef0/nuf0 and Ef90/nuf90
'nuf0': '', #Poisson ratio, fiber direction Empty for default value

```

```

'nuf90': '', #Poisson ratio, transversal direction Empty for default
↪ value. "SAME" assigns nuf90 = nuf0
}

#Fiber code: Dummy fibre for isotropic cores
#HR80 PVC Foam
PVC_Core = {
'Name': "PVC",
'Fib_inp': "PVC", #E-Glass ="E" R-Glass ="R" Carbon HS="CHS" Carbon
↪ IM="CIM" Carbon HM="CHM" Para-Amid="PA"
'Fab': "PVC Foam", #Fabric Type: UniDirectional Fabric = "UD" Woven
↪ Roven = "WR" Chopped Strand Mat = "CSM"
'Ceq': 0, #Woven Coefficient. Mass fraction of 0° fiber with respect to
↪ all fibers. Only used for Woven Roven or when it is defined on the
↪ Fiber, aka. Aux_Fab = "WR"
'Pf': PVC_HR80['d_r']*1e4*3/10, #Surface density [g/m^2]
'Pextra': 0, #Additional surface density [g/m^2] due to additional
↪ weight that could exist in form of (for instance) stabilization
↪ threads
'd_f': PVC_HR80['d_r'], #Density [g/cm^3] Empty for default value
'Ef0': PVC_HR80['Er'], #Young Modulus, fiber direction [MPa] Empty for
↪ default value
'Ef90': PVC_HR80['Er'], #Young Modulus, transversal direction [MPa]
↪ Empty for default value. "SAME" assigns Ef90 = Ef0
'Gf': PVC_HR80['Gr'], #Shear Modulus [MPa] Empty for default value.
↪ "ISO" computes isotropic value  $Gf = Ef / (2 * (1 + nuf))$  where  $Ef$  and  $nuf$ 
↪ are the mean values between  $Ef0/nuf0$  and  $Ef90/nuf90$ 
'nuf0': PVC_HR80['nur'], #Poisson ratio, fiber direction Empty for
↪ default value
'nuf90': PVC_HR80['nur'], #Poisson ratio, transversal direction Empty
↪ for default value. "SAME" assigns nuf90 = nuf0
}

#Fiber code: Dummy fibre for isotropic cores

```

```

#Al 6082-T6 (Isotropic mat)
Al6082_dummy = {
'Name': "Aluminum",
'Fib_inp': "PVC", #E-Glass ="E"  R-Glass ="R"  Carbon HS="CHS"  Carbon
↪ IM="CIM"  Carbon HM="CHM"  Para-Amid="PA"
'Fab': "PVC Foam", #Fabric Type: UniDirectional Fabric = "UD"  Woven
↪ Roven = "WR"  Chopped Strand Mat = "CSM"
'Ceq': 0, #Woven Coefficient. Mass fraction of 0° fiber with respect to
↪ all fibers. Only used for Woven Roven or when it is defined on the
↪ Fiber, aka. Aux_Fab = "WR"
'Pf': Al6082_mat['d_r']*1e4*2.5/10, #Surface density [g/m^2]
'Pextra': 0, #Additional surface density [g/m^2] due to additional
↪ weight that could exist in form of (for instance) stabilization
↪ threads
'd_f': Al6082_mat['d_r'], #Density [g/cm^3] Empty for default value
'Ef0': Al6082_mat['Er'], #Young Modulus, fiber direction [MPa] Empty
↪ for default value
'Ef90': Al6082_mat['Er'], #Young Modulus, transversal direction [MPa]
↪ Empty for default value. "SAME" assigns Ef90 = Ef0
'Gf': Al6082_mat['Gr'], #Shear Modulus [MPa] Empty for default value.
↪ "ISO" computes isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and  $\nu f$ 
↪ are the mean values between Ef0/ $\nu f0$  and Ef90/ $\nu f90$ 
'nuf0': Al6082_mat['nur'], #Poisson ratio, fiber direction Empty for
↪ default value
'nuf90': Al6082_mat['nur'], #Poisson ratio, transversal direction Empty
↪ for default value. "SAME" assigns nuf90 = nuf0
}

```

#Fiber code: Dummy fibre for isotropic cores

#Al 6068-T6 (Isotropic mat)

```

Al6068_dummy = {
'Name': "Aluminum",
'Fib_inp': "PVC", #E-Glass ="E"  R-Glass ="R"  Carbon HS="CHS"  Carbon
↪ IM="CIM"  Carbon HM="CHM"  Para-Amid="PA"

```

```

'Fab': "PVC Foam", #Fabric Type: UniDirectional Fabric = "UD"    Woven
↪ Roven = "WR"    Chopped Strand Mat = "CSM"
'Ceq': 0, #Woven Coefficient. Mass fraction of 0° fiber with respect to
↪ all fibers. Only used for Woven Roven or when it is defined on the
↪ Fiber, aka. Aux_Fab = "WR"
'Pf': Al6068_mat['d_r']*1e4*2.5/10, #Surface density [g/m^2]
'Pextra': 0, #Additional surface density [g/m^2] due to additional
↪ weight that could exist in form of (for instance) stabilization
↪ threads
'd_f': Al6068_mat['d_r'], #Density [g/cm^3] Empty for default value
'Ef0': Al6068_mat['Er'], #Young Modulus, fiber direction [MPa] Empty
↪ for default value
'Ef90': Al6068_mat['Er'], #Young Modulus, transversal direction [MPa]
↪ Empty for default value. "SAME" assigns Ef90 = Ef0
'Gf': Al6068_mat['Gr'], #Shear Modulus [MPa] Empty for default value.
↪ "ISO" computes isotropic value  $Gf = Ef / (2 * (1 + \nu_f))$  where Ef and  $\nu_f$ 
↪ are the mean values between Ef0/ $\nu_f0$  and Ef90/ $\nu_f90$ 
'nuf0': Al6068_mat['nur'], #Poisson ratio, fiber direction Empty for
↪ default value
'nuf90': Al6068_mat['nur'], #Poisson ratio, transversal direction Empty
↪ for default value. "SAME" assigns nuf90 = nuf0
}

```

#Fiber code: Carbon_520_UD from GV 520 U

```

Carbon_520 = {
'Name': "CFib-HS",
'Fib_inp': "CHS", #E-Glass = "E"    R-Glass = "R"    Carbon HS="CHS"    Carbon
↪ IM="CIM"    Carbon HM="CHM"    Para-Amid="PA"
'Fab': "UD", #Fabric Type: UniDirectional Fabric = "UD"    Woven Roven =
↪ "WR"    Chopped Strand Mat = "CSM"
'Ceq': 0.92, #Woven Coefficient. Mass fraction of 0° fiber with respect
↪ to all fibers. Only used for Woven Roven or when it is defined on
↪ the Fiber, aka. Aux_Fab = "WR"
'Pf': 520*0.92, #Surface density [g/m^2]

```

```

'Pextra': 520*0.08, #Additional surface density [g/m^2] due to
↳ additional weight that could exist in form of (for instance)
↳ stabilization threads
'd_f': '', #Density [g/cm^3] Empty for default value
'Ef0': '', #Young Modulus, fiber direction [MPa] Empty for default
↳ value
'Ef90': '', #Young Modulus, transversal direction [MPa] Empty for
↳ default value. "SAME" assigns Ef90 = Ef0
'Gf': '', #Shear Modulus [MPa] Empty for default value. "ISO" computes
↳ isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and  $\nu f$  are the mean
↳ values between  $Ef0 / \nu f0$  and  $Ef90 / \nu f90$ 
'\nu f0': '', #Poisson ratio, fiber direction Empty for default value
'\nu f90': '', #Poisson ratio, transversal direction Empty for default
↳ value. "SAME" assigns  $\nu f90 = \nu f0$ 
}

#Fiber code: Carbon_220_UD from GV 220 U
Carbon_220 = {
'Name': "CFib-HS",
'Fib_inp': "CHS", #E-Glass ="E" R-Glass ="R" Carbon HS="CHS" Carbon
↳ IM="CIM" Carbon HM="CHM" Para-Amid="PA"
'Fab': "UD", #Fabric Type: UniDirectional Fabric = "UD" Woven Roven =
↳ "WR" Chopped Strand Mat = "CSM"
'Ceq': 0.88, #Woven Coefficient. Mass fraction of 0° fiber with respect
↳ to all fibers. Only used for Woven Roven or when it is defined on
↳ the Fiber, aka. Aux_Fab = "WR"
'Pf': 220*0.88, #Surface density [g/m^2]
'Pextra': 220*0.12, #Additional surface density [g/m^2] due to
↳ additional weight that could exist in form of (for instance)
↳ stabilization threads
'd_f': '', #Density [g/cm^3] Empty for default value
'Ef0': '', #Young Modulus, fiber direction [MPa] Empty for default
↳ value

```

```
'Ef90': '', #Young Modulus, transversal direction [MPa] Empty for
↳ default value. "SAME" assigns Ef90 = Ef0
'Gf': '', #Shear Modulus [MPa] Empty for default value. "ISO" computes
↳ isotropic value  $Gf = Ef / (2 * (1 + \nu f))$  where Ef and nuf are the mean
↳ values between Ef0/nuf0 and Ef90/nuf90
'nuf0': '', #Poisson ratio, fiber direction Empty for default value
'nuf90': '', #Poisson ratio, transversal direction Empty for default
↳ value. "SAME" assigns nuf90 = nuf0
}
```

D.3 Composite plies dictionary

This Python dictionary stores the information related to each type of ply defined. It uses the information stored at the component dictionary.

The file name can be found commented at the beginning of the code.

```
"""
June 2024
@author: Pol Roca Iruela
"""

#FILE NAME: LaminatesCustom1.py

from MaterialsCustom1 import *

#List of different Laminaes

#Laminae code: GP_BX446_A
Name = "P_UD_1" #Ply code

Aux_Fibre = EGlass_446 #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber
```

```

Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↳ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↳ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↳ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':
    print("No fabric type defined on the laminae or the fabric
↳ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
    print("No Woven Coefficient defined on the laminae or the fabric
↳ characteristics. Ceq = ''. ")
    exit()
P_UD_1 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↳ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↳ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↳ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↳ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↳ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

#Laminae code: GP_BX300_A
Name = "P_UD_2" #Ply code

```

```

Aux_Fibre = EGlass_300  #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber
Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↪ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↪ Fabric = "UD"    Woven Roven = "WR"    Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↪ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab'] != '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab'] == '':
    print("No fabric type defined on the laminae or the fabric
    ↪ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] != '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] == '':
    print("No Woven Coefficient defined on the laminae or the fabric
    ↪ characteristics. Ceq = ''. ")
    exit()
P_UD_2 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↪ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↪ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↪ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']}

```

```

}

#Laminae code: GP_VV29_A
Name = "P_WR_1" #Ply code

Aux_Fibre = EGlass_130 #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber
Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↳ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↳ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↳ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab'] != '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab'] == '':
    print("No fabric type defined on the laminae or the fabric
↳ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] != '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] == '':
    print("No Woven Coefficient defined on the laminae or the fabric
↳ characteristics. Ceq = ''. ")
    exit()
P_WR_1 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↳ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↳ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↳ 'nur':Aux_Matrix['nur'],

```

```

'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

```

```
#Laminae code: Core PVC foam HR80
```

```
Name = "Core1" #Ply code
```

```
Aux_Fibre = PVC_Core #Fiber and matrix individual codes
```

```
Aux_Matrix = PVC_HR80
```

```
Aux_Vf = '' #Volume Fraction of fiber
```

```
Aux_Mf = 0 #Mass Fraction of fiber. Leave empty if Volume Fraction has
```

```
↪ a value and viceversa
```

```
Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
```

```
↪ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
```

```
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
```

```
↪ it is defined on the Fiber, aka. Aux_Fab = "WR"
```

```
if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
```

```
    Aux_Fab = Aux_Fibre['Fab']
```

```
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':
```

```
    print("No fabric type defined on the laminae or the fabric
```

```
    ↪ characteristics. Fab = '. ")
```

```
    exit()
```

```
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
```

```
    Aux_Ceq = Aux_Fibre['Ceq']
```

```
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
```

```
    print("No Woven Coefficient defined on the laminae or the fabric
```

```
    ↪ characteristics. Ceq = '. ")
```

```
    exit()
```

```
Core1 = {
```

```

'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↪ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↪ 'Fib_inp': 'VOID', 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↪ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'sgBrT':2.0, 'sgBrC': 1.6, 'sgBrS':1.2,
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

```

```
#Laminae code: Aluminum 6082-T6 isotropic material ply
```

```
Name = "Al6082" #Ply code
```

```
Aux_Fibre = Al6082_dummy #Fiber and matrix individual codes
```

```
Aux_Matrix = Al6082_mat
```

```
Aux_Vf = '' #Volume Fraction of fiber
```

```
Aux_Mf = 0 #Mass Fraction of fiber. Leave empty if Volume Fraction has
↪ a value and viceversa
```

```
Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
```

```
↪ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
```

```
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
```

```
↪ it is defined on the Fiber, aka. Aux_Fab = "WR"
```

```
if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
```

```
    Aux_Fab = Aux_Fibre['Fab']
```

```
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':
```

```
    print("No fabric type defined on the laminae or the fabric
```

```
    ↪ characteristics. Fab = '. ")
```

```
    exit()
```

```
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
```

```
    Aux_Ceq = Aux_Fibre['Ceq']
```

```

elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
    print("No Woven Coefficient defined on the laminae or the fabric
    ↪ characteristics. Ceq = ''.")
    exit()
Al6082 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↪ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↪ 'Fib_inp':'VOID', 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↪ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'sgBrT':2.0, 'sgBrC': 1.6, 'sgBrS':1.2,
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

#Laminae code: Aluminum 6068-T6 isotropic material ply
Name = "Al6068" #Ply code

Aux_Fibre = Al6068_dummy #Fiber and matrix individual codes
Aux_Matrix = Al6068_mat
Aux_Vf = '' #Volume Fraction of fiber
Aux_Mf = 0 #Mass Fraction of fiber. Leave empty if Volume Fraction has
↪ a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↪ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↪ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':

```

```

    print("No fabric type defined on the laminae or the fabric
    ↪ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
    print("No Woven Coefficient defined on the laminae or the fabric
    ↪ characteristics. Ceq = ''. ")
    exit()
Al6068 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↪ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↪ 'Fib_inp':'VOID', 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↪ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'sgBrT':2.0, 'sgBrC': 1.6, 'sgBrS':1.2,
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

#Laminae code: GV 520 U
Name = "P_CU_1" #Ply code

Aux_Fibre = Carbon_520 #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber
Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↪ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↪ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"

```

```

Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↳ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':
    print("No fabric type defined on the laminae or the fabric
↳ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
    print("No Woven Coefficient defined on the laminae or the fabric
↳ characteristics. Ceq = ''. ")
    exit()
P_CU_1 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↳ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↳ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↳ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↳ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↳ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

#Laminae code: GV 220 U
Name = "P_CU_2" #Ply code

Aux_Fibre = Carbon_220 #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber

```

```

Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↳ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↳ Fabric = "UD" Woven Roven = "WR" Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↳ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab']!= '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab']== '':
    print("No fabric type defined on the laminae or the fabric
↳ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']!= '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq']== '':
    print("No Woven Coefficient defined on the laminae or the fabric
↳ characteristics. Ceq = ''. ")
    exit()
P_CU_2 = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↳ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↳ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↳ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↳ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↳ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']
}

#Laminae code: L03
Name = "P_WR_2" #Ply code

```

```

Aux_Fibre = EGlass_130  #Fiber and matrix individual codes
Aux_Matrix = GP33_1
Aux_Vf = '' #Volume Fraction of fiber
Aux_Mf = 0.65 #Mass Fraction of fiber. Leave empty if Volume Fraction
↪ has a value and viceversa

Aux_Fab = '' #Empty to choose fiber defined value. UniDirectional
↪ Fabric = "UD"    Woven Roven = "WR"    Chopped Strand Mat = "CSM"
Aux_Ceq = '' #Woven Coefficient. Only non-empty for Woven Roven or when
↪ it is defined on the Fiber, aka. Aux_Fab = "WR"

if Aux_Fab == '' and Aux_Fibre['Fab'] != '':
    Aux_Fab = Aux_Fibre['Fab']
elif Aux_Fab == '' and Aux_Fibre['Fab'] == '':
    print("No fabric type defined on the laminae or the fabric
    ↪ characteristics. Fab = ''. ")
    exit()
if Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] != '':
    Aux_Ceq = Aux_Fibre['Ceq']
elif Aux_Fab == "WR" and Aux_Ceq == '' and Aux_Fibre['Ceq'] == '':
    print("No Woven Coefficient defined on the laminae or the fabric
    ↪ characteristics. Ceq = ''. ")
    exit()
GP_VV29_B = {
'Fab': Aux_Fab, 'Ceq':Aux_Ceq, 'Vf':Aux_Vf, 'Mf':Aux_Mf,
↪ 'Pf':Aux_Fibre['Pf'], 'Pextra':Aux_Fibre['Pextra'],
↪ 'Fib_inp':Aux_Fibre['Fib_inp'], 'Mat_inp':Aux_Matrix['Mat_inp'],
'd_r':Aux_Matrix['d_r'], 'Er':Aux_Matrix['Er'], 'Gr':Aux_Matrix['Gr'],
↪ 'nur':Aux_Matrix['nur'],
'd_f':Aux_Fibre['d_f'], 'Ef0':Aux_Fibre['Ef0'],
↪ 'Ef90':Aux_Fibre['Ef90'], 'Gf':Aux_Fibre['Gf'],
↪ 'nuf0':Aux_Fibre['nuf0'], 'nuf90':Aux_Fibre['nuf90'],
'Name': Name, 'FibName':Aux_Fibre['Name'], 'MatName':Aux_Matrix['Name']}

```

```

}

#List of laminae codes
LamDict ={
'P_UD_1':P_UD_1,
'P_UD_2':P_UD_2,
'P_WR_1':P_WR_1,
'Core1':Core1,
'Al6068':Al6068,
'Al6082':Al6082,
'P_CU_1':P_CU_1,
'P_CU_2':P_CU_2
}

```

D.4 Composite laminates dictionary

This Python dictionary stores the information related to each type of laminate defined. It uses the information stored at the ply dictionary.

The file name can be found commented at the beginning of the code.

```

"""
June 2024
@author: Pol Roca Iruela
"""

#FILE NAME: LaminatesDesign1.py

L1 = {'Name':'L1', 'NameStack':['P_WR_1', 'P_UD_1', 'P_UD_1',
↪ 'P_WR_1'], 'AngleStack':[0, -45, 45, 0], 'AngleU': 'deg'}
L2 = {'Name':'L2', 'NameStack':['P_WR_1', 'P_UD_2', 'P_UD_2',
↪ 'P_WR_1'], 'AngleStack':[0, -45, 45, 0], 'AngleU': 'deg'}

```

```

L3 = {'Name':'L3', 'NameStack':['P_WR_1', 'Core1', 'P_UD_1', 'P_UD_1',
↪ 'P_WR_1'], 'AngleStack':[0, 0, -45, 45, 0], 'AngleU': 'deg'}
L4 = {'Name':'L4', 'NameStack':['P_WR_1', 'Core1', 'P_UD_2', 'P_UD_2',
↪ 'P_WR_1'], 'AngleStack':[0, 0, -45, 45, 0], 'AngleU': 'deg'}
L5 = {'Name':'L5', 'NameStack':['P_WR_1', 'P_UD_2', 'P_UD_2', 'Core1',
↪ 'P_UD_2', 'P_UD_2', 'P_WR_1'], 'AngleStack':[0, 45, -45, 0, -45,
↪ 45, 0], 'AngleU': 'deg'}
TEST = {'Name':'TEST', 'NameStack':['P_UD_2', 'P_UD_2'],
↪ 'AngleStack':[0, 0], 'AngleU': 'deg'}

AL68 = {'Name':'AL68', 'NameStack':['AL6068'], 'AngleStack':[0],
↪ 'AngleU': 'deg'}
AL82 = {'Name':'AL82', 'NameStack':['AL6082'], 'AngleStack':[0],
↪ 'AngleU': 'deg'}

#LC1 = {'Name':'LC1', 'NameStack':['P_CU_1', 'P_CU_1', 'P_CU_1'],
↪ 'AngleStack':[0, 90, 0], 'AngleU': 'deg'}
LC1 = {'Name':'LC1', 'NameStack':['P_CU_1'], 'AngleStack':[0],
↪ 'AngleU': 'deg'}
LC2 = {'Name':'LC1', 'NameStack':['P_CU_2'], 'AngleStack':[0],
↪ 'AngleU': 'deg'}
LB = {'Name':'LB', 'NameStack':['P_CU_2', 'P_CU_2', 'P_CU_2',
↪ 'P_CU_2'], 'AngleStack':[0, 45, -45, 90], 'AngleU': 'deg'}

#Laminates={'L1':L1, 'L2':L2, 'L3':L3, 'L4':L4, 'TEST':TEST}
Laminates={'L1':L1, 'L2':L2, 'L3':L3, 'L4':L4, 'L5':L5, 'TEST':TEST,
↪ 'AL68':AL68, 'AL82':AL82, 'LC1':LC1, 'LC2':LC2, 'LB':LB}

for i in Laminates:
    Laminates[i]['Number'] =
    ↪ list(range(1, len(Laminates[i]['NameStack'])+1))

```

D.5 Main File

This is the main file of the Auxiliary Python code, the one to be executed. It uses information stored at the laminate dictionary and also the Bureau Veritas method dictionary. Most of the code is contained on a function which requires a vector of code names of the laminates to analyze, which are stored in the laminate dictionary. It is recommended to find the input variable at the middle of the code by searching in the code file the characters “Laminate_code_stack =” (in most windows programs this can be done by pressing the keys Control + F and then typing the word to find). By assigning the labels or code names of the laminates (for instance “L5”) in the vector and executing the code, the program will generate output .txt files with the analysis presented at the report.

```
"""
```

```
June 2024
```

```
@author: Pol Roca Iruela
```

```
"""
```

```
#FILE NAME: LaminateDesignFunction4.py
```

```
import BV_materials_dictionary as dt
```

```
import LaminatesCustom1 as dl
```

```
import LaminatesDesign1 as dd
```

```
import numpy as np
```

```
def LaminateDesign(Laminate_Code, Custom_Design_Name=['NoNameLam'],
```

```
↪ Custom_Design_Code=['NoLam'], Custom_Design_Angle=[0]):
```

```
    global Ceq, Fib, Mat, e, d_l, Vr, Mr, d_sup
```

```
    global Name, FibName, MatName, Fab, Pf, Pextra, Vf, Mf, d_f, d_r
```

```
    global Er, nur, Gr, Ef0, Ef90, nuf0, nuf90, Gf
```

```
def Input_Component_Type(Fab, Mat_inp, Fib_inp, ply_code):
```

```
global Ceq, Fib, Mat

#Decide fibre disposition and Ceq value
if Fab != "UD" and Fab != "WR" and Fab != "CSM":
    print('Invalid enter. Type of fabric, variable Fab')

if Fab == "UD":
    print ("UniDirectional Fabric")
    Ceq=0
elif Fab == "WR":
    print ("Woven Roven")
    Ceq=dl.LamDict[ply_code]['Ceq']
elif Fab == "CSM":
    print("Chopped Strand Mat")
    Ceq=0
else:
    print ("ERROR: Incorrect Value. Type of fabric, variable
    ↪ Fab")
    exit()

#Type of fibre and matrix determination
if Fib_inp == "E":
    print ("E-Glass")
    Fib = "E-Glass"
elif Fib_inp == "R":
    print ("R-Glass")
    Fib = "R-Glass"
elif Fib_inp == "CHS":
    print("Carbon HS")
    Fib = "Carbon-HS"
elif Fib_inp == "CIM":
    print("Carbon IM")
    Fib = "Carbon-IM"
```

```

elif Fib_inp == "CHM":
    print("Carbon HM")
    Fib = "Carbon-HM"
elif Fib_inp == "PA":
    print("Para-Amid")
    Fib = "Para-Amid"
else:
    print ("ERROR: Incorrect Value. Type of fibre, variable
    ↪ Fib_inp")
    exit()

if Mat_inp == "P":
    print ("Polyester")
    Mat = "Polyester"
elif Mat_inp == "V":
    print ("Vinilester")
    Mat = "Vinilester"
elif Mat_inp == "EP":
    print("Epoxy")
    Mat = "Epoxy"
else:
    print ("ERROR: Incorrect Value. Type of matrix,
    ↪ variable Mat_inp")
    exit()

def Input_Elastic_Properties(Mat, Fib, Er, Gr, nur, Ef0, Ef90, Gf,
↪ nuf0, nuf90):

    #Elastic properties calculaion
    #Mechanical Characteristics of Resins

    #Tensile Young Modulus of the Resin (Er) in MPa
    if Er==( ' '):
        Er=dt.MatrixDict[Mat]['Er']

```

```

else:
    Er=float(Er)

#Poisson Coefficient of the Resin (nur) in MPa
if nur=='':
    nur=dt.MatrixDict[Mat]['nur']
else:
    nur=float(nur)

#Shear Modulus of the Resin (Gr) in MPa
if Gr=='':
    Gr=dt.MatrixDict[Mat]['Gr']
elif Gr=="ISO":
    Gr = 0.5*Er/(1+nur)
else:
    Gr=float(Gr)

#Mechanical Characteristics of fibers

#Longitudinal Young modulus of the fiber, in MPa (Ef0)
if Ef0=='':
    Ef0=dt.FiberDict[Fib]['Ef0']
else:
    Ef0=float(Ef0)

#Transversal Young modulus of the fiber (Ef90) in MPa
if Ef90=='':
    Ef90=dt.FiberDict[Fib]['Ef90']
elif Ef90=="SAME":
    Ef90=Ef0
else:
    Ef90=float(Ef90)

#Poisson coefficient in fiber direction (nuf0)

```

```

if nuf0==(''):
    nuf0=dt.FiberDict[Fib]['nuf0']
else:
    nuf0=float(nuf0)

#Poisson coefficient in normal fiber direction (nuf90)
if nuf90==(''):
    nuf90=dt.FiberDict[Fib]['nuf90']
elif nuf90=="SAME":
    nuf90=nuf0
else:
    nuf90=float(nuf90)

#Shear modulus of fiber (Gf) in MPa
if Gf==(''):
    Gf=dt.FiberDict[Fib]['Gf']
elif Gf=="ISO":
    Gf=0.5*(Ef0+Ef90)/(2+nuf0+nuf90)
else:
    Gf=float(Gf)

return [Er, Gr, nur, Ef0, Ef90, Gf, nuf0, nuf90]

def Input_Densities_Thickness(Mat, Fib, d_f, d_r, Vf, Mf):

    global e, d_l, Vr, Mr, d_sup

    #Geometrical and Physical properties

    # Enter fiber and resin density of an individual layer
if d_f==(''):
    d_f = dt.FiberDict[Fib]['Density']
else:
    d_f=float(d_f) #g/cm3

```

```

if d_r=='':
    d_r = dt.MatrixDict[Mat]['Density']
else:
    d_r=float(d_r) #g/cm3

#Input data in volume or fiber content

if Vf==' ' and Mf==' ':
    print("ERROR: There is no value on fiber content of either
    → volume (Vf) or mass (Mf) fractions. Some value need to
    → be inputed")
    exit()

elif Vf!=' ' and Mf!=' ':
    print("ATTENTION: There are two values defined on fiber
    → content of volume (Vf) and mass (Mf) fractions. Only
    → one is needed, calculation is done with the VOLUME
    → FRACTION INPUT")

# Calculate Volume Fraction of Resin
    Vr=1-Vf

# Calculate Mass fraction of reinforcements
    Mf=(Vf*d_f/((Vf*d_f)+(1-Vf)*d_r))

#Calculate Mass fraction of resin
    Mr=(1-Mf)

elif Vf!=' ':

    # Calculate Volume Fraction of Resin
        Vr=1-Vf

    # Calculate Mass fraction of reinforcements
        Mf=(Vf*d_f/((Vf*d_f)+(1-Vf)*d_r))

    #Calculate Mass fraction of resin

```

```

Mr=(1-Mf)

elif Mf!='':

# Calculate Mass Fraction of Resin
Mr=1-Mf

# Calculate Volume fraction of reinforcements
Vf=(Mf/d_f/((Mf/d_f)+(1-Mf)/d_r))
#Calculate Volume fraction of resin
Vr=(1-Vf)

else:
    print("CODE ERROR, see final else on Input data in volume
    ↪ or mass fractions")
    exit()

#input total mass per square meter (g/m^2) of dry reinforcement
↪ fabric (Pf)
#Calculate the density of an individual layer
d_l=d_f*Vf+d_r*(1-Vf) #g/cm^3
#Calculate layer thickness
#e=Pf/(Vf*d_f)/1000 #e in [mm] then Pf in [g/m^2]

#Overall mass per square meter considering thickness e and
↪ density d_l
#d_sup = d_l*e/10 #g/cm^2 if e in[mm] and d_l in [g/cm^3]
d_sup = Pf/Mf/1e4 #g/cm^2

e=d_sup/d_l *10 #mm

d_sup = d_sup + Pextra/1e4 #g/cm^2 Addition of non counted
↪ additional weight that could exist in form of (for instane)
↪ stabilization threads

```

```

    return [d_f, d_r, Vf, Mf]

def Input_General>Loading(ply_code):
    #INTRODUIR DADES A PARTIR DE ply_code

    #INPUT VARIABLES: Fab * , Ceq *(for Fab = WR) , Fib_inp * ,
    ↪ Mat_inp * , d_f , d_r , Vf/Mf * , Pf * , Er, Gr, nur ,
    ↪ Ef0, Ef90, Gf, nuf0, nuf90
    global Name, FibName, MatName, Fab, Pf, Pextra, Vf, Mf, d_f,
    ↪ d_r
    global Er, nur, Gr, Ef0, Ef90, nuf0, nuf90, Gf
    global Ceq, Fib, Mat, e, d_l, Vr, Mr, d_sup

    Name = dl.LamDict[ply_code]['Name']
    FibName = dl.LamDict[ply_code]['FibName']
    MatName = dl.LamDict[ply_code]['MatName']

    Fab = dl.LamDict[ply_code]['Fab']
    Fib_inp = dl.LamDict[ply_code]['Fib_inp']
    Mat_inp = dl.LamDict[ply_code]['Mat_inp']
    d_f = dl.LamDict[ply_code]['d_f']
    d_r = dl.LamDict[ply_code]['d_r']
    Vf = dl.LamDict[ply_code]['Vf']
    Mf = dl.LamDict[ply_code]['Mf']
    Pf = dl.LamDict[ply_code]['Pf']
    Pextra = dl.LamDict[ply_code]['Pextra']

    Er = dl.LamDict[ply_code]['Er']
    Gr = dl.LamDict[ply_code]['Gr']
    nur = dl.LamDict[ply_code]['nur']

    Ef0 = dl.LamDict[ply_code]['Ef0']
    Ef90 = dl.LamDict[ply_code]['Ef90']
    Gf = dl.LamDict[ply_code]['Gf']

```

```

nuf0 = dl.LamDict[ply_code]['nuf0']
nuf90 = dl.LamDict[ply_code]['nuf90']

Input_Component_Type(Fab, Mat_inp, Fib_inp, ply_code) #Matrix
↪ and fibre type and Ceq value

[d_f, d_r, Vf, Mf]=Input_Densities_Thickness(Mat, Fib, d_f,
↪ d_r, Vf, Mf) #Calculation of volumetric participation and
↪ densities

[Er, Gr, nur, Ef0, Ef90, Gf, nuf0,
↪ nuf90]=Input_Elastic_Properties(Mat, Fib, Er, Gr, nur, Ef0,
↪ Ef90, Gf, nuf0, nuf90) #Loading of mechanical data values

def Ply_Elastic_Properties():

    #global CUD1, CUD2, CUD12, CUDnu

    #Elastic properties calculation
    #BV Material Coefficients
    CUD1 = dt.CoeffDict[Fib]['CUD1']
    CUD2 = dt.CoeffDict[Fib]['CUD2']
    CUD12 = dt.CoeffDict[Fib]['CUD12']
    CUDnu = dt.CoeffDict[Fib]['CUDnu']

    #Calculate results
    #UD
    EUD1=CUD1*(Ef0*Vf+Er*(1-Vf))

    ↪ EUD2=CUD2*((Er/(1-nur**2)))*((1+(0.85*Vf**2))/(((1-Vf)**1.25)+((Er*Vf)

    EUD3=EUD2
    eta=((Gf/Gr)-1)/((Gf/Gr)+1)
    GUD12=CUD12*Gr*((1+eta*Vf)/(1-eta*Vf))
    GUD13=GUD12

```

```

GUD23=0.7*GUD12
nuUD12=CUDnu*(nuf0*Vf+nur*(1-Vf))
nuUD13=nuUD12
nuUD21=nuUD12*EUD2/EUD1
nuUD31=nuUD21
nuprimaf=nuf0*Ef90/Ef0
nuUD23=CUDnu*(nuprimaf*Vf+nur*(1-Vf))
nuUD32=nuUD23

if Fab == 'UD':
    return [EUD1, EUD2, EUD3, GUD12, GUD23, GUD13, nuUD12,
            ↪ nuUD23, nuUD13, nuUD21, nuUD32, nuUD31]
elif Fab == 'WR':
    #WR
    Q11=EUD1/(1-(nuUD12*nuUD21))
    Q22=EUD2/(1-(nuUD12*nuUD21))
    Q12=(nuUD21*EUD1)/(1-(nuUD12*nuUD21))
    Q33=GUD12
    A11=e*(Ceq*Q11+(1-Ceq)*Q22)
    A22=e*(Ceq*Q22+(1-Ceq)*Q11)
    A12=e*Q12
    A33=e*Q33
    E1=(1/e)*(A11-(A12**2/A22))
    E2=(1/e)*(A22-(A12**2/A11))
    E3=EUD3
    G12=(1/e)*A33
    G13=0.9*G12
    G23=G13
    nu12=A12/A22
    nu21=nu12*(E2/E1)
    nu32=(nuUD32+nuUD31)/2
    nu31=nu32
    nu13=(nuUD23+nuUD13)/2
    nu23=nu13

```

```

    return [E1, E2, E3, G12, G23, G13, nu12, nu23, nu13, nu21,
            ↪ nu32, nu31]

elif Fab == 'CSM':
    #CSM
    E1=(3/8)*EUD1+(5/8)*EUD2
    E2=E1
    E3=EUD3
    G12=E1/(2*(1+0.3))
    G23=0.7*GUD12
    G13=G23
    nu12=0.3
    nu21=nu12
    nu32=nu12
    nu23=nu32*E2/E3
    nu13=nu12
    nu31=nu13*E3/E1
    return [E1, E2, E3, G12, G23, G13, nu12, nu23, nu13, nu21,
            ↪ nu32, nu31]

else:
    print("Variable Fab has changed value other than UD, WR,
          ↪ CSM after loading data to the program. See error around
          ↪ function Ply_Elastic_Properties().")

def Ply_R_S_Matrix(E1, E2, nu12, nu21, G12, G13, G23):
    #In-plane rigidity of an individual layer
    R11=(E1/(1-nu12*nu21))
    R22=E2/(1-nu12*nu21)
    R12=nu21*E1/(1-nu12*nu21)
    R21=nu12*E2/(1-nu12*nu21)
    R33=G12
    R_local_axes=np.array([[R11, R12, 0], [R21, R22, 0],
                            ↪ [0,0,R33]])      #Units of E1 (MPa)

```

```

#In-plane flexibility of an individual layer
S11=1/E1
S22=1/E2
S12=-nu21/E2
S21=-nu12/E1
S33=1/G12
S_local_axes=np.array([[S11, S12, 0], [S21, S22, 0],
↪ [0,0,S33]]) #Units of 1/E1 (mm**2/N = 1/MPa)

#Interlaminar rigidity (FSDT)
H11 = G23
H22 = G13
R_H_local_axes = np.array([[H11, 0], [0, H22]]) #Units of G23
↪ (MPa)

#Interlaminar flexibility (FSDT)
Hinv11 = 1/G23
Hinv22 = 1/G13
S_H_local_axes = np.array([[Hinv11, 0], [0, Hinv22]]) #Units
↪ of 1/G23 (mm**2/N = 1/MPa)

return [R_local_axes, S_local_axes, R_H_local_axes,
↪ S_H_local_axes]

def Ply_Isotropic_Core (PLY):
    #Isotropic material parameters
    E=PLY['Er']
    nu=PLY['nur']
    G=PLY['Gr']

    #Obtention of failure values
    sigmabrt1=PLY['sgBrT']
    sigmabrcl=PLY['sgBrC']

```

```

    taubr12=PLY['sgBrS']
    #Obtention of densities
    Pf = PLY['Pf'] #g/m^2
    d_r = PLY['d_r'] #g/cm^3
    d_f=d_r
    e = Pf/d_r/1000 #e in [mm] then Pf in [g/m^2]
    #Overall mass per square meter considering thickness e
    ↪ and density d_l
    d_sup = d_r*e/10 #g/cm^2 if e in[mm] and d_l in
    ↪ [g/cm^3]
    if PLY['Mat_inp']=="PVCF":
        Mat = "PVC Foam"
    else:
        Mat = ''

    return [E, nu, G, sigmabrt1, sigmabrcl, taubr12, Pf,
    ↪ d_r, d_f, e, d_sup, Mat]

def Ply_Output_File(OutFile, PlyDictionary, plyCode, d_f, d_r):
    PD = PlyDictionary[plyCode]

    rhoFib = d_f #g/cm**3
    rhoMat = d_r #g/cm**3

    R = PD['R'] #MPa
    S = PD['S'] #1/MPa

    OutFile.write('\t' + '\t' + "*****" + '\t' + '\t'
    ↪ +'\n'+'\n')
    OutFile.write(PD['Name'] +'\n'+'\n')

    OutFile.write(PD['FibName']+'\t' + "VPart" +'\t' + "mPart" +'\t'+
    ↪ "rho" +'\t' + PD['MatName']+'\t' + "VPart" +'\t' + "mPart"
    ↪ +'\t' + "rho" +'\n')

```

```

OutFile.write('\t'+ str(round(100*PD['Vf'],2)) +'\t'+
↳ str(round(100*PD['Mf'],2)) +'\t'+ str(round(rhoFib,2))
↳ +'\t'+'\t'+ str(round(100*PD['Vmat'],2)) +'\t'+
↳ str(round(100*PD['Mmat'],2)) +'\t'+ str(round(rhoMat,2))
↳ +'\n')
OutFile.write('\t'+ "%" +'\t'+ "%" +'\t'+ "g/cm3" +'\t'+'\t'+
↳ "%" +'\t'+ "%" +'\t'+ "g/cm3" +'\n'+'\n')

OutFile.write("E1" +'\t'+ "E2" +'\t'+ "E3" +'\t'+ "nu12" +'\t'+
↳ "nu1z" +'\t'+ "nu2z" +'\t'+ "nu21" +'\t'+ "nuz1" +'\t'+
↳ "nuz2" +'\n')
OutFile.write(str(round(PD['E1'],0)) +'\t'+
↳ str(round(PD['E2'],0)) +'\t'+ str(round(PD['E3'],0)) +'\t'+
↳ str(round(PD['nu12'],2)) +'\t'+ str(round(PD['nu13'],2))
↳ +'\t'+ str(round(PD['nu23'],2)) +'\t')
OutFile.write(str(round(PD['nu21'],2)) +'\t'+
↳ str(round(PD['nu31'],2)) +'\t'+ str(round(PD['nu32'],2))
↳ +'\n')
OutFile.write("MPa" +'\t'+ "MPa" +'\t'+ "MPa" +'\t'+ '\t'
↳ +'\t'+ '\t'+ '\t'+ '\t'+ '\n'+'\n')

OutFile.write("G12" +'\t'+ "G1z" +'\t'+ "G2z" +'\t'+ "rho"
↳ +'\t'+ "rhoSup" +'\t'+ "rhoFib" +'\t'+ "thknss" +'\n')
OutFile.write(str(round(PD['G12'],0)) +'\t'+
↳ str(round(PD['G13'],0)) +'\t'+ str(round(PD['G23'],0))
↳ +'\t'+ str(round(PD['rho'],2)) +'\t'+
↳ str(round(PD['rhoSup']*1e4,2)) +'\t'+
↳ str(round(PD['Pf'],2)) +'\t'+ str(round(PD['e'],2)) +'\n')
OutFile.write("MPa" +'\t'+ "MPa" +'\t'+ "MPa" +'\t'+ "g/cm3"
↳ +'\t'+ "g/m2" +'\t'+ "g/m2" +'\t'+ "mm" +'\n'+'\n')

OutFile.write("sigXt" +'\t'+ "sigXc" +'\t'+ "sigYt" +'\t'+
↳ "sigYc" +'\t'+ "sigS" +'\t'+ "sigIL1" +'\t'+ "sigIL2"
↳ +'\n')

```

```

OutFile.write(str(round(PD['sg1t'],0)) +'\t'+
↳ str(round(PD['sg1c'],0)) +'\t'+ str(round(PD['sg2t'],0))
↳ +'\t'+ str(round(PD['sg2c'],0)) +'\t'+
↳ str(round(PD['sgS'],0)) +'\t'+ str(round(PD['sg1IL'],0))
↳ +'\t'+ str(round(PD['sg2IL'],0)) +'\n')
OutFile.write("MPa" +'\t'+ "MPa" +'\t'+ "MPa" +'\t'+ "MPa"
↳ +'\t'+ "MPa" +'\t'+ "MPa" +'\t'+ "MPa" +'\n'+'\n')

OutFile.write("epsXt" +'\t'+ "epsXc" +'\t'+ "epsYt" +'\t'+
↳ "epsYc" +'\t'+ "epsS" +'\t'+ "epsIL1" +'\t'+ "epsIL2"
↳ +'\n')
OutFile.write(str(round(PD['ep1t'],2)) +'\t'+
↳ str(round(PD['ep1c'],2)) +'\t'+ str(round(PD['ep2t'],2))
↳ +'\t'+ str(round(PD['ep2c'],2)) +'\t'+
↳ str(round(PD['epS'],2)) +'\t'+ str(round(PD['ep1IL'],2))
↳ +'\t'+ str(round(PD['ep2IL'],2)) +'\n')
OutFile.write("%" +'\t'+ "%" +'\t'+ "%" +'\t'+ "%" +'\t'+ "%"
↳ +'\t'+ "%" +'\t'+ "%" +'\n'+'\n')

OutFile.write("R = " +'\t'+ str(round(R[0][0], 0)) +'\t'+
↳ str(round(R[0][1], 0)) +'\t'+ str(round(R[0][2], 0)) +'\t')
OutFile.write("S = " +'\t'+ str(round(S[0][0], 6)) +'\t'+
↳ str(round(S[0][1], 6)) +'\t'+ str(round(S[0][2], 6)) +'\n')
OutFile.write("[MPa]" +'\t'+ str(round(R[1][0], 0)) +'\t'+
↳ str(round(R[1][1], 0)) +'\t'+ str(round(R[1][2], 0)) +'\t')
OutFile.write("[1/MPa]" +'\t'+ str(round(S[1][0], 6)) +'\t'+
↳ str(round(S[1][1], 4)) +'\t'+ str(round(S[1][2], 6)) +'\n')
OutFile.write('\t'+ str(round(R[2][0], 0)) +'\t'+
↳ str(round(R[2][1], 0)) +'\t'+ str(round(R[2][2], 0)) +'\t')
OutFile.write('\t'+ str(round(S[2][0], 6)) +'\t'+
↳ str(round(S[2][1], 6)) +'\t'+ str(round(S[2][2], 6))
↳ +'\n'+'\n')

```

```

#OutFile.write("S = " + '\t'+ str(round(S[0][0], 4)) + '\t'+
↳ str(round(S[0][1], 4)) + '\t'+ str(round(S[0][2], 4)) + '\n')
#OutFile.write("[1/MPa]" + '\t'+ str(round(S[1][0], 4)) + '\t'+
↳ str(round(S[1][1], 4)) + '\t'+ str(round(S[1][2], 4)) + '\n')
#OutFile.write('\t'+ str(round(S[2][0], 4)) + '\t'+
↳ str(round(S[2][1], 4)) + '\t'+ str(round(S[2][2], 4))
↳ + '\n'+'\n')

OutFile.write('\t'+ '\t'+ "*****" + '\t'+ '\t'
↳ + '\n'+'\n')

def Ply_Output_Console(PlyDictionary, plyCode, d_f, d_r):
    PD = PlyDictionary[plyCode]

    #rhoFib = float(dl.LamDict[plyCode]['d_f'])
    #rhoMat = float(dl.LamDict[plyCode]['d_r'])

    rhoFib = d_f
    rhoMat = d_r

    print(PD['Name'] + '\n'+'\n')

    print(PD['FibName']+'\t'+ "VPart" +'\t'+ "mPart" +'\t'+ "rho"
↳ +'\t'+ PD['MatName']+'\t'+ "VPart" +'\t'+ "mPart" +'\t'+
↳ "rho" +'\n')
    print('\t'+ str(round(100*PD['Vf'],2)) +'\t'+
↳ str(round(100*PD['Mf'],2)) +'\t'+ str(round(rhoFib,2))
↳ +'\t'+'\t'+ str(round(100*PD['Vmat'],2)) +'\t'+
↳ str(round(100*PD['Mmat'],2)) +'\t'+ str(round(rhoMat,2))
↳ +'\n')
    print('\t'+ "%" +'\t'+ "%" +'\t'+ "g/cm3" +'\t'+'\t'+ "%"
↳ +'\t'+ "%" +'\t'+ "g/cm3" +'\n'+'\n')

```

```

print("E1" + '\t' + "E2" + '\t' + "E3" + '\t' + "nu12" + '\t' + "nu1z"
↪ + '\t' + "nu2z" + '\t' + "nu21" + '\t' + "nuz1" + '\t' + "nuz2"
↪ + '\n')

print(str(round(PD['E1'],0)) + '\t' + str(round(PD['E2'],0))
↪ + '\t' + str(round(PD['E3'],0)) + '\t' +
↪ str(round(PD['nu12'],2)) + '\t' + str(round(PD['nu13'],2))
↪ + '\t' + str(round(PD['nu23'],2)) + '\t')

print(str(round(PD['nu21'],2)) + '\t' + str(round(PD['nu31'],2))
↪ + '\t' + str(round(PD['nu32'],2)) + '\n')

print("MPa" + '\t' + "MPa" + '\t' + "MPa" + '\t' + '\t' + '\t' + '\t'
↪ + '\t' + '\t' + '\n'+'\n')

print("G12" + '\t' + "G1z" + '\t' + "G2z" + '\t' + "rho" + '\t' +
↪ "rhoSup" + '\n')

print(str(round(PD['G12'],0)) + '\t' + str(round(PD['G13'],0))
↪ + '\t' + str(round(PD['G23'],0)) + '\t' +
↪ str(round(PD['rho'],2)) + '\t' + str(round(PD['rhoSup'],2))
↪ + '\n')

print("MPa" + '\t' + "MPa" + '\t' + "MPa" + '\t' + "g/cm3" + '\t' +
↪ "g/cm2" + '\n'+'\n')

print("sigXt" + '\t' + "sigXc" + '\t' + "sigYt" + '\t' + "sigYc"
↪ + '\t' + "sigS" + '\n')

print(str(round(PD['G12'],0)) + '\t' + str(round(PD['G13'],0))
↪ + '\t' + str(round(PD['G23'],0)) + '\t' +
↪ str(round(PD['rho'],0)) + '\t' + str(round(PD['rhoSup'],0))
↪ + '\n')

print("MPa" + '\t' + "MPa" + '\t' + "MPa" + '\t' + "MPa" + '\t' + "MPa"
↪ + '\n'+'\n')

print("epsXt" + '\t' + "epsXc" + '\t' + "epsYt" + '\t' + "epsYc"
↪ + '\t' + "epsS" + '\n')

```

```

print(str(round(PD['G12'],2)) +'\t'+ str(round(PD['G13'],2))
↪ +'\t'+ str(round(PD['G23'],2)) +'\t'+
↪ str(round(PD['rho'],2)) +'\t'+ str(round(PD['rhoSup'],2))
↪ +'\n')
print("%" +'\t'+ "%" +'\t'+ "%" +'\t'+ "%" +'\t'+ "%" +'\t'+ "%")
↪ +'\n'+'\n')

#MAIN PROGRAM
ply_dict={}

#Output file name
OutFile_name = "OUTPUT_"+Laminate_Code+".txt"

#INPUT VARIABLES for 'Custom' Laminate_Code manual design
#Custom_Design_Name = 'Custom Laminate'
#Custom_Design_Code = ['P_UD_1', 'P_UD_2', 'P_WR_1', 'P_UD_1']
↪ #Code of the plies considered ordered from bottom to top along
↪ z axis
#Custom_Design_Angle = [0, 30, -30, 90] #Angle of the plies
↪ (ordered) in DEGREES

#LAMINATE STRUCTURE GENERATION -- Initialization of L dictionary
↪ with design parameters
if Laminate_Code == ('custom' or 'Custom' or 'CUSTOM'):
    CodeStack = Custom_Design_Code #CodeStack = Ply Codes ordered
    ↪ from lowest ply to highest ply
    L = {'Name':Custom_Design_Name, 'NameStack':CodeStack,
    ↪ 'AngleStack':Custom_Design_Angle, 'AngleU': 'deg'}
    ↪ #Generation of custom laminate structure

```

```

L['Number'] = list(range(1, len(L['NameStack'])+1))
elif Laminate_Code in dd.Laminates:
    CodeStack = dd.Laminates[Laminate_Code]['NameStack'] #CodeStack
    ↪ = Ply Codes ordered from lowest ply to highest ply
    L = dd.Laminates[Laminate_Code] #Import Laminate from
    ↪ dictionary
else:
    print("ERROR: Laminate_Code not recognized. Typed string should
    ↪ appear at dd.Laminates or else 'Custom' should be typed in
    ↪ order to define manually a custom laminate.")

#SORTING OF DIFFERENT PLIES TO ANALIZE INDIVIDUALLY
ply_code = [CodeStack[0]] #Initialization of ply_code = Vector with
    ↪ codes of all DIFFERENT (non repeating) ply types in the
    ↪ laminate
#ply_code.append(CodeStack[0])
for i in range(len(CodeStack)):
    if CodeStack[i] not in dl.LamDict:
        print("The ply with code: "+CodeStack[i]+" does not appear
        ↪ on the ply dictionary dl.LamDict. Its information must
        ↪ be provided.")
        exit()
    if CodeStack[i] not in ply_code:
        ply_code.append(CodeStack[i])

OutFile = open(OutFile_name, "w") #Open file to write
Lines = '*****LAMINATE CALCULATOR*****'+'\n\n'
OutFile.write(Lines)

#COMPUTATION OF INDIVIDUAL PLY PARAMETERS
for i in range(len(ply_code)): #For every type of ply

    if ply_code[i] not in dl.LamDict: #Ply not found on dictionary
        print()

```

```

print("\033[4;35m'+Attention!' +'\033[0;m')
print("Ply named "+ply_code[i]+" is not found on the
↳ dictionary. Calculation proceeds without considering
↳ it.")

Lines = 'Attention!'+'\n'+ "Ply named "+ply_code[i]+" is not
↳ found on the dictionary. Calculation proceeds without
↳ considering it."+'\n'+'\n'
OutFile.write(Lines)

elif dl.LamDict[ply_code[i]]['Vf'] == 0 or
↳ dl.LamDict[ply_code[i]]['Mf'] == 0: #If the current ply is
↳ an isotropic core material (special case)

#Tailored parameters to isotropic core material case
PLY = dl.LamDict[ply_code[i]]

#Elastic properties generation for isotropic core (special
↳ case)
[E, nu, G, sigmabrt1, sigmabrcl, taubr12, Pf, d_r, d_f, e,
↳ d_sup, Mat] = Ply_Isotropic_Core(PLY)

#Elastic matrices generation (plane stress)
[R_local_axes, S_local_axes, R_H_local_axes,
↳ S_H_local_axes] = Ply_R_S_Matrix(E, E, nu, nu, G, G, G)

#PLY DICTIONARY DATA FILL
ply_dict[ply_code[i]]={'Name':PLY['Name'],
↳ 'e':e, 'R':R_local_axes, 'S':S_local_axes, 'RShear':R_H_local_axes, 'S
↳ 'Pf':Pf, 'Vf':0, 'Mf':0, 'Vmat':1,
↳ 'Mmat':1,
↳ 'rho':d_r, 'rhoSup':d_sup, 'E1':E,
↳ 'E2':E, 'E3':E, 'G12':G, 'G13':G,
↳ 'G23':G,

```

```

'nu12':nu, 'nu21':nu, 'nu13':nu,
↪ 'nu31':nu, 'nu23':nu, 'nu32':nu,
'sg1t':sigmabrt1, 'sg2t':sigmabrt1,
↪ 'sg1c':sigmabrc1, 'sg2c':sigmabrc1,
↪ 'sgS':taubr12, 'sg1IL':taubr12,
↪ 'sg2IL':taubr12,
'ep1t':-1.0, 'ep2t':-1.0, 'ep1c':-1.0,
↪ 'ep2c':-1.0, 'epS':-1.0,
↪ 'ep1IL':-1.0, 'ep2IL':-1.0,
'Fab':PLY['Fab'], 'Fib':Mat, 'Mat':Mat,
↪ 'FibName':PLY['FibName'],
↪ 'MatName':PLY['MatName']}

```

```
Ply_Output_File(OutFile, ply_dict, ply_code[i], d_f, d_r)
```

```
else: #Regular composite ply
```

```
Input_General>Loading (ply_code[i]) #Loading data from
↪ materials dictionary
```

```
#Elastic properties generation
```

```
[E1, E2, E3, G12, G23, G13, nu12, nu23, nu13, nu21, nu32,
↪ nu31] = Ply_Elastic_Properties()
```

```
#Elastic matrices generation (plane stress)
```

```
[R_local_axes, S_local_axes, R_H_local_axes,
↪ S_H_local_axes] = Ply_R_S_Matrix(E1, E2, nu12, nu21,
↪ G12, G13, G23)
```

```
#theoretical strains calculations
```

```
epsbrt1=dt.StrainDict[Fib][Fab]['epsbrt1']
epsbrc1=dt.StrainDict[Fib][Fab]['epsbrc1']
epsbrt2=dt.StrainDict[Fib][Fab]['epsbrt2']
epsbrc2=dt.StrainDict[Fib][Fab]['epsbrc2']
```

```

gammabr12=dt.StrainDict[Fib][Fab]['gammabr12']
gammaIL1=dt.StrainDict[Fib][Fab]['gammaIL1']
gammaIL2=dt.StrainDict[Fib][Fab]['gammaIL2']

#In-plane theoretical breaking stresses
#stating coefres due to adhesive properties of the resin
Coefres=dt.MatrixDict[Mat]['Coefres']
sigmabrt1=epsbrt1*E1*Coefres/100
sigmabrcl=epsbrcl*E1*Coefres/100
sigmabrt2=epsbrt2*E2*Coefres/100
sigmabrc2=epsbrc2*E2*Coefres/100
taubr12=gammabr12*G12*Coefres/100
tauIL1=gammaIL1*G23*Coefres/100
tauIL2=gammaIL2*G13*Coefres/100

#PLY DICTIONARY DATA FILL
ply_dict[ply_code[i]]={'Name':Name,
    ↪ 'e':e, 'R':R_local_axes, 'S':S_local_axes, 'RShear':R_H_local_axes, 'S
        'Pf':Pf, 'Vf':Vf, 'Mf':Mf, 'Vmat':Vr,
    ↪ 'Mmat':Mr,
        'rho':d_l, 'rhoSup':d_sup, 'E1':E1,
    ↪ 'E2':E2, 'E3':E3, 'G12':G12,
    ↪ 'G13':G13, 'G23':G23,
        'nu12':nu12, 'nu21':nu21, 'nu13':nu13,
    ↪ 'nu31':nu31, 'nu23':nu23,
    ↪ 'nu32':nu32,
        'sg1t':sigmabrt1, 'sg2t':sigmabrt2,
    ↪ 'sg1c':sigmabrcl, 'sg2c':sigmabrc2,
    ↪ 'sgS':taubr12, 'sg1IL':tauIL1,
    ↪ 'sg2IL':tauIL2,
        'ep1t':epsbrt1, 'ep2t':epsbrt2,
    ↪ 'ep1c':epsbrcl, 'ep2c':epsbrc2,
    ↪ 'epS':gammabr12, 'ep1IL':gammaIL1,
    ↪ 'ep2IL':gammaIL2,

```

```

        'Fab':Fab, 'Fib':Fib, 'Mat':Mat,
        ↪ 'FibName':FibName,
        ↪ 'MatName':MatName}

#R, RShear, E[x], G[xy], sg[x]t, sg[x]c, sgS, sg[x]IL -->
↪ MPa
#S, SShear --> 1/MPa
#e --> mm
#rho --> g/cm**3
#Pf, rhoSup --> g/m**2
#ep[x]t, ep[x]c, epS, ep[x]IL --> % --> (epsilon
↪ non-dimensional = ep[x]t / 100)

Ply_Output_File(OutFile, ply_dict, ply_code[i], d_f, d_r)

#Storage of ply_dict info for generating NASTRAN .bdf file
L['ply_dict'] = ply_dict
L['ply_code'] = ply_code

def Lam_Structure_Coordinate_Change(L, nPlies):
    zPos = np.zeros(nPlies+1) #Position in z axis of the joints
    ↪ between plies [mm]
    Rmat = np.array([[1,0,0],[0,1,0],[0,0,2]]) #Transformation
    ↪ matrix used in change of coordinates
    RmatInv = np.array([[1,0,0],[0,1,0],[0,0,0.5]])

#CALCULATION OF LAMINATE PROPERTIES DEPENDING ON INDIVIDUAL PLY
↪ DATA

for i in range(nPlies): # i --> ply number on the laminate
    ID = L['NameStack'][i] #Ply code to access ply_dict for
    ↪ the ply number i

```

```

L['eStack'][i] = ply_dict[ID]['e'] #mm

L['rhoSupStack'][i] = ply_dict[ID]['rhoSup'] #g/m**2
L['totalrhoSup'] += L['rhoSupStack'][i] #g/m**2

#Assignment of strings to define the type of ply in the
↪ laminate
match ply_dict[ID]['Mat']:
    case "Epoxy": MatTypeString = 'Ep'
    case "Vinilester": MatTypeString = 'Vin'
    case "Polyester": MatTypeString = 'Pol'
    case "PVC Foam": MatTypeString = ' '

match ply_dict[ID]['Fib']:
    case "E-Glass": FibTypeString = 'E-G'
    case "R-Glass": FibTypeString = 'R-G'
    case "Carbon-HS": FibTypeString = 'C-HS'
    case "Carbon-IM": FibTypeString = 'C-IM'
    case "Carbon-HM": FibTypeString = 'C-HM'
    case "Para-Amid": FibTypeString = 'P-Am'
    case "PVC Foam": FibTypeString = 'PVC'
    case " ": FibTypeString = MatTypeString

match ply_dict[ID]['Fab']:
    case "CSM": FabTypeString = 'CSMat'
    case "UD": FabTypeString = 'UDirect'
    case "WR": FabTypeString = 'WovenR'
    case "PVC Foam": FabTypeString = 'Foam'

L['Type1'][i] = FibTypeString+'/'+MatTypeString
L['Type2'][i] = FabTypeString

zPos[i+1] = zPos[i]+L['eStack'][i] #mm

```

```

#PLY TO LAMINATE COORDINATE CHANGE
#Cosine and sine of the angle of ply number i (measured
↳ from the laminate coordinate system to the ply
↳ coordinate system)
if L['AngleU']=='deg':
    c = np.cos(np.deg2rad(L['AngleStack'][i]))
    s = np.sin(np.deg2rad(L['AngleStack'][i]))
else:
    c = np.cos(L['AngleStack'][i])
    s = np.sin(L['AngleStack'][i])
    print('The given angle of the plies is considered to be
↳ in RADIANS.')

#Transformation matrix T ({123} = T @ {xyz}) and its
↳ inverse ({xyz} = Tinv @ {123})
TrotMat = np.array([[c**2, s**2, 2*s*c], [s**2, c**2,
↳ -2*s*c], [-s*c, s*c, c**2-s**2]])
TrotMatInv = np.array([[c**2, s**2, -2*s*c], [s**2, c**2,
↳ 2*s*c], [s*c, -s*c, c**2-s**2]])
L['stackCmat'][i] = TrotMatInv @ ply_dict[ID]['R'] @ Rmat @
↳ TrotMat @ RmatInv #C matrix on laminate coordinates
↳ [MPa]

#Transformation matrix T for the transverse shear matrix H
TrotMatH = np.array([[c, -s], [s, c]])
TrotMatHInv = np.array([[c, s], [-s, c]])
L['stackShearmat'][i] = TrotMatHInv @
↳ ply_dict[ID]['RShear'] @ TrotMatH #H matrix on
↳ laminate coordinates [MPa]

L['totalTh'] = zPos[-1] #mm

```

```

Z0 = L['totalTh']/2 #MidPlane z coordinate distance from the
↳ outer surfaces
zPos = zPos - Z0 #All positions should be taken from MidPlane
↳ coordinate as the origin

L['zPos'] = zPos #mm

return [L, zPos, Z0]

def Lam_ABC_H_Matrix(L, zPos, Z0):
    deltaZ = zPos[1:]-zPos[:-1] #Thickness [mm]
    deltaZ2 = zPos[1:]**2 - zPos[:-1]**2 #Ply position related
    ↳ parameter [mm**2]
    deltaZ3 = zPos[1:]**3-zPos[:-1]**3 # mm**3
    zPosHalf = zPos[:-1] + 0.5*deltaZ #z position of the
    ↳ respective half thickness point [mm]

    #PLATE STIFFNESS MATRICES

    #Inicialization of plate stiffness matrices
    L['A'] = np.zeros((3,3)) #Extenssional stiffness matrix
    ↳ [N/mm]
    L['B'] = np.zeros((3,3)) #Bending-extension coupling stiffness
    ↳ matrix [N]
    L['D'] = np.zeros((3,3)) #Bendinf stiffness matrix [N*m]
    L['H'] = np.zeros((2,2)) #Transverse shear stiffness matrix
    ↳ [N/mm]

    for i in range(nPlies): #For each ply
        #Construction of matrices by addition of ply contributions
        L['A'] += L['stackCmat'][i] * deltaZ[i]

```

```

L['B'] += L['stackCmat'][i] * deltaZ2[i] / (2) #Valid for
↳ an INDIRECT coordinate base such that:  $z = -x \wedge y$  (x
↳ cross product y)
#L['B'] += L['stackCmat'][i] * deltaZ2[i] / (-2) #Valid for
↳ an DIRECT coordinate base such that:  $z = x \wedge y$  (x
↳ cross product y)
L['D'] += L['stackCmat'][i] * deltaZ3[i] / 3

L['H'] += L['stackShearmat'][i] * (deltaZ[i] - (deltaZ[i] *
↳ (zPosHalf[i])**2 + deltaZ[i]**3 /12)/(Z0**2)) * 5/4

L['ABD'] = np.r_[np.c_[L['A'], L['B']], np.c_[L['B'], L['D']]]
↳ #General stiffness matrix
L['DBA'] = np.linalg.inv(L['ABD']) #Inverse of the general
↳ stiffness matrix

L['Hinv'] = np.linalg.inv(L['H']) #Inverse of the transverse
↳ shear stiffness matrix

return L

def Lam_BV_Interlaminar_Stress(L, nPlies, zPos):

L['T-tau'] = np.zeros((nPlies,2,2)) #1/mm

AuxRant = np.zeros((3,3))
AuxR = np.zeros((3,3))
AuxCxzant = np.zeros((6))
AuxCyzant = np.zeros((6))
AuxCxz = np.zeros((6))
AuxCyz = np.zeros((6))
for i in range(nPlies):

```

```

AuxMAB = zPos[i] * L['DBA'][:3,:3] + 0.5*zPos[i]**2 *
↳ L['DBA'][3:,:3]
AuxMBD = zPos[i] * L['DBA'][3:,:3] + 0.5*zPos[i]**2 *
↳ L['DBA'][3:,3:]
AuxM = np.c_[AuxMAB, AuxMBD]

AuxR = ply_dict[L['NameStack'][i]]['R']

AuxCxz = AuxCxzant + (AuxR[0][:] - AuxRant[0][:]) @ AuxM
AuxCyz = AuxCyzant + (AuxR[1][:] - AuxRant[1][:]) @ AuxM

L['T-tau'][i] = np.array([[AuxCyz[4] - AuxR[1,:]] @
↳ AuxMBD[:,1],
                               AuxCyz[5] - AuxR[1,:] @
                               ↳ AuxMBD[:,2]],
                          [AuxCxz[5] - AuxR[0,:] @ AuxMBD[:,2],
                          AuxCxz[3] - AuxR[0,:] @
                          ↳ AuxMBD[:,0]]]) #1/mm

AuxRant = AuxR
AuxCxzant = AuxCxz
AuxCyzant = AuxCyz

return L

def Lam_Equivalent_Elastic_Parameters(L):
L['EngCtt']={}
L['EngCtt']['Ex'] = 1/(L['DBA'][0,0]*L['totalTh']) #MPa
L['EngCtt']['Ey'] = 1/(L['DBA'][1,1]*L['totalTh']) #MPa

L['EngCtt']['Gxy'] = 1/(L['DBA'][2,2]*L['totalTh']) #MPa
L['EngCtt']['Gxz'] = 1/(L['Hinv'][0,0]*L['totalTh']) #MPa
L['EngCtt']['Gyz'] = 1/(L['Hinv'][1,1]*L['totalTh']) #MPa

```

```

L['EngCtt']['nuxy'] = L['ABD'][1,0]/L['ABD'][1,1] #nuxy -->
↳ nu_x
L['EngCtt']['nuyx'] = L['ABD'][0,1]/L['ABD'][0,0] #nuyx -->
↳ nu_y

L['EngCtt']['EIx'] = 1/(L['DBA'][3,3]) #Global Bending rigidity
↳ x [N*mm]
L['EngCtt']['EIy'] = 1/(L['DBA'][4,4]) #Global Bending rigidity
↳ y [N*mm]

return L

def Lam_Output_File(OutFile, L):
    OutFile.write('\t\t' + "-----" + '\t\t'
↳ +'\n')
    OutFile.write('\t\t' + "Laminate: " + L['Name'] + '\t\t'
↳ +'\n')
    OutFile.write('\t\t' + "-----" + '\t\t'
↳ +'\n'+'\n')
    #OutFile.write("Number" +'\t\t' + "Code" +'\t\t' + "Angle" +'\t\t'
↳ "Unit" +'\t\t' + "Type" +'\t\t'+'\t\t' + "Thknss" +'\t\t' + "rhoSup"
↳ +'\n')
    OutFile.write("Number" +'\t\t' + "Code" +'\t\t' + "Angle" +'\t\t'
↳ "Type" +'\t\t'+'\t\t' + "Thknss" +'\t\t' + "rhoSup" +'\n')
    for i in range(len(L['NameStack'])-1, -1, -1):
        #OutFile.write(L['Number'][i]+' \t\t' + L['NameStack'][i]
↳ +'\t\t' + str(round(L['AngleStack'][i], 1)) +'\t\t'
↳ L['AngleU'] +'\t\t' + L['Type1'][i] +'\t\t' + L['Type2'][i]
↳ +'\t\t' + str(round(L['e'][i], 2)) +'\t\t'
↳ str(round(L['rhoSup'][i], 2)) +'\n')

```

```

    OutFile.write(str(L['Number'][i])+'\t'+ L['NameStack'][i]
        ↪ +'\t'+ str(round(L['AngleStack'][i], 1)) +'\t'+
        ↪ L['Type1'][i] +'\t'+ L['Type2'][i] +'\t'+
        ↪ str(round(L['eStack'][i], 2)) +'\t'+
        ↪ str(round(1e4*L['rhoSupStack'][i], 2)) +'\n')
    OutFile.write("Total" +'\t'+ L['Name'] +'\t'+'\t'+'\t'+'\t'+
        ↪ str(round(L['totalTh'], 2)) +'\t'+
        ↪ str(round(1e4*L['totalrhoSup'], 2)) +'\n')
    OutFile.write('\t'+'\t'+ L['AngleU'] +'\t'+'\t'+'\t'+ "mm"
        ↪ +'\t'+ "g/m2" +'\n'+'\n')
    OutFile.write("Global rigidity matrix (ABD) = " +'\n')
    OutFile.write(str(np.round(L['ABD'], 2)) +'\n'+'\n')
    OutFile.write("Global compliance matrix (ABD-1) = " +'\n')
    OutFile.write(str(np.round(L['DBA'], 6)) +'\n'+'\n')
    OutFile.write("Global transverse shear matrix (H) = " +'\n')
    OutFile.write(str(np.round(L['H'], 2)) +'\n'+'\n')
    OutFile.write("Global inverse transverse shear matrix (H-1)
        ↪ = " +'\n')
    OutFile.write(str(np.round(L['Hinv'], 6)) +'\n'+'\n')

```

#LAMINATE DICTIONARY VALUES INTIALIZATION

```

nPlies = len(L['NameStack']) #Number of plies in the laminate

L['eStack']=[0]*nPlies #Vector containing thickness of every ply
    ↪ [mm]
L['rhoSupStack']=[0]*nPlies #Vector containing superficial density
    ↪ of each ply [g/m**2]
L['Type1']=[0]*nPlies #String defining properties of the ply
L['Type2']=[0]*nPlies
L['totalTh']=0 #Scalar indicating the total thickness of the
    ↪ laminate [mm]
L['totalrhoSup']=0 #Scalar indicating the total surface density of
    ↪ the laminate [g/m**2]

```

```

L['stackCmat']=[np.zeros((3,3))*nPlyes #Rigidity matrix C of each
↳ ply in laminate coordinates [MPa]
L['stackShearmat']=[np.zeros((2,2))*nPlyes #Transverse shear
↳ stiffness matrix of each ply in laminate coordinates [MPa]

#CALCULATION OF LAMINATE PROPERTIES DEPENDING ON INDIVIDUAL PLY
↳ DATA
[L, zPos, Z0] = Lam_Structure_Coordinate_Change(L, nPlyes)

#PLATE STIFFNESS MATRICES
L = Lam_ABC_H_Matrix(L, zPos, Z0)

#INTERLAMINAR stresses tau according to Bureau Veritas
L = Lam_BV_Interlaminar_Stress(L, nPlyes, zPos)

#LAMINATE EQUIVALENT ELASTIC PROPERTIES
L = Lam_Equivalent_Elastic_Parameters(L)

#LAMINATE OUTPUT FILE
Lam_Output_File(OutFile, L)

OutFile.close() #Close file

return L

def MSC_Nastran_Material_BDF(PlyDictionary, plyCodeStack):
def Nform(Input, CellCharLen = 8):
    if Input > 0:
        Negative = False
        OccupiedSpaces = 1 #Number of spaces occupied by
↳ characters other than digits (1 for the decimal point)
    elif Input < 0:
        Negative = True

```

```

OccupiedSpaces = 2 #Number of spaces occupied by
↳ characters other than digits (1 for the decimal point
↳ and another for the negative sign)

else:
    return "0.0"+" "*(CellCharLen-3)

DigInt = 0 #Number of integer digits
DigDec = 0 #Number of decimal digits
DigExp = 0 #Value of scientific notation base 10 exponent of
↳ the number
IntInp = np.abs(Input)
DecInp = IntInp
while IntInp >= 1: #Counter of integer digits
    DigInt += 1
    IntInp /= 10.0

while np.abs(DecInp-np.round(DecInp,0)) > 1e-9: #Counter of
↳ decimal digits
    DigDec += 1
    if DecInp < 1: #Counter to store the value of negative
↳ exponent when value < 1
        DigExp += 1
    DecInp *= 10.0

SciInp = IntInp
while SciInp >=10 or SciInp < 1:
    SciInp *= 10
SciInp = np.round(SciInp, DigInt+DigDec)

DigExp = -DigExp #If the exponent is non zero, then the
↳ positive value that it has is the correct value changed of
↳ sign
#If the value is zero, then the change of sign does not affect
↳ the value.

```

```

#Now DigExp is the true value of the exponent if that is
↪ negative and 0 otherwise

if DigExp == 0: #The exponent value is not yet computed if it
↪ is positive
    DigExp = DigInt-1

if DigExp >= 3 or DigExp <= -3 or DigDec > 3: #Case where too
↪ many digits are present and scientific notation is needed
    Cientific = True
else: #Scientific notation not needed
    Cientific = False

if Cientific:
    if np.abs(DigExp) >= 10:
        OccupiedSpaces += 3 #There are 2 digits occupied by
↪ the exponent and another for the sign of the
↪ exponent
    else:
        OccupiedSpaces += 2 #There is 1 digit occupied by the
↪ exponent and another for the sign of the exponent

    if DigDec + DigInt > CellCharLen-OccupiedSpaces:
        Out = f"{np.round(SciInp,
↪ (CellCharLen-OccupiedSpaces-1))}"
    else:
        Out = f"{SciInp}"

    if Negative:
        Out = f"-{Out}{DigExp:+d}"
    else:
        Out = f"{Out}{DigExp:+d}"

else:

```

```

    if DigDec == 0:
        Out = f"{int(np.trunc(Input))}."
    elif DigInt == 0 and Negative:
        Out = f"{Input}"
        Out = f"-{Out[2:]}"
    elif DigInt == 0:
        Out = f"{Input}"
        Out = Out[1:]
    else:
        Out = f"{Input}"

SpaceAddition = ""
for i in range(CellCharLen-len(Out)):
    SpaceAddition += " "

return Out + SpaceAddition

FileName = "NastranMAT.bdf"
BDF_File = open(FileName, "w") #Open file to write

CellCharLen = 8
BlankSpace = " "*CellCharLen

for i in range(len(plyCodeStack)):
    PD = PlyDictionary[plyCodeStack[i]]
    MID = f"{i+1}"
    MID = MID + (CellCharLen-len(MID))*" "
    E1 = Nform(PD['E1']) #MPa
    E2 = Nform(PD['E2']) #Mpa
    NU12 = Nform(PD['nu12'])
    G12 = Nform(PD['G12']) #Mpa
    G1Z = Nform(PD['G13']) #Mpa
    G2Z = Nform(PD['G23']) #Mpa

```

```

RHO = Nform(PD['rho']/1e9)  #g/cm**3 /1e9
A1 = BlankSpace
A2 = BlankSpace
TREF = "293"  #K
TREF = TREF + (CellCharLen-len(TREF))*" "
Xt = Nform(PD['sg1t'])  #Mpa
Xc = Nform(PD['sg1c'])  #Mpa
Yt = Nform(PD['sg2t'])  #Mpa
Yc = Nform(PD['sg2c'])  #Mpa
S = Nform(PD['sgS'])  #Mpa
GE = BlankSpace
F12 = BlankSpace
STRN = BlankSpace

#Hashin failure
HF1 = Nform(PD['sg1t'])  #Mpa
HF2 = Nform(PD['sg1c'])  #Mpa
HF3 = Nform(56.0)  #Mpa  Now imposed GP33 Tensile strength
HF4 = Nform(110.0)  #Mpa  Now imposed GP33 Compressive
↳ strength
HF10 = Nform(PD['sgS'])  #Mpa
HF11 = Nform(np.max([PD['sg1IL'], PD['sg2IL']]))  #Mpa
#HF11 = Nform(51.0)  #Mpa  To impose intralaminar strength as
↳ GP33 shear strength

BDF_File.write(f"${PD['Name']}")
BDF_File.write(f"\n")
BDF_File.write(f"MAT8
↳ {MID}{E1}{E2}{NU12}{G12}{G1Z}{G2Z}{RHO}{BlankSpace}\n")

↳ BDF_File.write(f"{BlankSpace}{A1}{A2}{TREF}{Xt}{Xc}{Yt}{Yc}{S}{BlankSp
BDF_File.write(f"{BlankSpace}{GE}{F12}{STRN}{BlankSpace*6}\n")

```

```

BDF_File.write(f"{BlankSpace}{"HFAIL"+"
↳ "*" (CellCharLen-5) {"HF1}{HF2}{HF3}{HF4}{HF10}{HF11}\n"}
#BDF_File.write("\n")

BDF_File.close()

#Laminate_Code_stack = ['L1', 'L2', 'L3', 'L4', 'L5', 'LC1']
#Laminate_Code_stack = ['AL68', 'L3', 'L5']
Laminate_Code_stack = ['TEST', 'L5', 'LB']

Lam = {} #General Dictionary including all types of laminates in
↳ Laminate_Code_stack
MatCode = [] #List of every different material (ply) appearing in at
↳ least one of the laminates in Laminate_Code_stack
MatDict = {} #Dictionary with the data on the materials (plies) in
↳ MatDict

nLam = len(Laminate_Code_stack) #Number of laminates

for LCSNum in range(nLam):
    Lam[Laminate_Code_stack[LCSNum]] =
↳ Laminatedesign(Laminate_Code_stack[LCSNum])
    for i in range(len(Lam[Laminate_Code_stack[LCSNum]]['ply_code'])):
        if Lam[Laminate_Code_stack[LCSNum]]['ply_code'][i] not in
↳ MatCode:

            ↳ MatCode.append(Lam[Laminate_Code_stack[LCSNum]]['ply_code'][i])
        MatDict[MatCode[-1]] =
↳ Lam[Laminate_Code_stack[LCSNum]]['ply_dict'][MatCode[-1]]

MSC_Nastran_Material_BDF(MatDict, MatCode)

```

```

#print("Various interlaminar H from BV")
#print(Lam['L3']['T-tau'][1])
#print("H from Barbero")
#print(Lam['L3']['stackShearmat'][0] @ Lam['L3']['Hinv'])

#4 POINTS TEST ON A LAMINATE
def Loads_4PointTest(X, L, d, F, Tors=0, Shear=0, Nx=0, Ny=0):

    #X, L, d --> mm
    #F, Nx, Ny, Shear --> N/mm
    #Tors --> N * mm/mm = N

    x = X[0]
    if x < d[0] :
        nx = 0.5*F[0] #N/mm
        mx = 0 #N
    elif (x >= d[0]) and (x <= L[0]-d[0]):
        nx = 0
        mx = 0.5*F[0]*d[0]
    elif x > L[0]-d[0]:
        nx = -0.5*F[0]
        mx = -nx*L[0]

    Mx = nx*x + mx #N

    if x <= d[0] :
        Tx = 0.5*F[0] #N/mm
    elif (x > d[0]) and (x < L[0]-d[0]):
        Tx = 0
    elif x >= L[0]-d[0]:
        Tx = -0.5*F[0]

    y = X[1]

```

```

    if y < d[1] :
        ny = 0.5*F[1]
        my = 0

    elif (y >= d[1]) and (y <= L[1]-d[1]):
        ny = 0
        my = 0.5*F[1]*d[1]

    elif y > L[1]-d[1]:
        ny = -0.5*F[1]
        my = -ny*L[1]

My = ny*y + my #N

y = X[1]
if y <= d[1] :
    Ty = 0.5*F[1]
elif (y > d[1]) and (y < L[1]-d[1]):
    Ty = 0
elif y >= L[1]-d[1]:
    Ty = -0.5*F[1]

Nxy = Shear #N/mm
Mxy = Tors #N

return np.array([Nx, Ny, Nxy, Mx, My, Mxy, Tx, Ty]) #N/m and N

#4 POINT TEST INPUT PARAMETERS
L = np.array([1000, 50]) #mm
d = np.array([1/3, 0])*L #mm
X = np.array([1/3, 0])*L #mm
F = np.array([-0.856, 0]) #N/mm

FourPRes = Loads_4PointTest(X, L, d, F)

Load_Results = np.atleast_2d(FourPRes[:6]).T #N/mm and N

```

```

Load_Results_Shear = np.atleast_2d(FourPRes[6:]).T #N/mm

Lam_Strain_Results = []
Lam_Strain_Results_Shear = []
for i in range(nLam):
    Lam_Strain_Results.append(Lam[Laminate_Code_stack[i]]['DBA'] @
        ↪ Load_Results) #Non-dimensional and 1/mm
    Lam_Strain_Results_Shear.append(Lam[Laminate_Code_stack[i]]['Hinv']
        ↪ @ Load_Results_Shear) #Non-dimensional

def T_Rot_Mat(L, i):
    if L['AngleU']=='deg':
        c = np.cos(np.deg2rad(L['AngleStack'][i]))
        s = np.sin(np.deg2rad(L['AngleStack'][i]))
    else:
        c = np.cos(L['AngleStack'][i])
        s = np.sin(L['AngleStack'][i])
    print('The given angle of the plies is considered to be in
        ↪ RADIANS.')

    TrotMat = np.array([[c**2, s**2, 2*s*c], [s**2, c**2, -2*s*c],
        ↪ [-s*c, s*c, c**2-s**2]])
    TrotMatInv = np.array([[c**2, s**2, -2*s*c], [s**2, c**2, 2*s*c],
        ↪ [s*c, -s*c, c**2-s**2]])

    TrotMatH = np.array([[c, -s], [s, c]])
    TrotMatHInv = np.array([[c, s], [-s, c]])

    return [TrotMat, TrotMatInv, TrotMatH, TrotMatHInv]

def InterLam_stress(L, Qx, Qy, Theory='Nastran'):

    #Qx, Qy --> N/mm

```

```

#IST (Nastran method):

z = L['zPos'] #mm
G = L['stackCmat'] #MPa

Dxx = L['EngCtt']['EIx'] #N*mm
Dyy = L['EngCtt']['EIy']

#Dxx = (z[-1]-z[0])**3 /12 *L['EngCtt']['Ex']
#Dyy = (z[-1]-z[0])**3 /12 *L['EngCtt']['Ey']

sum1x = 0
sum1y = 0
sum2x = 0
sum2y = 0
for i in range(len(z)-1):
    sum1x +=
        ↪ (z[i+1]**2-z[i]**2)/np.linalg.inv(L['stackCmat'][i])[0,0]
        ↪ /2
    sum1y +=
        ↪ (z[i+1]**2-z[i]**2)/np.linalg.inv(L['stackCmat'][i])[1,1]
        ↪ /2
    sum2x += (z[i+1]-z[i])/np.linalg.inv(L['stackCmat'][i])[0,0]
    sum2y += (z[i+1]-z[i])/np.linalg.inv(L['stackCmat'][i])[1,1]

#Neutral axis
znx = sum1x/sum2x
zny = sum1y/sum2y

#Neutral axis: epsilon = eps_0 + psi * z --> epsilon = 0 --> z =
    ↪ -eps_0/psi
#znx = -Str[0,0]/Str[3,0] #Case bending in x axis
#zny = -Str[1,0]/Str[4,0] #Case bending in y axis

```

```

#print(znx)

taux = np.zeros(len(z)) #MPa
tauy = np.zeros(len(z))
Cx = np.zeros(len(z)) #MPa
Cy = np.zeros(len(z))

for i in range(1, len(z)):
    Cx[i] = -Qx * G[i-1][0,0] / Dxx * (znx * z[i-1] - z[i-1]**2 /2)
    ↪ + taux[i-1]
    Cy[i] = -Qy * G[i-1][1,1] / Dyy * (zny * z[i-1] - z[i-1]**2 /2)
    ↪ + tauy[i-1]

    taux[i] = Qx * G[i-1][0,0] / Dxx * (znx * z[i] - z[i]**2 /2) +
    ↪ Cx[i]
    tauy[i] = Qy * G[i-1][1,1] / Dyy * (zny * z[i] - z[i]**2 /2) +
    ↪ Cy[i]

#Bureau-Veritas method

taux_BV = np.zeros(len(z))
tauy_BV = np.zeros(len(z))
for i in range(len(z)-1):
    tau_BV = L['T-tau'][i] @ np.array([[Qx],[Qy]]) #MPa
    taux_BV[i] = tau_BV[0,0]
    tauy_BV[i] = tau_BV[1,0]

print(f"Nastran vs Bureau-Veritas: Laminate {L['Name']}")
print(f"Tau_IL_x")
print(f"{taux}\n{taux_BV}")
print(f"Tau_IL_y")
print(f"{tauy}\n{tauy_BV}\n")

```

```

if Theory == 'BureauVeritas' or Theory == 'BV' or Theory ==
↪ 'Bureau-Veritas' or Theory == 'bureauveritas' or Theory ==
↪ 'bureau-veritas':
    taux = taux_BV #MPa
    tauy = tauy_BV

return taux, tauy #MPa

def Stress_from_Lam_Strains(Str, ShearStr, L):

    #Str --> Non-dimensional and 1/mm
    #ShearStr --> Non-dimensional

    zPos = L['zPos'] #mm
    #print(f"Str = {zPos[0]*Str[3:,:]}")
    #Strain_Results = np.zeros(len(zPos))
    #Stress_Results = np.zeros(2*(len(zPos)-1))
    #Local_Stress_Results = np.zeros(2*(len(zPos)-1))
    Strain_Results = [0]*len(zPos)
    Stress_Results = [0]*2*(len(zPos)-1)
    Local_Stress_Results = [0]*2*(len(zPos)-1)
    for i in range(len(zPos)):

        Strain_Results[i] = Str[:3,:] + zPos[i]*Str[3:,:]
        ↪ #Non-dimensional
        if i == 0:
            Stress_Results[0] = L['stackCmat'][0] @ Strain_Results[i]
            ↪ #MPa
            Local_Stress_Results[0] = T_Rot_Mat(L,0)[0] @
            ↪ Stress_Results[0] #MPa

        elif i == len(zPos)-1:
            Stress_Results[-1] = L['stackCmat'][-1] @ Strain_Results[i]
            ↪ #MPa

```

```

Local_Stress_Results[-1] = T_Rot_Mat(L,-1)[0] @
↳ Stress_Results[-1] #MPa
else:
Stress_Results[2*i-1] = L['stackCmat'][i-1] @
↳ Strain_Results[i] #MPa
Local_Stress_Results[2*i-1] = T_Rot_Mat(L,i-1)[0] @
↳ Stress_Results[2*i-1] #MPa
Stress_Results[2*i] = L['stackCmat'][i] @ Strain_Results[i]
↳ #MPa
Local_Stress_Results[2*i] = T_Rot_Mat(L,i)[0] @
↳ Stress_Results[2*i] #MPa

return [Local_Stress_Results, Stress_Results, Strain_Results]

def Point_Results_File_Output(Lam, X, L, Lam_Strain,
↳ Load_Results_Shear, Laminate_Code_stack, File):

#Lam_strain --> Non-dimensional and 1/mm
#Load_Results_Shear --> N/mm
#L, X --> mm

tabSpaces = 8

nLam = len(Laminate_Code_stack)

File.write("TENSION STATE OF THE LAMINATE AT A SINGULAR POINT \n
↳ \n")
File.write(f"Plate DIMENSIONS:\t\tL_x = {L[0]}\t\tL_y = {L[1]} \n
↳ \n")
File.write(f"POINT:\t\t\tX = {X[0]}\t\t\tY = {X[1]} \n \n")

```

```

for i in range(nLam):
    L = Lam[Laminate_Code_stack[i]]
    InputStress = Stress_from_Lam_Strains(Lam_Strain[0][i],
    ↪ Lam_Strain[1][i], L)  #Stress in MPa and Strain in
    ↪ Non-dimensional

    #Interlaminar stress Nastran
    tauILX, tauILY = InterLam_stress(L, Load_Results_Shear[0,0],
    ↪ Load_Results_Shear[1,0])  #MPa

    #print(tauILX)
    #print(L['T-tau']@Load_Results_Shear)

    #print(InputStress[1][0])

    #Stress = [[0.0,0.0,0.0],[0.0,0.0,0.0]]*len(L['NameStack'])
    #LocStress = [[0.0,0.0,0.0],[0.0,0.0,0.0]]*len(L['NameStack'])

    #for j in range(len(L['NameStack'])):
    #    for k in range(3):
    #        Stress[j][0][k] = InputStress[1][2*j].T[0].tolist()[k]
    #        Stress[j][1][k] = InputStress[1][2*j+1][k,0]
    #        LocStress[j][0] = InputStress[0][2*j].T[0].tolist()
    #        LocStress[j][1] = InputStress[0][2*j+1].T[0].tolist()

    #print(InputStress[1][2*0].T[0].tolist())
    #print(Stress[0])

    eps=[]  #%
    tauIL=[]
    for j in range(len(L['NameStack'])+1):
        eps.append(100*InputStress[2][j].T[0])  #%
        tauIL.append([tauILX[j], tauILY[j]])  #MPa

```

```

File.write(f"LAMINATE CODE: {Laminate_Code_stack[i]} \n \n")

↪ File.write(f"sigX\tsig1\ttauILX\tsigY\tsig2\ttauILY\tepsX\tepsY\tgamma")
File.write(f"{MPa\t}*6}{%\t"*3}\n \n")

for j in range(len(L['NameStack'])-1, -1, -1):
    PlyChar = "#"

    ↪ File.write(f{"-"*(2*tabSpaces-1)+"<"}{np.round(tauIL[j+1][0],
    ↪ 3)}\t{"-"*(2*tabSpaces-1)+"<"}{np.round(tauIL[j+1][1],
    ↪ 3)}\t{np.round(eps[j+1][0], 2)}\t{np.round(eps[j+1][1],
    ↪ 2)}\t{np.round(eps[j+1][2], 2)}\n")
    #File.write(f"{np.round(Stress[j][1][0],
    ↪ 2)}\t{np.round(LocStress[j][1][0],
    ↪ 2)}\t\t{np.round(Stress[j][1][1],
    ↪ 2)}\t\t{np.round(LocStress[j][1][1], 2)}\t\t\t\t\t\n")
File.write(f"{np.round(InputStress[1][2*j+1][0,0],
    ↪ 2)}\t{np.round(InputStress[0][2*j+1][0,0],
    ↪ 2)}\t\t{np.round(InputStress[1][2*j+1][1,0],
    ↪ 2)}\t\t{np.round(InputStress[0][2*j+1][1,0],
    ↪ 2)}\t\t\t\t\t\n")
File.write(f"#[j+1] {PlyChar*(tabSpaces-4)}"
    ↪ "{L['NameStack'][j]}{" "+PlyChar*(2*tabSpaces-2)}"
    ↪ "{np.round(L['AngleStack'][j],2)} {L['AngleU']}{"
    ↪ "+PlyChar*(2*tabSpaces-2)}"
    ↪ "{np.round(L['eStack'][j],2)} mm{"
    ↪ "+PlyChar*(tabSpaces-1)}\n")
    #File.write(f"{np.round(Stress[j][0][0],
    ↪ 2)}\t{np.round(LocStress[j][0][0],
    ↪ 2)}\t\t{np.round(Stress[j][0][1],
    ↪ 2)}\t\t{np.round(LocStress[j][0][1], 2)}\t\t\t\t\t\n")

```

```

File.write(f"{np.round(InputStress[1][2*j][0,0],
↪ 2)}\t{np.round(InputStress[0][2*j][0,0],
↪ 2)}\t\t{np.round(InputStress[1][2*j][1,0],
↪ 2)}\t{np.round(InputStress[0][2*j][1,0],
↪ 2)}\t\t\t\t\t\n")
File.write(f"{ "-"*(2*tabSpaces-1)+"<"}{np.round(tauIL[0][0],
↪ 2)}\t{"-"*(2*tabSpaces-1)+"<"}{np.round(tauIL[0][1],
↪ 2)}\t{np.round(eps[0][0], 2)}\t{np.round(eps[0][1],
↪ 2)}\t{np.round(eps[0][2], 2)}\n \n \n")

```

```

FileName = "OUTPUT_POINT_RESULTS.txt"
File = open(FileName, "w") #Open file to write
Point_Results_File_Output(Lam, X, L, [Lam_Strain_Results,
↪ Lam_Strain_Results_Shear], Load_Results_Shear, Laminate_Code_stack,
↪ File)
File.close()

```

Bibliography

1. JOAQUÍN A. HERNÁNDEZ ORTEGA. *Enginyeria Aeroespacial Computacional - Finite Element Method (1D)*. ESEIAAT - UPC, 2023. Enginyeria Aeroespacial Computacional.
2. ZIENKIEWICZ, Olgierd Cecil; TAYLOR, Robert Lee; ZHU, J. Z. *The finite element method: its basis and fundamentals*. 6th ed. Burlington (Mass.): Elsevier/Butterworth-Heinemann, 2005. ISBN 978-0-7506-6320-5.
3. REDDY, Junuthula Narasimha. *Mechanics of laminated composite plates and shells: theory and analysis*. 2nd ed. Boca Raton (Fla.): CRC press, 2004. ISBN 978-0-8493-1592-3.
4. KAW, Autar K. *Mechanics of composite materials*. 2nd ed. Boca Raton, FL: Taylor & Francis, 2006. Mechanical engineering, no. v. 29. ISBN 978-0-8493-1343-1.
5. BARBERO, Ever J. *Introduction to composite materials design*. Second edition. Boca Raton, FL: CRC Press, 2011. ISBN 978-1-62870-636-9. OCLC: 880900083.
6. MSC SOFTWARE. *MSC.Nastran 2004 Reference Manual* [online]. MSC Software, 2004 [visited on 2024-05-30]. Available from: <http://www.ae.metu.edu.tr/~ae464/refman.pdf>.
7. RANGER, Ken. *MSC Nastran. Appendix A - Composite Theory for Plate Elements*. [N.d.].
8. JONES, Robert M. *Mechanics of composite materials*. 2nd ed. Philadelphia, PA: Taylor & Francis, 1999. ISBN 978-1-56032-712-7.
9. BUREAU VERITAS. *NR546 Hull in composite, plywood and high density polyethylene materials* [online]. Paris La Défense Cedex - France: Bureau Veritas Marine & Offshore, 2022. NR546 DT R04 [visited on 2024-02-19]. No. NR546. Available from: <https://marine-offshore.bureauveritas.com/nr546-hull-composite-plywood-and-high-density-polyethylene-materials>.
10. ALFONS BORRÀS BRELL; FERMÍN ENRIQUE OTERO GRUER; XAVIER MARTÍNEZ GARCIA. *Structural analysis of the FRP connections in an offshore semi-submersible platform*. Facultat de Nàutica de Barcelona, 2023. Treball de Final de Màster. Universitat Politècnica de Catalunya.