

# Manuel utilisateur

Ces classes permettent de réaliser des études d'homogénéisation. Il s'agit de pouvoir d'une part constituer les propriétés homogénéisées d'un stratifié, dans notre cas à partir des propriétés de la fibre et de la résine. On propose ensuite de pouvoir « localiser » les déformations et « concentrer » les contraintes à partir d'efforts généralisés.

Le modèle d'homogénéisation utilisé est celui d'Hashin et Shtrikman.

Deux modèles cinématiques sont proposés : la poutre et la plaque.

On commencera par expliquer les deux modes d'exécution puis on expliquera quelques modifications pouvant être apportées au code pour répondre à certains besoins.

## Table des matières

1	Exécution par l'interface.....	3
1.1	Initialisation .....	3
1.1.1	Pli unique.....	3
1.1.2	Poutre stratifiée .....	4
1.1.3	Plaque stratifiée.....	6
1.2	Interface de calcul de contrainte .....	6
2	Exécution en ligne de commande .....	9
2.1	Objets Materiau, Pli et Empilement .....	9
2.2	Etats de contrainte et déformation .....	10
2.2.1	Glossaire acronymes.....	12
2.2.2	Représentation des valeurs importantes.....	12
2.3	Resistances .....	13
3	Le code.....	14
3.1	Généralités .....	14
3.2	Modifier les graphes.....	15
3.2.1	Echantillonnage .....	15
3.2.2	Seuils.....	16
3.2.3	Ouverture automatique des graphes .....	17
3.3	Autres variables importantes .....	18
3.3.1	Mat_map .....	18
3.3.2	Dict_mat .....	18
4	Index .....	20
4.1	Index des figures.....	20
4.2	Index des tableaux.....	20



# 1 Exécution par l'interface

**Exécuter « interface.py » pour lancer l'application.** L'interface suivante apparaît (Figure 1), il suffit de suivre les instructions.

## 1.1 Initialisation

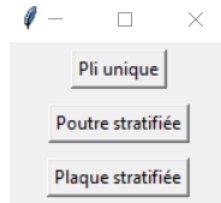


Figure 1 : interface de départ

### 1.1.1 Pli unique

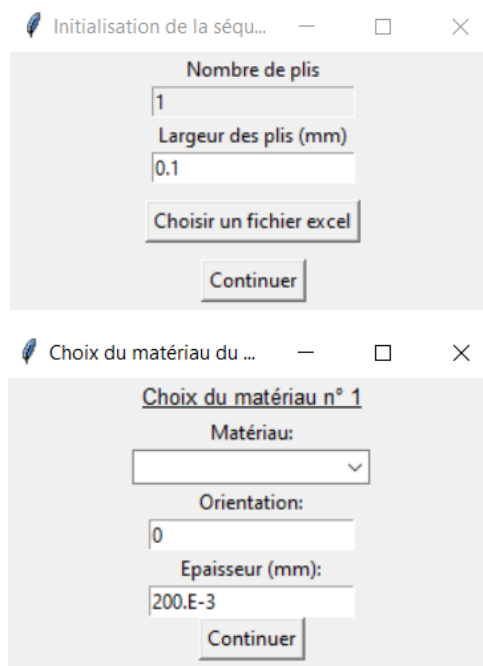


Figure 2 : choix du matériau

Avant de continuer, il faut ouvrir le menu déroulant sous Matériau et sélectionner le matériau souhaité (Figure 2). T300\_914 désigne un pli de fibre de carbone/résine et E\_914 fibre de verre/résine.

L'épaisseur est modifiable, la fraction volumique de fibre  $V_f$  sera recalculée en conséquence. L'orientation n'a ici pas d'incidence.

On précise qu'avec les valeurs actuelles fournies dans le dictionnaire de matériau, une épaisseur de 0.156 mm (0.1559278350515464) correspond à une fraction volumique de fibre de 0.6.

La fenêtre suivante apparaît (figure 3).

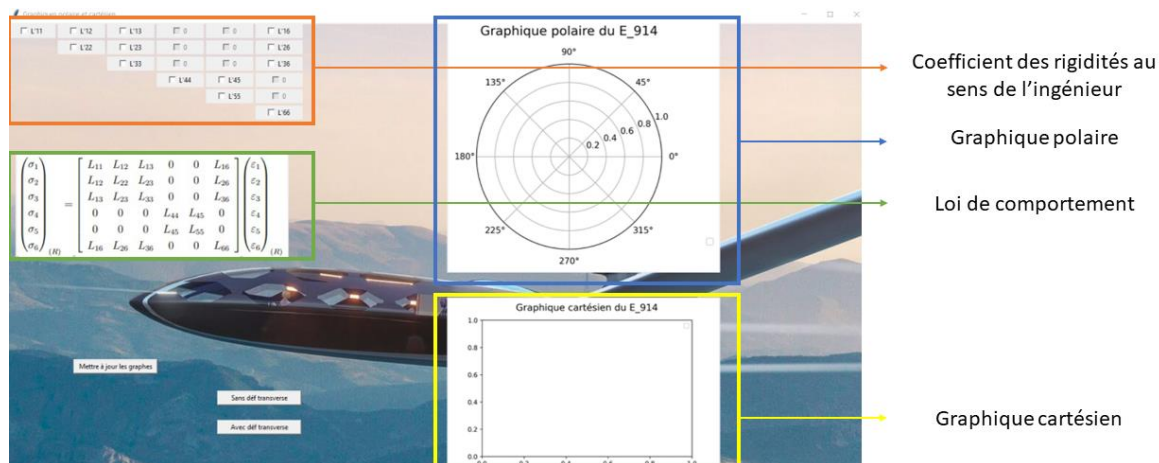


Figure 3 : interface présentant les caractéristiques d'un pli de fibre de verre

Il suffit de cocher les coefficients à afficher et à cliquer sur « Mettre à jour les graphes ».

### 1.1.2 Poutre stratifiée

Figure 4 shows the initialization window for a sequence. The window contains the following elements:

- Nombre de plis:** A text input field for the number of layers.
- Largeur des plis (mm):** A text input field with the value 0.1.
- Choisir un fichier excel:** A button to select an Excel file.
- Continuer:** A button to proceed with the sequence.

Figure 4 : initialisation de la séquence pour un empilement

Ici (figure 4), soit on entre un nombre de pli soit on clique sur « choisir un Excel ». L'explorateur de fichier s'ouvre dans le dossier Ressources et il faut sélectionner « Template\_batch.xlsx ».

Pour modifier l'Excel (utiliser Template\_batch.xlsx et sauvegarder les modifications), remplir les colonnes A à C incluses à partir de la ligne 2 comme montré dans l'exemple (voir figure 5). Utiliser les valeurs données dans la section notamment pour le nom car celui-ci est comparé au nom dans le dictionnaire. Voici quelques exemples de remplissage :

1	Nom_materiau	Orientation	Epaisseur	Valeurs par défaut										Indication:	Remplir les co
2	T300_914	90	160.E-3	Nom_materiau:	Orientation (en degré)	Epaisseur (mm)									
3	T300_914	90	160.E-3	T300_914	E_914	0 45 90 -45 -90	160.E-3	230.E-3							
4	T300_914	0	160.E-3												
5	T300_914	0	160.E-3												
6	E_914	45	230.E-3												
7	E_914	45	230.E-3												
8	E_914	-45	230.E-3												
9	E_914	-45	230.E-3												
10	T300_914	0	160.E-3												
11	T300_914	0	160.E-3												
12	T300_914	90	160.E-3												
13	T300_914	90	160.E-3												
14															
15															

Figure 5 : Excel représentant un empilement

**Attention :** Enregistrer l'Excel avec un pli va provoquer une erreur, utiliser le mode pli unique pour ce cas ou un nombre de pli égal à 1 dans l'interface « initialisation de la séquence ».

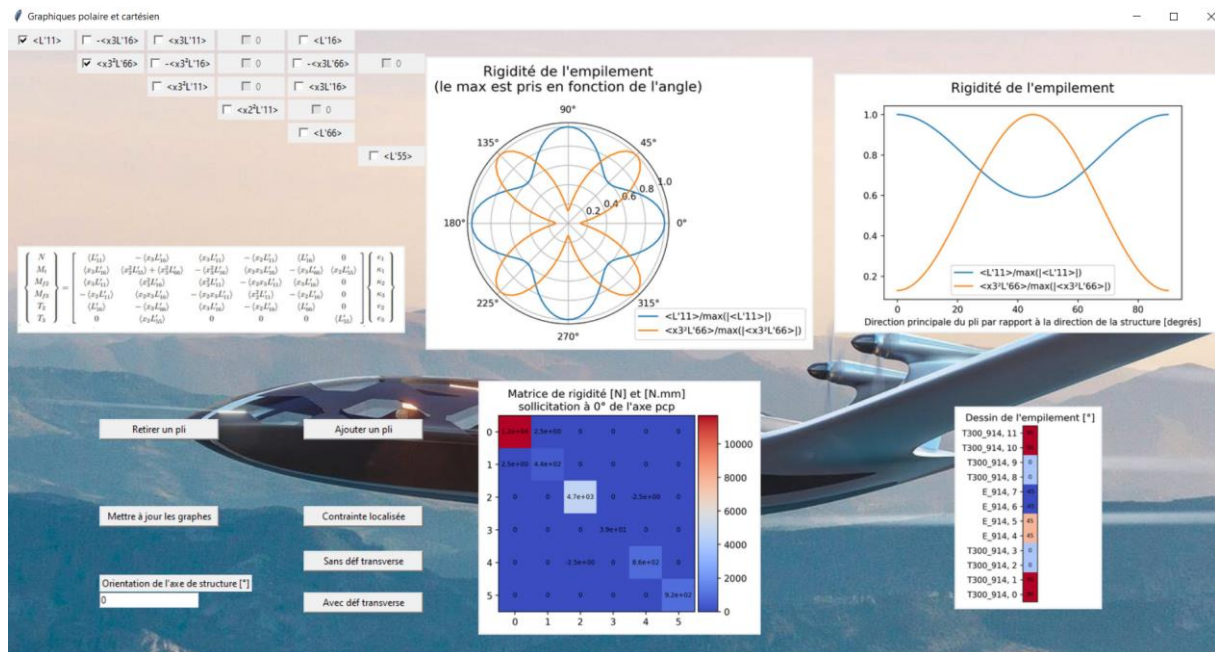


Figure 6 : interface de calcul de propriétés d'un empilement stratifié avec la théorie des poutres

On obtient alors l'interface présentée figure 6. On précise que les données générées seront stockées dans des dossiers présentés plus tard.

#### « Retirer un pli » :

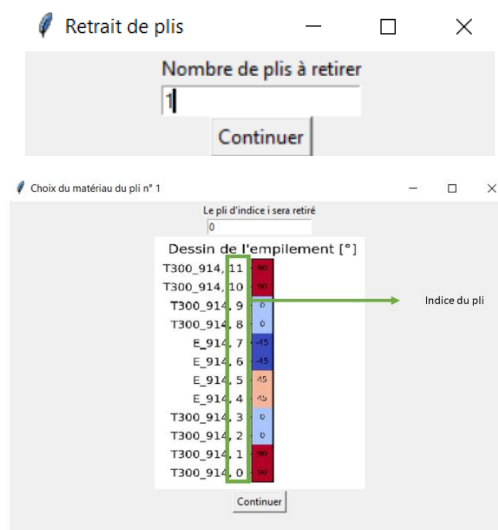


Figure 7 : interface pour retirer des plis

En cas de plusieurs plis à retirer (figure 7), le dessin est actualisé.

#### « Ajouter un pli » :

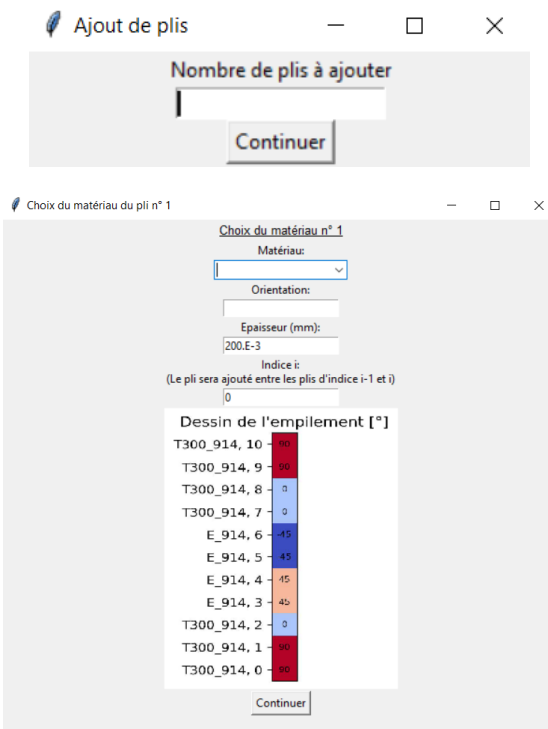


Figure 8 : interface pour ajouter un pli

**Attention :** Le pli est ajouté entre les plis d'indices i-1 et i, si on veut mettre un pli « en premier », c'est-à-dire à l'indice 0, il faut saisir 0 comme indice. Si on veut le mettre à l'indice 2, il faut saisir 2.

### 1.1.3 Plaque stratifiée

Le cheminement est le même pour la théorie des plaques (voir figure 9)

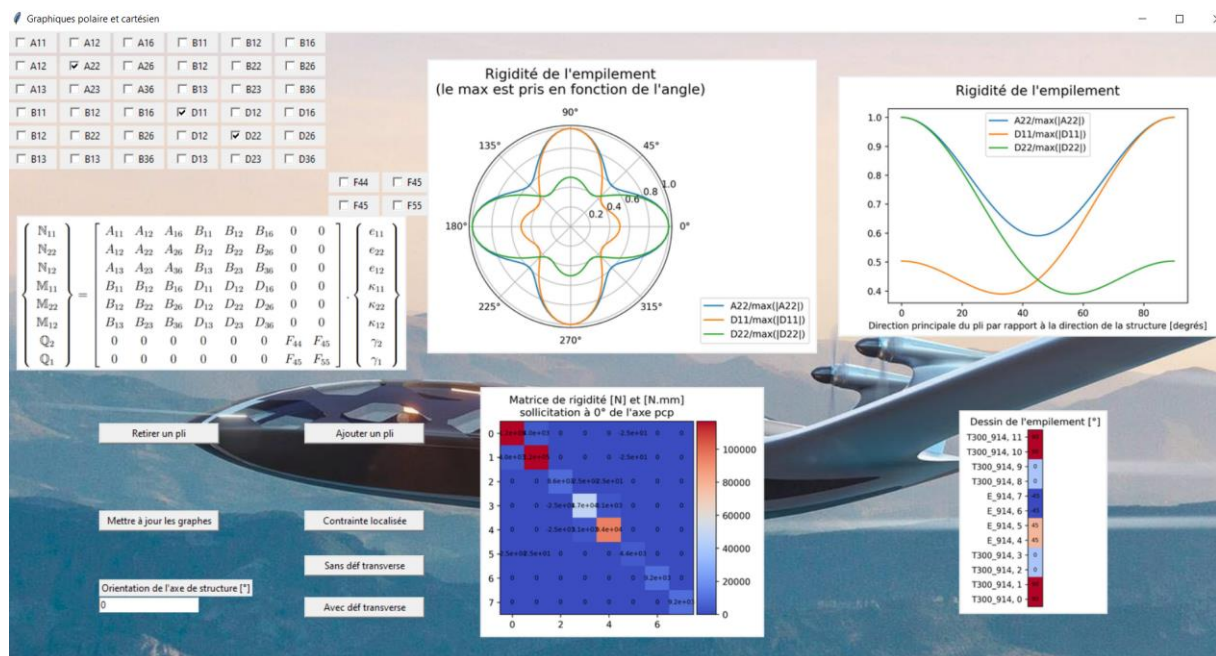


Figure 9 : interface des propriétés d'un empilement avec la théorie des plaques

## 1.2 Interface de calcul de contrainte

Appuyer sur le bouton « Contrainte localisée » (voir encadré rouge figure 10):

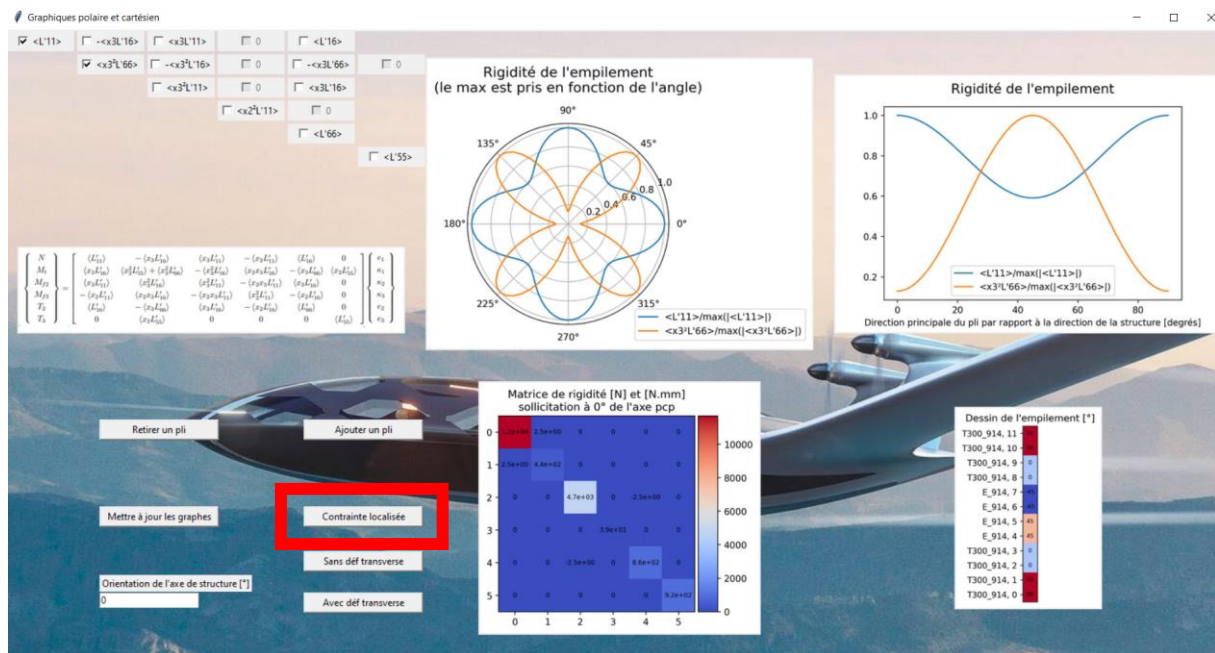


Figure 10 : emplacement de l'ouverture de l'interface de calcul de contraintes

L'interface suivante (figure 11) se lance :

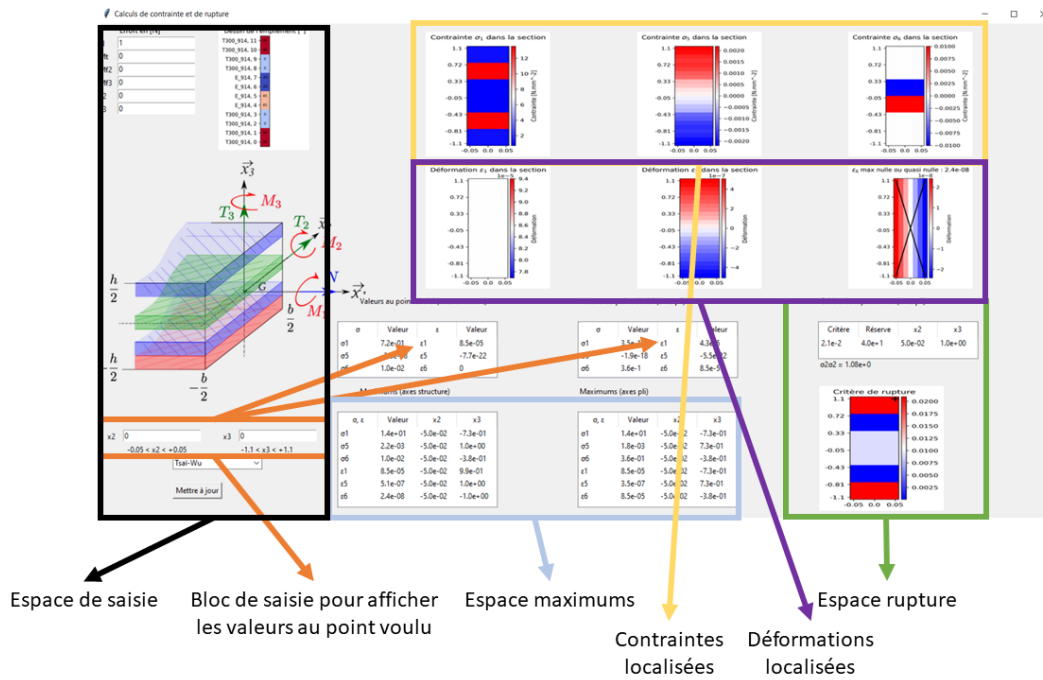


Figure 11 : interface de calcul de contrainte (théorie des poutres), non exécutée

Il faut entrer les efforts généralisés. On rappelle que ces derniers sont pris en un point  $x_1$  choisit lors de la mesure. On peut choisir un point auquel les valeurs précises de contrainte et de déformations seront données. On peut aussi choisir un critère de rupture entre Contrainte Max et Tsai-Wu.

Il faut ensuite cliquer sur « Mise à jour ». L'interface de calcul des contraintes affiche des dessins des contraintes et déformations dans la section pour une poutre et le long d'un brin transverse au feuillet moyen pour une plaque. Elle affiche également les valeurs des différentes contraintes et déformations au point sélectionné, et les contraintes et déformations maximums. Des informations sur la rupture sont aussi affichées (voir figure 12)



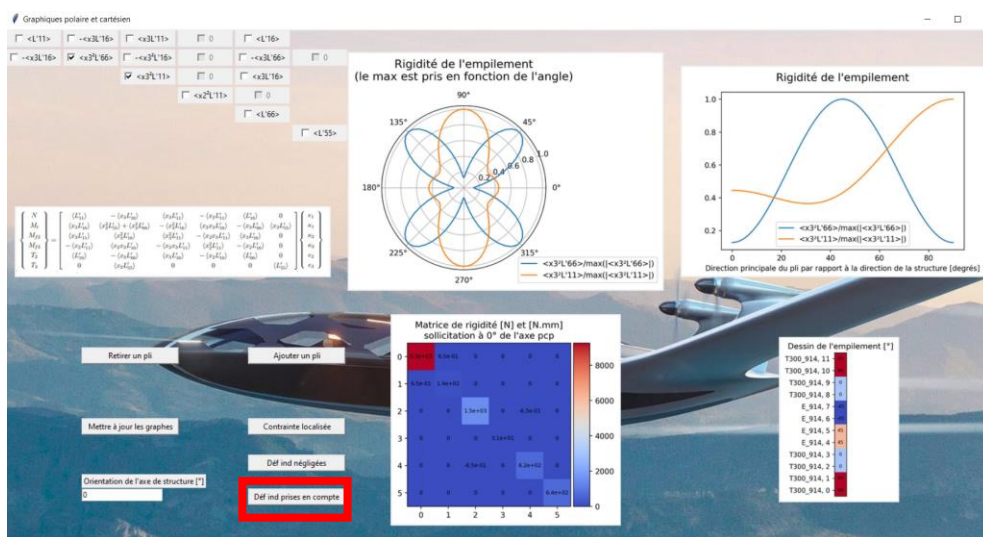


### Remarques :

- Au lancement, l'application peut se lancer en arrière-plan
- Les déformations induites sont par défaut prises en compte. On prend alors l'expression des rigidités réduites suivante pour chaque pli.

$$Q'_{ij} = L'_{ij} - \frac{L'_{i3}L'_{j3}}{L'_{33}}$$

Pour les prendre en compte, cliquer sur « def ind prises en compte » :  
L'interface ci-dessous (figure 13) se relance.



- Cliquer sur « Contrainte localisée » lance l'interface pour calculer les états de contraintes et les résistances.



## 2 Exécution en ligne de commande

Le script « test\_manu » montre comment utiliser les différentes fonctions et attributs.

Il faut commencer par décommenter les lignes plt.close( car elles permettent d'empêcher l'ouverture des figures à la fermeture de l'interface.

### 2.1 Objets Materiau, Pli et Empilement

On commence par construire des plis en utilisant le constructeur. Il faut lui fournir un matériau de type Materiau, une orientation en degré et une épaisseur en mm (le tableau 1 montre un exemple de création de plusieurs instances de pli avec des orientations différentes et des matériaux différents).

Tableau 1 : création de pli en ligne de code

Création d'instances de plis à l'aide d'instances de matériau du dictionnaire « Dict_map »	<pre> 81 # ep1 = 157.E-3 82 ep1 = 160.E-3 83 ep2 = 230.E-3 84 85 ### Plis 86 87 T300_914 = Materiau(Dict_mat["T300_914"], ep = ep1) 88 print(T300_914) 89 90 T300_914_pli_0 = Pli(Dict_mat["T300_914"], 0, ep = ep1) 91 T300_914_pli_20 = Pli(Dict_mat["T300_914"], 20, ep = ep1) 92 T300_914_pli_45 = Pli(Dict_mat["T300_914"], 45, ep = ep1) 93 T300_914_pli_45_neg = Pli(Dict_mat["T300_914"], -45, ep = ep1) 94 T300_914_pli_90 = Pli(Dict_mat["T300_914"], 90, ep = ep1) 95 T300_914_pli_90_neg = Pli(Dict_mat["T300_914"], -90, ep = ep1) 96 97 E_914_pli_0 = Pli(Dict_mat["E_914"], 0, ep = ep2) 98 E_914_pli_45 = Pli(Dict_mat["E_914"], 45, ep = ep2) 99 E_914_pli_90 = Pli(Dict_mat["E_914"], 90, ep = ep2) 100 101 print(T300_914_pli_0) 102 print(E_914_pli_0) </pre>
Affichages console	<pre> Nom=T300_914, Vf=0.4866071428571428, Em=3500.0, num=0.3, Ef=260000.0, nuf=0.33, Gflt=97700.0, ep=0.16 T300_914, 0°, [[161506.18536337 6456.26629461 6456.26629461 0. 0. 0. 6456.26629461 14476.75001154 5825.97418534 0. 0. 0. 6456.26629461 5825.97418534 14476.75001154 0. 0. 0. 0. 0. 0. 4325.3879131 0. 0. 0. 0. 5120.92955176 0. 0. 0. 0. 0. 0. 5120.92955176]], [158626.85302471 5120.92955176 5120.92955176 0. ] E_914, 0°, [[6892.8838351 2208.10284743 2208.10284743 0. 0. 0. 2208.10284743 6478.54678397 2283.76610265 0. 0. 0. 2208.10284743 2283.76610265 6478.54678397 0. 0. 0. 0. 0. 0. 2097.39034066 0. 0. 0. 0. 0. 0. 4714.85536829 0. 0. 0. 0. 4714.85536829]], [6140.28941117 4714.85536829 4714.85536829 0. ] </pre>

La matrice affichée est la matrice de rigidité d'un pli dans les axes de la structure (selon  $x_1$ ).

Pour créer un empilement manuellement, il faut d'abord créer une **liste** d'objet type Pli comme montré ligne 199. On donne ensuite cette liste comme premier argument au constructeur Empilement avec une largeur comme deuxième argument (voir tableau 2).

Tableau 2 : Création d'empilement en ligne de code

Création d'instance d'empilement à l'aide d'objet Pli	<pre>199 seq0_bis = [T300_914_pli_0] * 12 200 T300_914_emp0_bis = Empilement(seq0_bis, largeur)</pre> <p>Ou</p> <pre>230 emp_pan = Plis_from_excel(chemin_pour_sphinx('Ressources/Template_batch.xlsx'))</pre>
Affichages console	<pre>In [6]: print(T300_914_emp0_bis) Rigidité section : [[ 3.04563558e+04  0.00000000e+00  7.27595761e-12  0.00000000e+00    0.00000000e+00  0.00000000e+00]  [ 0.00000000e+00  3.02044715e+02  0.00000000e+00  0.00000000e+00   -2.72848411e-13  0.00000000e+00]  [ 7.27595761e-12 -0.00000000e+00  9.35619250e+03  0.00000000e+00    0.00000000e+00  0.00000000e+00]  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.01521186e+02    0.00000000e+00  0.00000000e+00]  [ 0.00000000e+00 -2.72848411e-13  0.00000000e+00  0.00000000e+00    9.83218474e+02  0.00000000e+00]  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00    0.00000000e+00  9.83218474e+02]] pli n°1 : T300_914, 0° pli n°2 : T300_914, 0° pli n°3 : T300_914, 0° pli n°4 : T300_914, 0° pli n°5 : T300_914, 0° pli n°6 : T300_914, 0° pli n°7 : T300_914, 0° pli n°8 : T300_914, 0° pli n°9 : T300_914, 0° pli n°10 : T300_914, 0° pli n°11 : T300_914, 0° pli n°12 : T300_914, 0°</pre>

Pour créer un empilement à l'aide d'un Excel utiliser la fonction Plis\_from\_excel après la définition de celle-ci (figure 14) :

```
230 emp_pan = Plis_from_excel(chemin_pour_sphinx('Ressources/Template_batch.xlsx'))
```

Figure 14 : création d'un empilement en ligne de commande à l'aide d'un fichier excel

## 2.2 Etats de contrainte et déformation

Pour calculer des états de contrainte, saisir les informations suivantes (efforts, géométrie) :

```

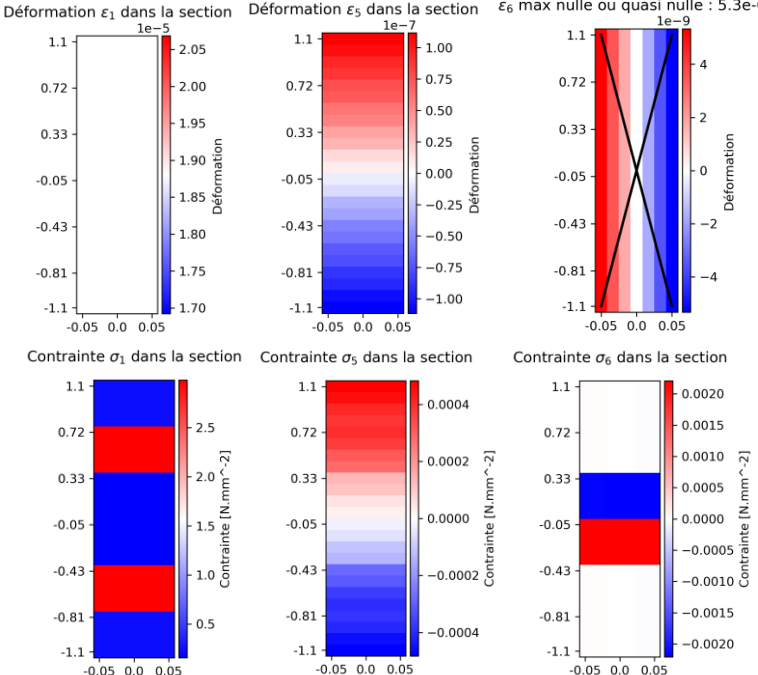
244 ''' 4/ Test du calculateur de contraintes'''
245 emp_test = emp_pan # emp_pan, T300_914_empEchec, T300_914_en
246
247 # Efforts
248 dir_sol = 0
249 Force = (1)*emp_test.get_ep_emp()*largeur
250 print(f'Force généralisée {Force} N')
251 eff_gen = [Force,0,0,0,0,0]
252 # print(f'effort généralisés: {eff_gen}')
253
254 # Géométrie
255 # alt = 3*ep1/2
256 alt = 0 # 8
257 # alt = 0.626
258 coord = [0, 0, alt]
259 taille_echant = 150
260
261 # Contraintes, déformations
262 indice_voigt = 5
263 liste_indice_voigt = [1,5,6]
264 liste_contraintes_loc_pli = []
265 nb_point = 30
266 def_ind_max = 'def_' + str(indice_voigt) + '_max'
267 cont_ind_max = 'cont_' + str(indice_voigt) + '_max'
268 methode_rupt = 'Tsai-Wu' # Tsai-Wu ou Contraintes Max
269

```

Figure 15 : Configuration du calcul des contraintes en ligne de code

Le seul paramètre modifiable dans la partie « Contraintes » (à partir de la ligne 261 de la figure 15) est la methode\_rupt. Copier-coller un des deux noms commentés à la place du nom de critère actif (ligne 268 de la figure 15). Le paramètre **nombre\_de\_point** n'est pas fonctionnel, voir la partie échantillonnage pour modifier le nombre de point où l'on calcule les déformations et contraintes dans la section.

Tableau 3 : Calcul des contraintes en ligne de code et résultats

<p>Calcul des grandeurs en un point (l. 277) dans la section (l.284) et dessin des grandeurs</p>	<pre> 272 calculateur = CalculContraintes(emp test) 273 rig = calculateur.get_rigidite.angle() 274 def_gen = calculateur.def_gen(rig, eff_gen) 275 def_memb, def_courb = calculateur.def_memb_cour(def_gen) 276 def_loc, def_loc_pli = calculateur.def_loc(eff_gen, coord) 277 sig_struct, sig_pli, pli, def_loc_struct, def_loc_pli = calculateur.contraintes_et_def_loc 278 pli_voulu = calculateur.cherche_pli(coord) 279 l_i = calculateur.li_struct(pli_voulu) 280 281 # Calculs états contraintes et déformations dans la section 282 start_time = time.time() 283 tableau_coord, hauteur_grid, largeur_grid = calculateur.tableau_coord(hb_point) 284 contraintes_section, def_section = calculateur.contraintes_et_def_section(eff_gen, table 285 end_time = time.time() 286 287 # Dessins 288 for mode in ['def', 'cont']: 289     if mode == 'def': 290         tableau = def_section 291     else: 292         tableau = contraintes_section 293     for ind_voigt in [1,5,6]: 294         CalculContraintes.dessin_section(calculateur, tableau, largeur_grid, hauteur_grid </pre>
<p>Affichages console</p>	

On crée ensuite un objet de type **CalculContraintes**. On exécute ensuite les fonctions les unes après les autres comme montré ci-dessus (tableau 3). De même il faut laisser les paramètres donnés. Voici la liste des acronymes (pour les sorties voir les docstrings des fonctions) donnée par le tableau 4 :

### 2.2.1 Glossaire acronymes

Tableau 4 : Glossaire des acronymes

Cont	Contraintes
Courb	Courbures
Def	Déformations
Gen	Généralisées
Loc	Localisées
Memb	Membrane
Struct	Structure (axe)

### 2.2.2 Représentation des valeurs importantes

Quand il s'agit d'une valeur en un point, les variables sont stockées dans un tableau 1D. Par exemple  $def_{loc} = [\varepsilon_1, \varepsilon_5, \varepsilon_6]$ .

Quand il s'agit de valeur dans une section, on a un tableau 3D dont la profondeur (troisième indice) indique l'indice de voigt correspondant et les deux premiers indices indiquent les coordonnées. Par

exemple `def_section[:, :, 0]` renvoie un tableau contenant les déformations  $\varepsilon_5$  en chaque point de la section (voir figure 16 pour la représentation des données dans une théorie des poutres).

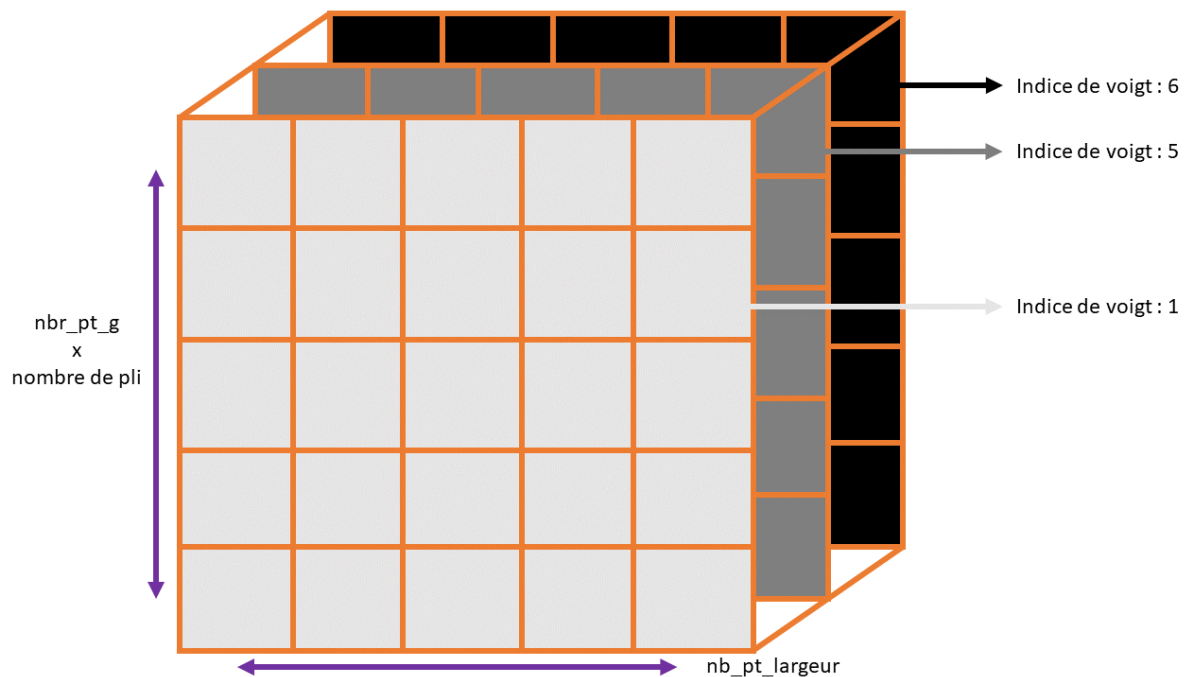


Figure 16 : Stockage de grandeurs dans la section

## 2.3 Resistances

Tableau 5 : Dessin du critère de la rupture dans la section

Dessin du critère de rupture dans la section	<pre> 316 CalculResistances.dessin_rupture(largeur_grid, hauteur_grid, calculateur.rupture) 317 318 Utils.stockage_final() </pre>
Affichages console	

Enfin, pour les résistances, on instancie la classe et on utilise uniquement la méthode de dessin car le calcul des résistances en tout point a été réalisé par la fonction **contraintes\_et\_def\_section** qui remplit le dictionnaire `rupture` qui contient également l'emplacement où le critère est maximum. C'est pour cela qu'on lui passe le dictionnaire `rupture` de l'objet `CalculContraintes`. Le pli demandé au constructeur permet de récupérer les coefficients de rupture lors du calcul du critère. Ici, on utilise seulement la fonction de classe pour dessiner la section.

On note que la ligne 318 dans le tableau 5 permet de stocker en fichiers json tous les fichiers sauvegardés temporairement dans `Utils`. Cette méthode doit donc être exécutée à la fin totale de l'utilisation du programme, elle est particulièrement utile dans le cas de l'interface où l'on génère de nombreuses fois des données à la filée.

## 3 Le code

### 3.1 Généralités

Les scripts et documents générés sont rassemblés dans un projet python (contr\_emp\_V3, montré figure 17).

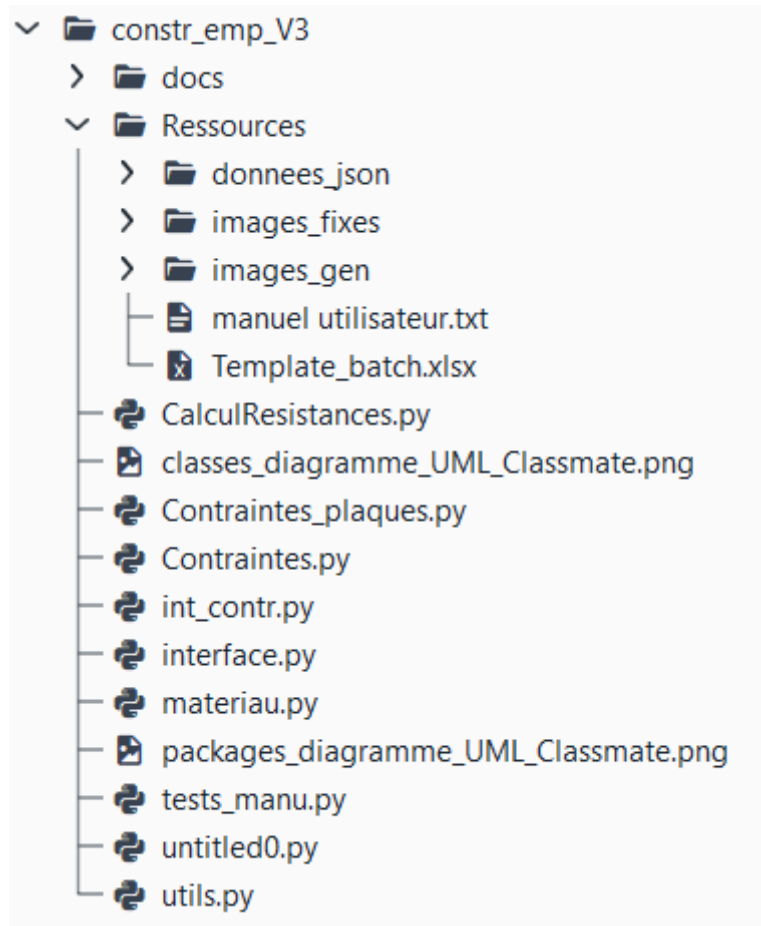


Figure 17 : arborescence du programme

Le dossier est composé de plusieurs éléments :

- Dossier docs qui contient la documentation dans build -> html -> index.html
- Dossier Ressources qui contient :
  - Un fichier Excel **Template\_batch.xlsx** à ouvrir depuis l'interface qui permet de créer des empilements avec de nombreux plis plus facilement. Son utilisation est expliquée à l'intérieur de ce dernier.
  - **Donnes\_json** pour récupérer les informations générées depuis l'interface après son utilisation.

Avant d'utiliser une fonction stockée, il faut cependant rendre à la variable son format d'origine. Pour cela, il faut utiliser la fonction **destockage** de la classe **Utils** : Inversement, pour sauvegarder une grandeur d'intérêt dans le code il suffit de saisir

```
from utils import Utils  
L_section = Utils.destockage("Ressources/donnees_json/L_section.json")
```

la commande suivante avant d'exécuter l'interface :

Utils.stockage("Nom\_de\_la\_grandeur\_a\_stocker ", grandeur\_a\_stocker)

Les différents numéros à côté de la sauvegarde correspondent à la n-ième mise à jour

de l'interface (n-ième clique sur l'interface). Par exemple, rupture\_3.json contient les données de la troisième fois où l'interface de calcul des contraintes a été mise à jour. Pour les propriétés matériau, props\_mat\_2.json par exemple correspond aux propriétés du deuxième matériau de l'empilement. Si trop de props\_mat sont présents, c'est que des empilements tests ou des plis dans tests\_manu n'ont pas été commentés.

- Un dossier **image\_gen** qui contient toutes les images générées à partir de l'interface ou d'un script utilisant les fonctions de calcul.
- Un dossier **images\_fixes** qui contient des images utiles pour le fonctionnement de l'interface
- Des classes pour calculer les propriétés matériau, les états de contrainte et les résistances. Le script **utils.py** rassemble tous les imports utiles à son utilisation et une classe contenant des attributs importants et des fonctions pour stocker les variables.
- Des scripts pour les interfaces.
- Un script utilisé pour déboguer et tester les classes de calcul seules : **tests\_manu.py**. Cette classe montre comment utiliser les fonctions des classes de calcul.

## 3.2 Modifier les graphes

### 3.2.1 Echantillonnage

Pour une théorie des poutres, les dessins de section visent à représenter une coupe transverse à l'axe  $x_1$ . La vitesse d'exécution du code dépend principalement de l'exécution de contraintes\_et\_def\_section, qui parcourt la discrétisation de la section et effectue les calculs de concentration et localisation, du critère de rupture en chacun des points parcourus, dans les axes de la structure et du pli. Un levier pour diminuer le temps de calcul est donc de diminuer la discrétisation de la section, qui se fait au détriment d'un profil des contraintes, déformations et rupture moins riche.

Deux options d'échantillonnage sont possibles à modifier dans **tableau\_coord** de **Contraintes.py** :

**La deuxième méthode d'intégration est préférée, car elle assure de calculer les valeurs en chaque pli.**

1. On fixe un nombre  $n$  de point dans la hauteur (indépendant du nombre de pli) et un nombre de point  $nb\_pt\_largeur$  dans la largeur.

$n$  est donné en argument de la fonction et  $nb\_pt\_largeur$  est un argument de la fonction. Les deux sont donc modifiables.

Pour utiliser cette méthode, il suffit de décommenter la ligne de commande entre #1 et #2 à l'intérieur de **tableau\_coord**. Et commenter les lignes entre #2 et # Create coordinate grids (voir figure 18).

```

103     # Plusieurs types de tableau
104     #1
105     hauteur_table = np.linspace(-H, H, n) # n valeurs réparties uniformément adns la hauteur
106     #2
107     # hm = -H
108     # hauteur_table = np.zeros(len(self.Emplacement.liste_de_pli)*nbr_pt_g) # une hauteur par pli
109     # k = 0
110     # while k < len(self.Emplacement.liste_de_pli):
111     #     for i in range(nbr_pt_g):
112     #         hauteur_table[k*nbr_pt_g+i] = hm + (i+1)*(self.Emplacement.liste_de_pli[k].props["ep"]
113     #     hm += self.Emplacement.liste_de_pli[k].props["ep"]
114     #     k += 1
115
116     # Create coordinate grids
117     hauteur_grid, largeur_grid= np.meshgrid(hauteur_table, largeur_table, indexing='ij')
118

```

Figure 18 : nombre de points de calcul fixés sur l'empilement

**Attention :** Cette méthode ne fonctionne que pour un nombre  $n$  qui est pair et est un multiple du nombre de pli, sinon cette méthode calculera les quantités plus de fois pour certains plis.



## 2. On fixe un nombre de point d'intégration par pli selon la hauteur.

Ici, le nombre de point d'intégration dans la hauteur est lié au pli et il y en a `nbr_pt_g` par pli. Cet attribut est modifiable directement dans la fonction **tableau\_coord**.

Il s'agit ensuite de commenter la ligne de commande entre #1 et #2 à l'intérieur de **tableau\_coord**. Et décommenter les lignes entre #2 et # Create coordinate grids.

```

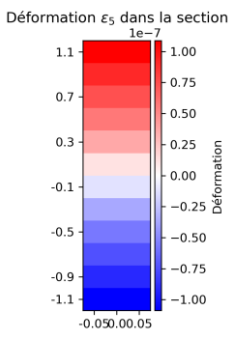
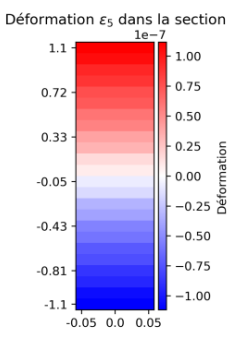
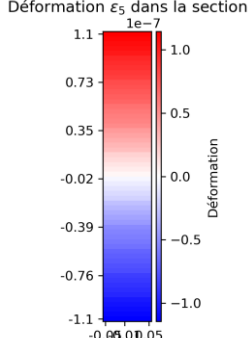
103 # Plusieurs types de tableau
104 #1
105 # hauteur_table = np.linspace(-H, H, n) # n valeurs réparties uniformément adns la hauteur
106 # 2
107 hm = -H
108 hauteur_table = np.zeros(len(self.Empilement.liste_de_pli)*nbr_pt_g) # une hauteur par pli
109 k = 0
110 while k < len(self.Empilement.liste_de_pli):
111     for i in range(nbr_pt_g):
112         hauteur_table[k*nbr_pt_g+i] = hm + (i+1)*(self.Empilement.liste_de_pli[k].props["ep"])/(nbr_p
113         hm += self.Empilement.liste_de_pli[k].props["ep"]
114         k += 1
115
116 # Create coordinate grids
117 hauteur_grid, largeur_grid= np.meshgrid(hauteur_table, largeur_table, indexing='ij')

```

Figure 19 : Nombre de calcul fixés par pli

Voici quelques valeurs de vitesse d'exécution pour la méthode 2. où l'on fixe un nombre de point d'intégration par pli :

Tableau 6 : comparaison de vitesse d'exécution du code pour différents nombre de point d'intégration

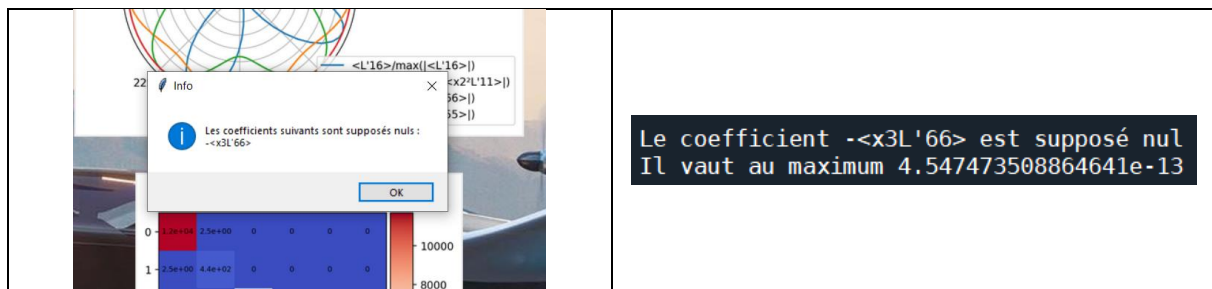
Comparaison de vitesse d'exécution			
	nb_pt_largeur = 3 nbr_pt_g = 1	Par défaut nb_pt_largeur = 7 nbr_pt_g = 2	nb_pt_largeur = 10 nbr_pt_g = 5
Temps d'exécution (s)	0.053951263427734375	0.2998218536376953	0.8118724822998047
Dessin de la section			

### 3.2.2 Seuils

Dans Utils, on retrouve deux types de seuils. Tous les coefficients de la matrice de rigidité ou les valeurs de contrainte et déformation en dessous de cette valeur sont considérés comme nuls car sont trop faible pour avoir une signification physique :

- Un seuil nommé **seuil** pour les graphes polaires ou cartésiens qui est comparé aux coefficients de la matrice de rigidité. Lorsqu'une valeur est en dessous du seuil, elle est signalée dans la console et par une message box dans l'interface. On affiche également la valeur maximum de ce coefficient parmi toutes les orientations. Le tableau ci-dessous (tableau 7) montre les effets d'un coefficient considéré comme nul.

Tableau 7 : incidence d'un coefficient nul sur l'interface (à gauche) et la console (à droite)



- Deux seuils nommés **offset\_eps** et **offset\_sig** qui sont comparés respectivement à la déformation maximum et à la contrainte maximum (dans les axes de la structure) dans la section. Lorsque la valeur est en dessous de ce seuil, la section est barrée et le titre affiche la valeur maximale dans la section (voir figure 20).

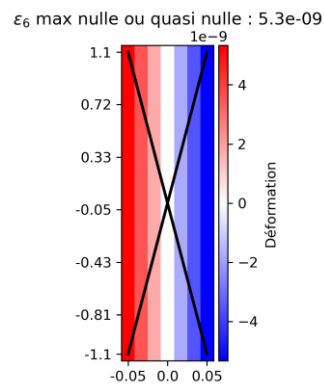


Figure 20 : influence de l'offset sur les dessins

Pour modifier ces valeurs, il suffit de les modifier directement dans la classe Utils (voir figure 21) :

```

36 class Utils:
37
38     # Type théorie
39     plaque = 0 # Si 0 -> théorie de
40     def3_induite = 1 # Si 1 -> défo
41
42     # Seuils
43     seuil = 10**(-10) # Les coeffi
44     offset_eps = 10**(-7) # Seuils
45     offset_sig = 10**(-10)
46

```

Figure 21 : Seuil (ligne 43) et offset (lignes 44 et 45) et paramètres permettant de choisir la théorie (ligne 39) et la prise en compte de la déformation transverse (l.40)

### 3.2.3 Ouverture automatique des graphes

Lors d'une exécution en ligne de code, il peut être utile d'afficher les diagrammes directement dans l'onglet plot, ainsi il faut décommenter les lignes `plt.close()` à la fin des méthodes d'affichage.

Au contraire, quand on exécute l'interface et que l'on génère à plusieurs reprises des graphes, si ces lignes sont actives elles vont générer l'apparition des graphes tour à tour après la fermeture de l'interface.

Ces plt.close se trouvent dans le script **Contraintes.py** et **Materiau.py**. Faire ctrl+f pour les rechercher.

### 3.3 Autres variables importantes

#### 3.3.1 Mat\_map

Le tableau Mat\_map permet de faire l'intermédiaire entre les checkboxes de l'interface et les différentes informations physiques.

L'indice ou l'ID de la checkbox est représenté par l'indice de la colonne du tableau. Et la ligne correspond à un type d'information (voir figure 22 ci-dessous).

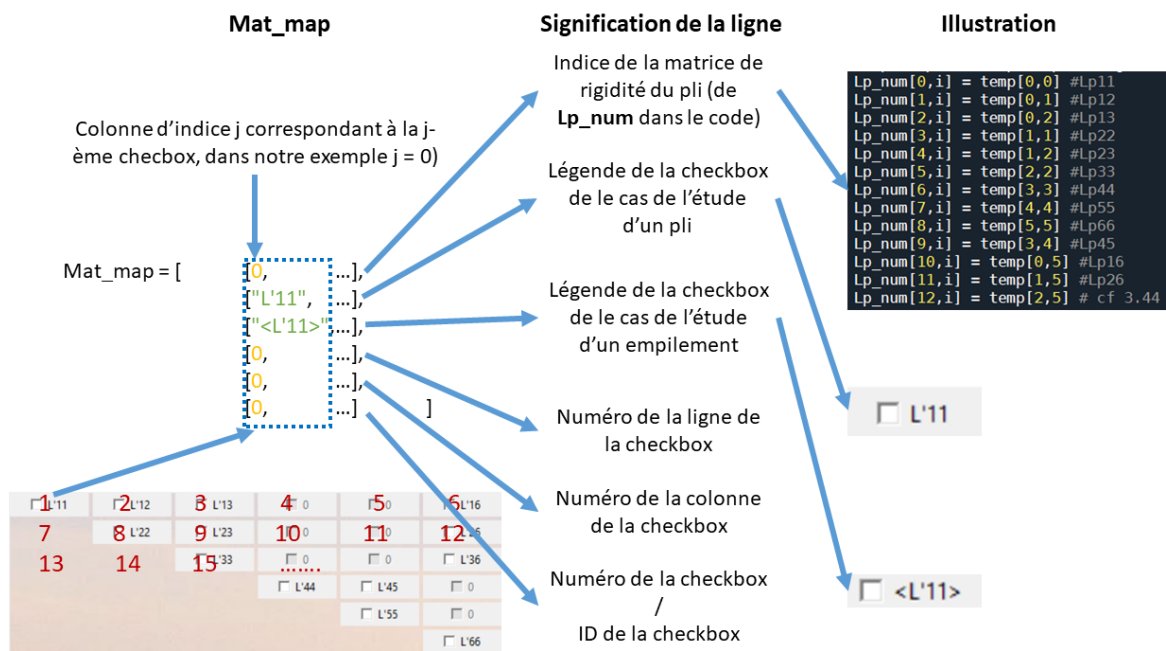


Figure 22 : Explication du dictionnaire Mat\_map faisant le lien entre la numérotation des checkboxes de l'interface et la numérotation des grandeurs dans les calculs

#### 3.3.2 Dict\_mat

**Dict\_mat** est un dictionnaire qui contient les informations sur les matériaux. La clé correspond au nom du matériau et la valeur à un dictionnaire contenant ses caractéristiques (voir figure 23 pour la structure de ce dernier).

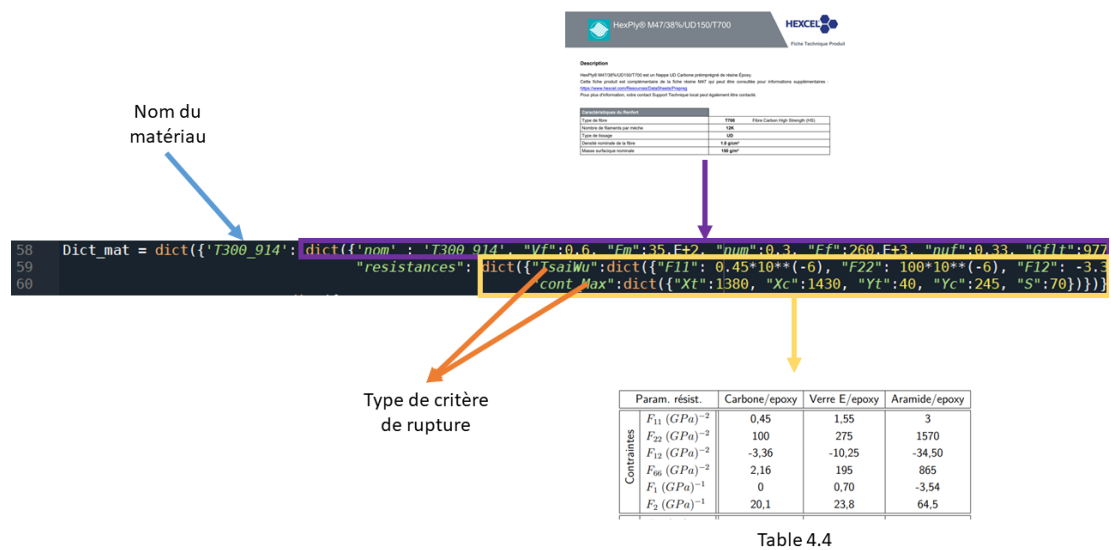


Figure 23 : Structure du dictionnaire des matériaux

## 4 Index

### 4.1 Index des figures

Figure 1 : interface de départ .....	3
Figure 2 : choix du matériau .....	3
Figure 3 : interface présentant les caractéristiques d'un pli de fibre de verre .....	4
Figure 4 : initialisation de la séquence pour un empilement .....	4
Figure 5 : Excel représentant un empilement .....	4
Figure 6 : interface de calcul de propriétés d'un empilement stratifié avec la théorie des poutres .....	5
Figure 7 : interface pour retirer des plis .....	5
Figure 8 : interface pour ajouter un pli .....	6
Figure 9 : interface des propriétés d'un empilement avec la théorie des plaques .....	6
Figure 10 : emplacement de l'ouverture de l'interface de calcul de contraintes .....	7
Figure 11 : interface de calcul de contrainte (théorie des poutres), non exécutée .....	7
Figure 12 : composition de l'interface pour calculer les contrainte (théorie des poutres ici) .....	8
Figure 13 : interface qui se relance lors de la prise en compte ou non des déformations transverses ..	8
Figure 14 : création d'un empilement en ligne de commande à l'aide d'un fichier excel .....	10
Figure 15 : Configuration du calcul des contraintes en ligne de code .....	11
Figure 16 : Stockage de grandeurs dans la section .....	13
Figure 17 : arborescence du programme .....	14
Figure 18 : nombre de points de calcul fixés sur l'empilement .....	15
Figure 19 : Nombre de calcul fixés par pli .....	16
Figure 20 : influence de l'offset sur les dessins .....	17
Figure 21 : Seuil (ligne 43) et offset (lignes 44 et 45) et paramètres permettant de choisir la théorie (ligne 39) et la prise en compte de la déformation transverse (l.40) .....	17
Figure 22 : Explication du dictionnaire Mat_map faisant le lien entre la numérotation des checkboxes de l'interface et la numérotation des grandeurs dans les calculs .....	18
Figure 23 : Structure du dictionnaire des matériaux .....	19

### 4.2 Index des tableaux

Tableau 1 : création de pli en ligne de code .....	9
Tableau 2 : Création d'empilement en ligne de code .....	10
Tableau 3 : Calcul des contraintes en ligne de code et résultats .....	12
Tableau 4 : Glossaire des acronymes .....	12
Tableau 5 : Dessin du critère de la rupture dans la section .....	13
Tableau 6 : comparaison de vitesse d'exécution du code pour différents nombre de point d'intégration .....	16
Tableau 7 : incidence d'un coefficient nul sur l'interface (à gauche) et la console (à droite) .....	17