Etude des remplissages à densité variable en impression 3D

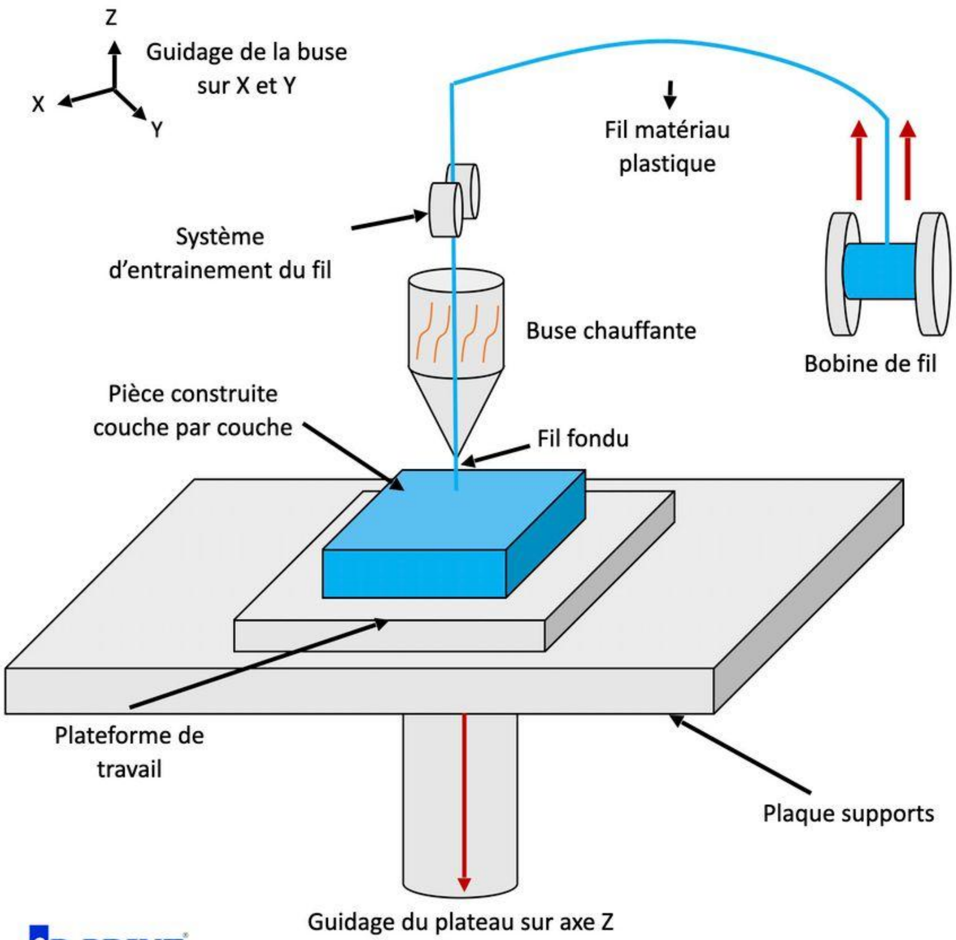
Oscar THOMAS

11717

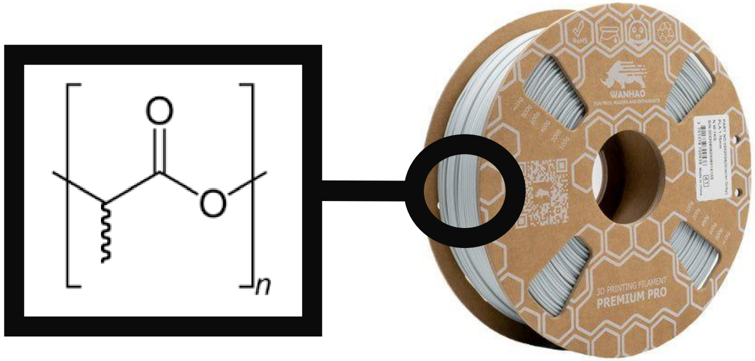
Florian VERNE

11215

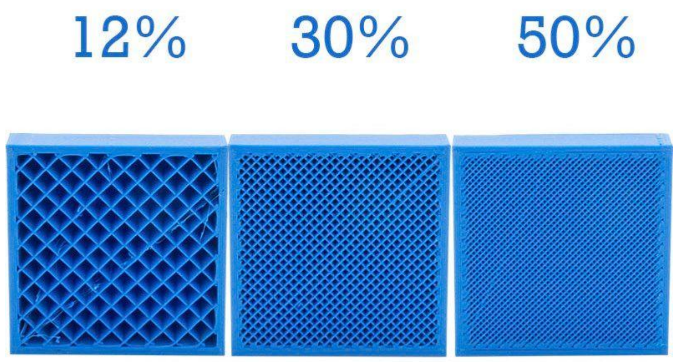
Introduction



Dépôt de fil (FDM)



Bobine de PLA



Problématique

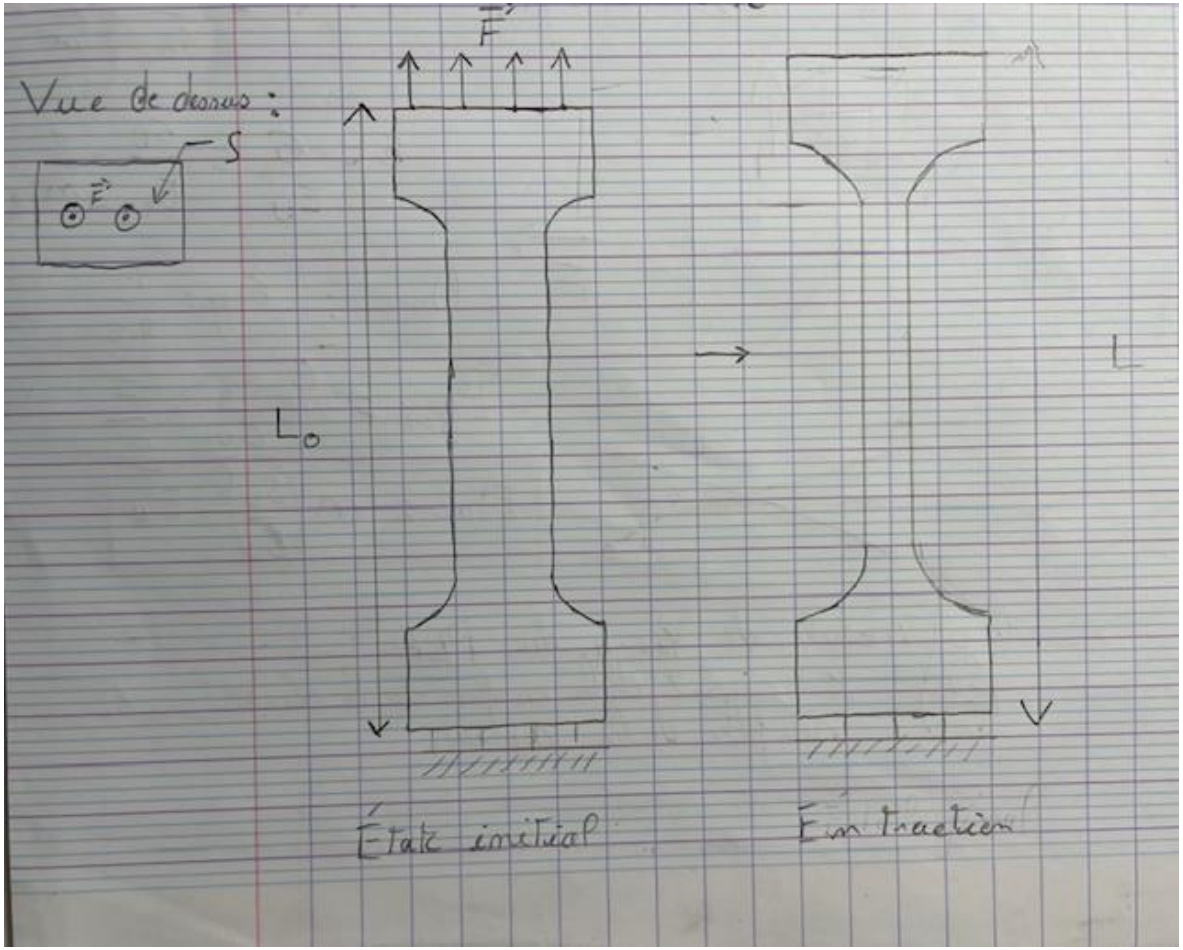
Problématique :

En quoi la modification du remplissage d'une pièce imprimée peut-elle influer sur ses propriétés mécaniques ?

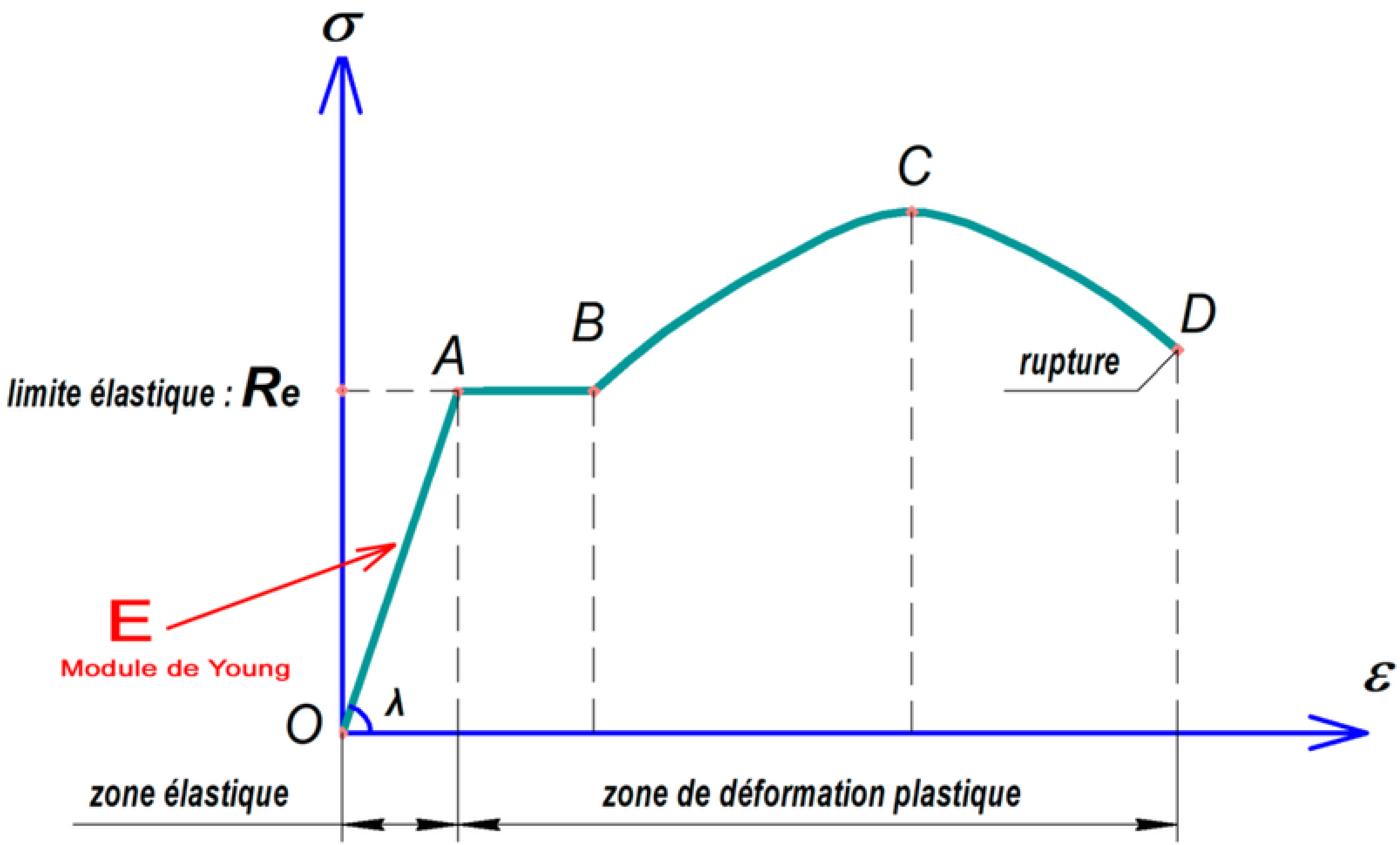
Sommaire

<u>Loi de Hooke</u> : σ =E.ε

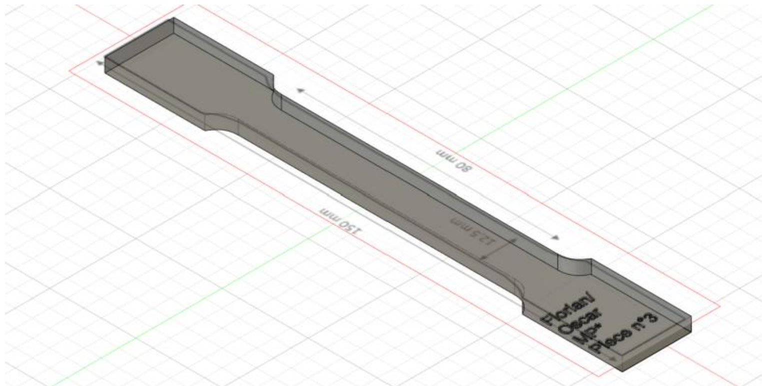
-σ = F/S contrainte en Pa.
-E module d'élasticité de Young en Pa.
-$\varepsilon = (L-L_0)/L_0$ allongement relatif.



Elongation en réponse à une traction uni-axiale :



I.1 Implémenter un remplissage

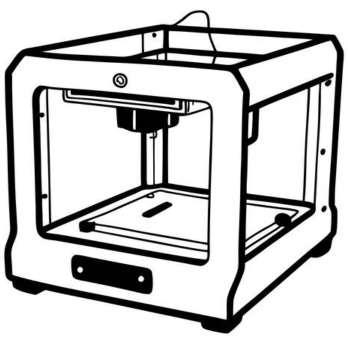


Nouveau remplissage

Fillpavage.cpp

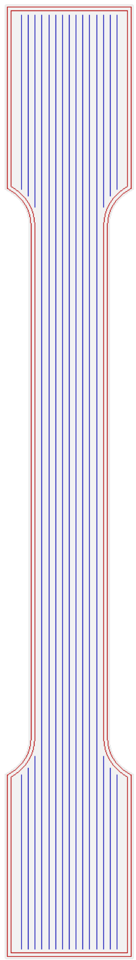
Slic3r

G-code

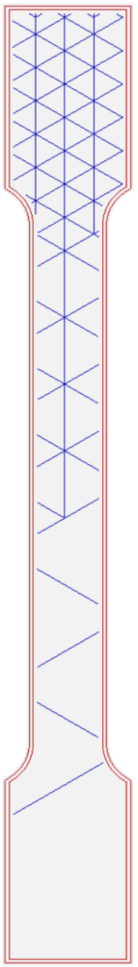


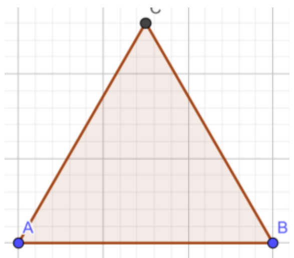
I.1 Implémenter un remplissage

Remplissage linéaire : ci-dessous exemple de remplissage à 9 lignes

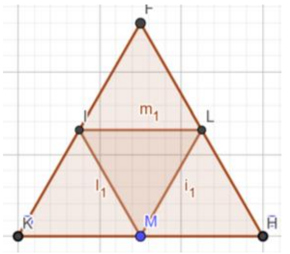


Remplissage pavage triangulaire : ci-dessous profondeur 2,3,4





Profondeur 0



Profondeur 1

I.2 Présentation du protocole d'essai en traction

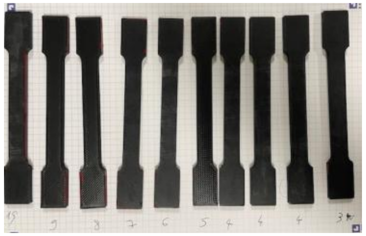
Dispositif expérimental :

Effort appliqué

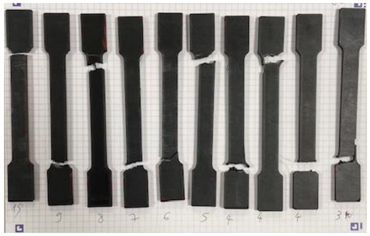
Machine d'essai en traction du lycée Livet (Nantes).

Eprouvettes testées :
-En haut : remplissage pavage triangulaire
-En bas : remplissage lignes

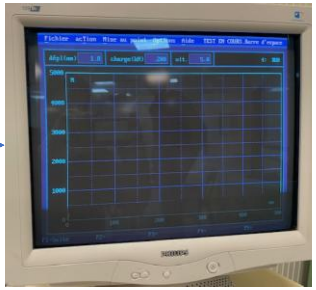
Etat initial :



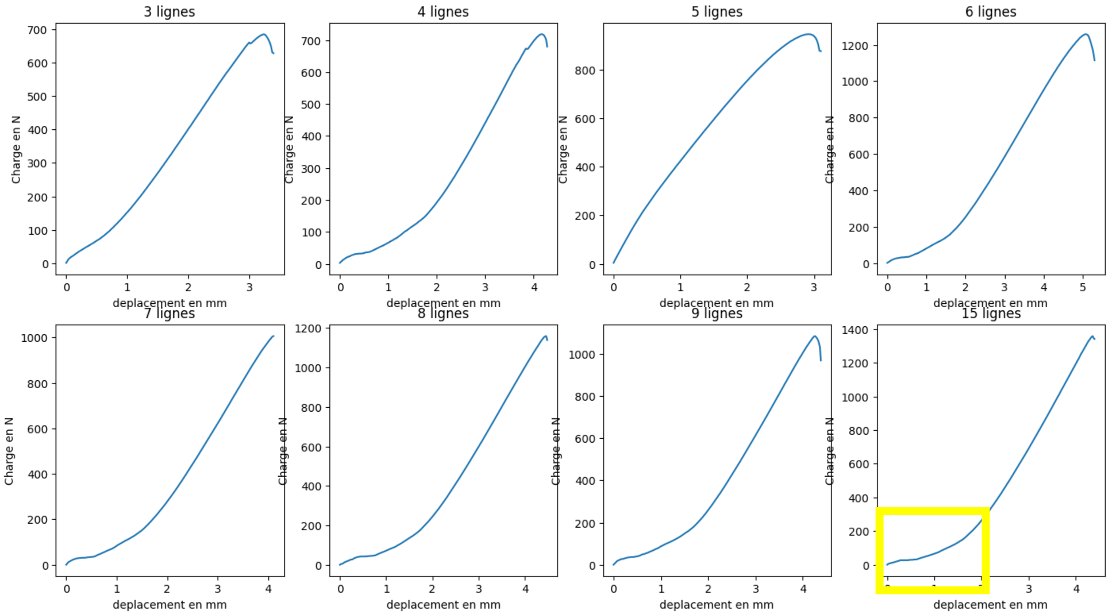
Etat final :

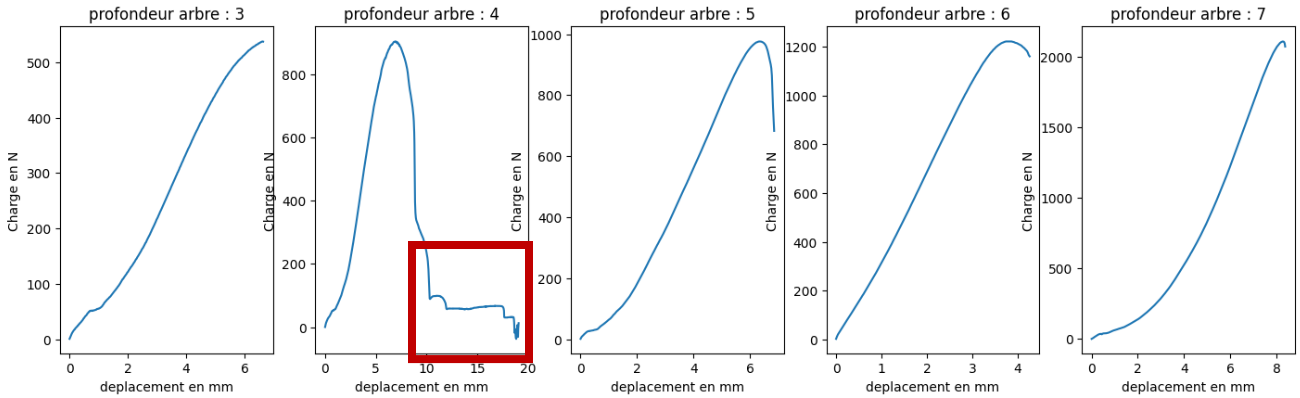


Mesure de l'élongation et de la force appliquée.
Résultats renvoyés au format .asc



I.3 Bilan de cette première expérience



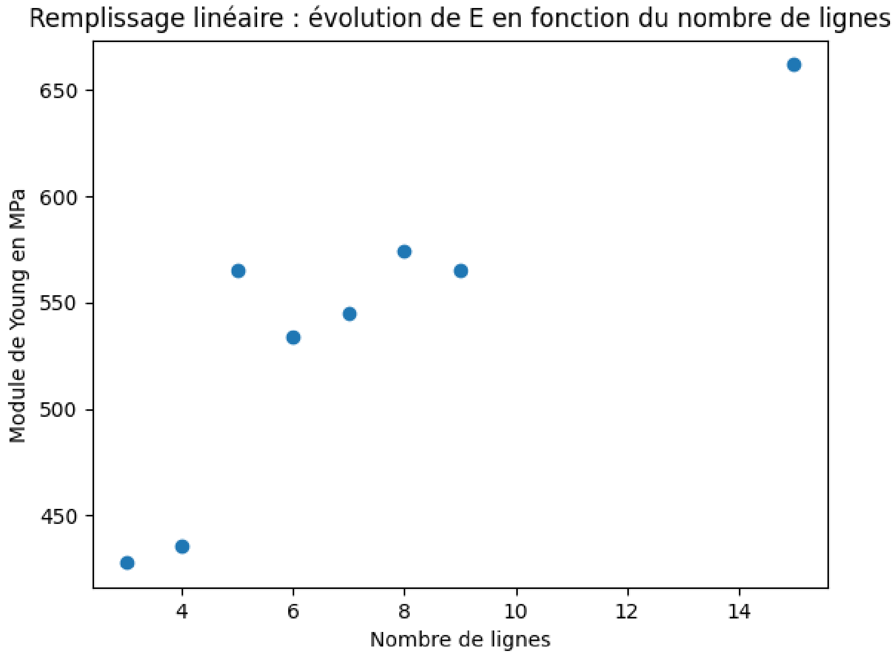


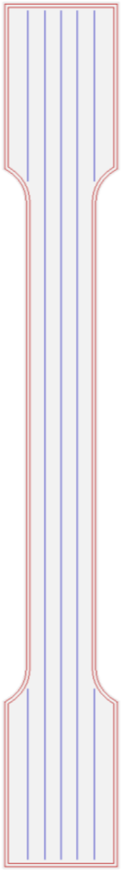
Observations :

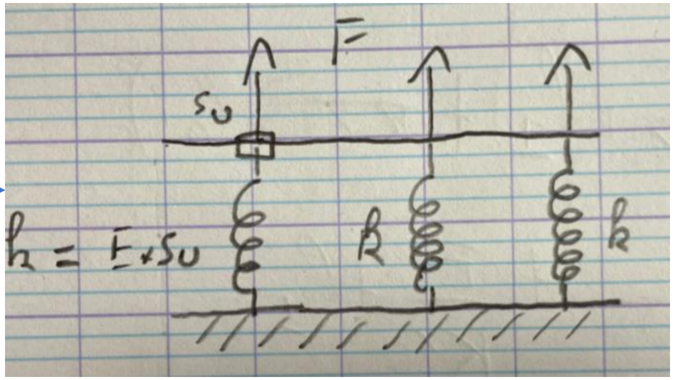
-Comportement hookéen.
-Pas de domaine plastique.
-Un remplissage plus dense donne une contrainte à la rupture plus forte.
-Un remplissage plus dense donne un module de Young (E) plus élevé.

Quelques défauts :
: zone de glissement
: rupture incomplète

I.3 Bilan de cette première expérience

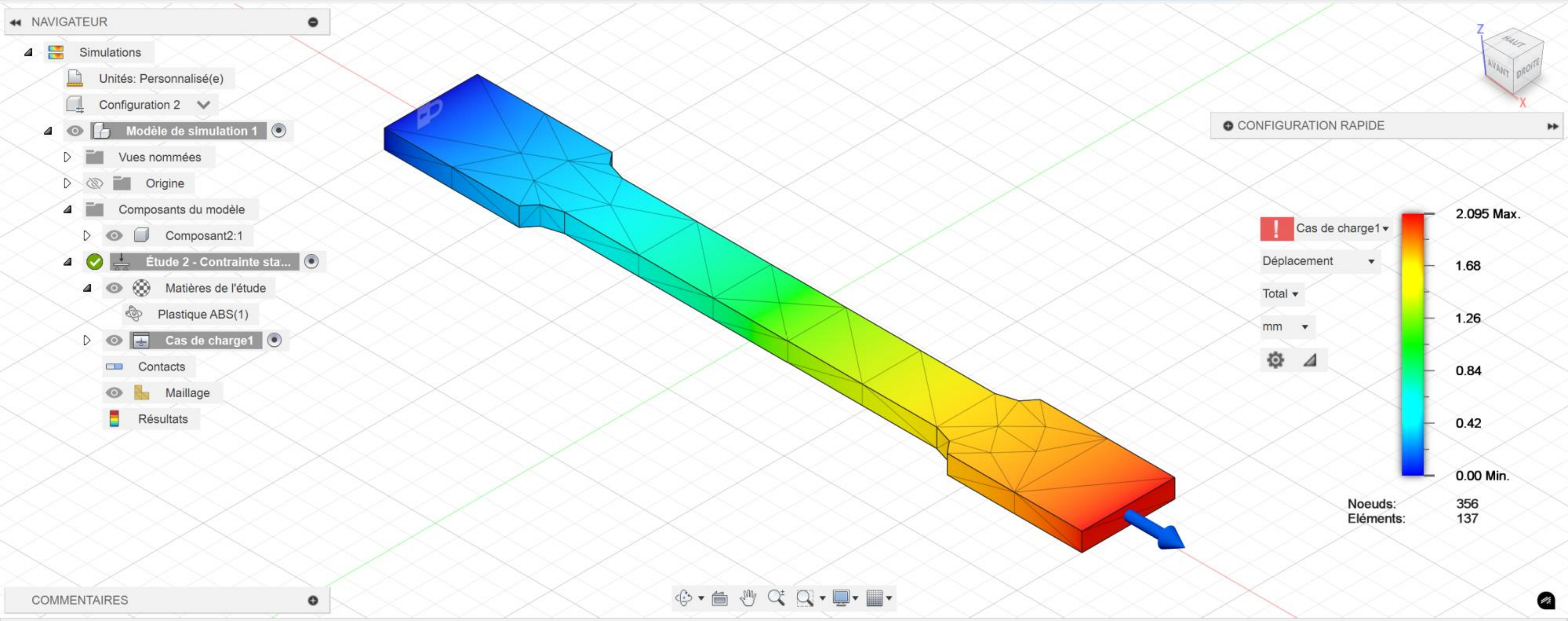






Sur la figure ci-dessus :
-E a été calculé par régression linéaire.
-On vérifie que le remplissage se comporte comme un assemblage de ressorts en parallèle.

II.1 Reproduction d'un essai traction par la FEM



Réalisation d'une FEM sur une éprouvette à l'aide du logiciel Fusion 360 :

-L'éprouvette de PLA est soumise à une charge de 3 kN sur sa face droite.
-La face gauche est quand à elle forcée de ne pas bouger.

II.2 Mécanique locale

Lors d'une transformation on définit :
- le vecteur déplacement u
- le tenseur de déformation
- le tenseur de contrainte

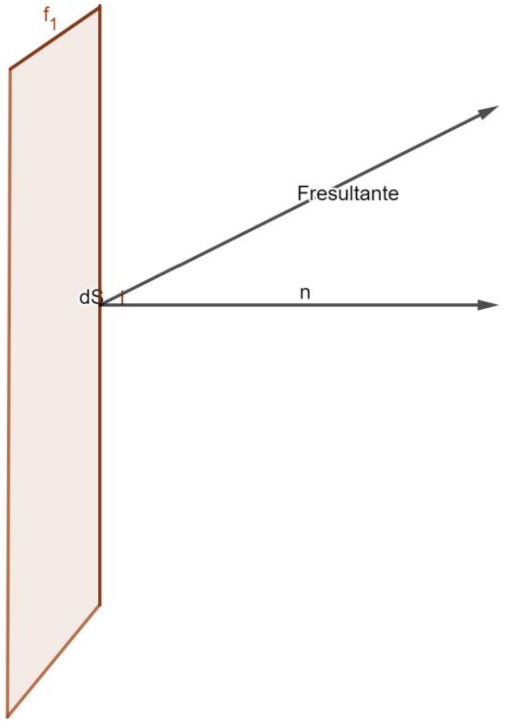
$$u : (x_1, x_2) \mapsto (u_1, u_2)$$

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

$$\underline{\sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}$$

La contrainte est la généralisation de la notion de pression.

$$\overrightarrow{dF} = \underline{\sigma} \cdot \overrightarrow{dS}$$



II.2 Mécanique locale

Loi de Hooke locale : σ = C : ε

- C tenseur des rigidités élastiques
- ‘:’ est l’opérateur produit tensoriel
- v est le coefficient de Poisson

Représentation matricielle:

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\nu \end{pmatrix} \cdot \begin{pmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{12} \end{pmatrix}$$

II.2 Critère de Von Mises

C'est un critère énergétique.
La pièce ne peut stocker qu'une quantité limitée d'énergie lors du changement de forme.

Formulaire :

$$tr(A \cdot B^T) = \sum_{1 \leq i,j \leq n} a_{ij} b_{ij}$$

Energie élastique locale :

$U = \frac{1}{2} tr(\underline{\sigma} \cdot \underline{\epsilon}^T)$

$U = U_f + U_v$ avec $\boxed{U_f = \frac{1}{2} tr(dev(\underline{\sigma}) \cdot \underline{\epsilon}^T)}$ et $U_v = \frac{1}{2} tr(iso(\underline{\sigma}) \cdot \underline{\epsilon}^T)$

En définissant : $iso(\underline{\sigma}) = \frac{1}{2} tr(\underline{\sigma}) I_2$, $dev(\underline{\sigma}) = \underline{\sigma} - iso(\underline{\sigma})$

U_f est l'énergie liée au changement de forme.
On peut démontrer :

$U_f = \frac{1+\nu}{2E} ((\sigma_{11} - \sigma_{22})^2 + 2\sigma_{12}^2)$

$U_f = \dfrac{1+\nu}{2E} \sigma_{vonMisses}^2$ avec $\boxed{\sigma_{vonMisses} = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 2\sigma_{12}^2}}$

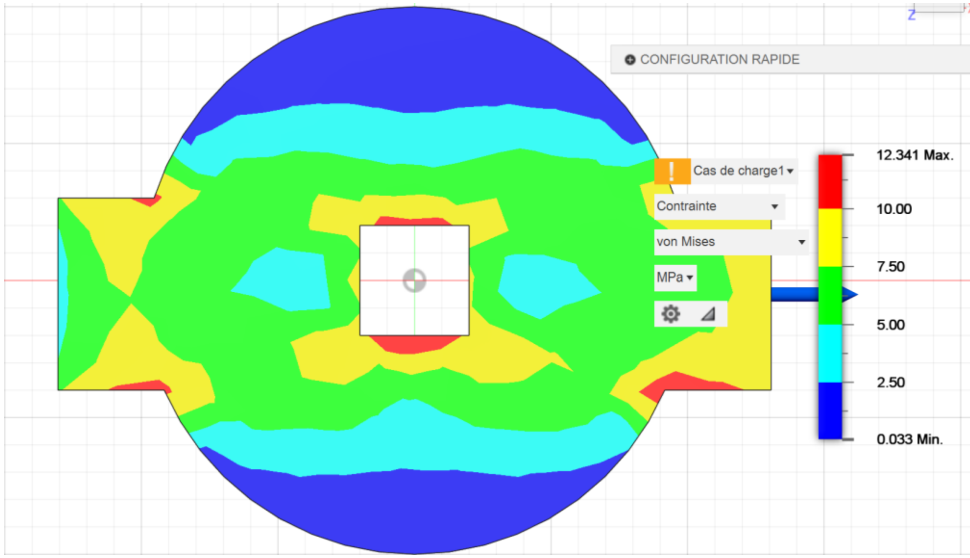
Le critère se résume à :

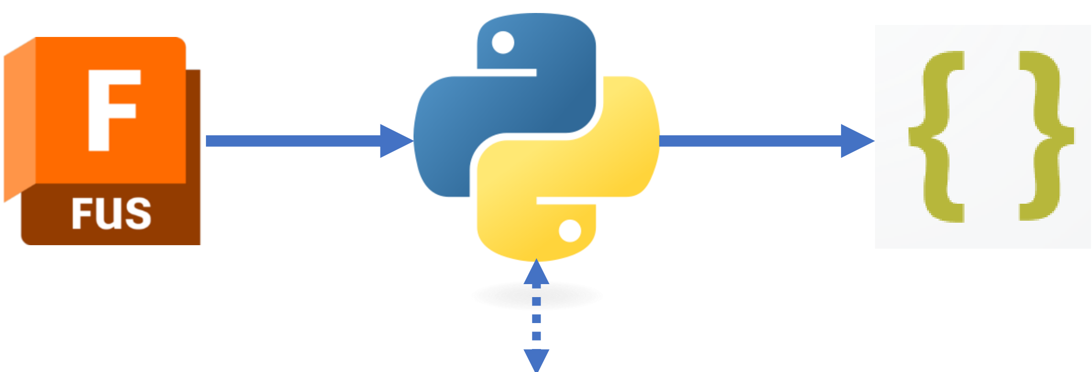
$U_f \leq U_{lim}$ et donc $\sigma_{vonMisses} \leq \sigma_{lim}$

II.3 Génération d'un remplissage variable

Sur la figure ci-contre :

-Les contraintes de von Mises sont affichées en couleurs
-Plus la couleur est proche du rouge, plus la contrainte est forte. Il faut une densité plus importante.



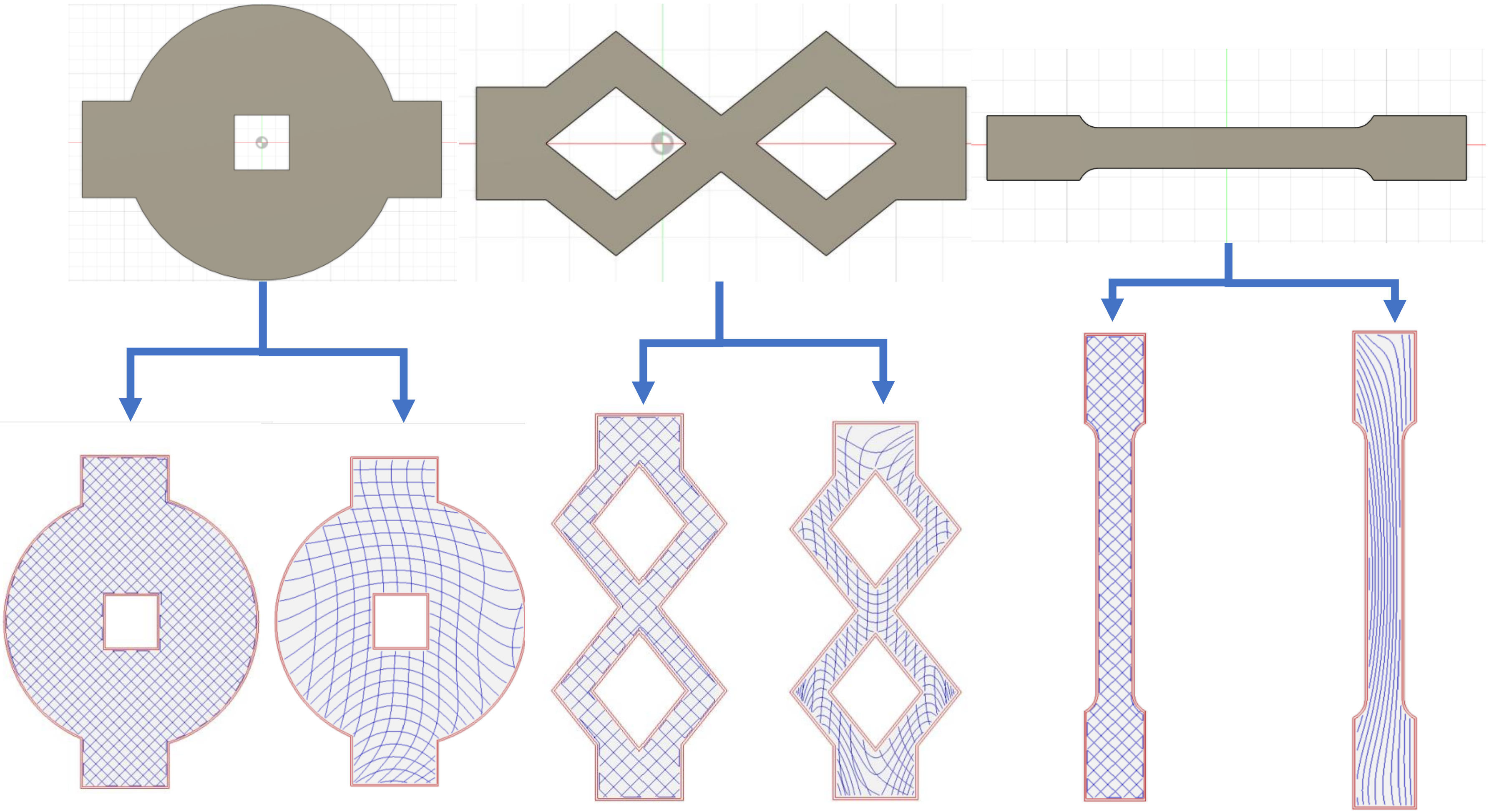


Fichier json lu par Slic3r.

Génération d'un polynôme représentant la contrainte en fonction des coordonnées.

III. Comparatif des remplissages

Comparaison des remplissages sur trois pièces différentes.



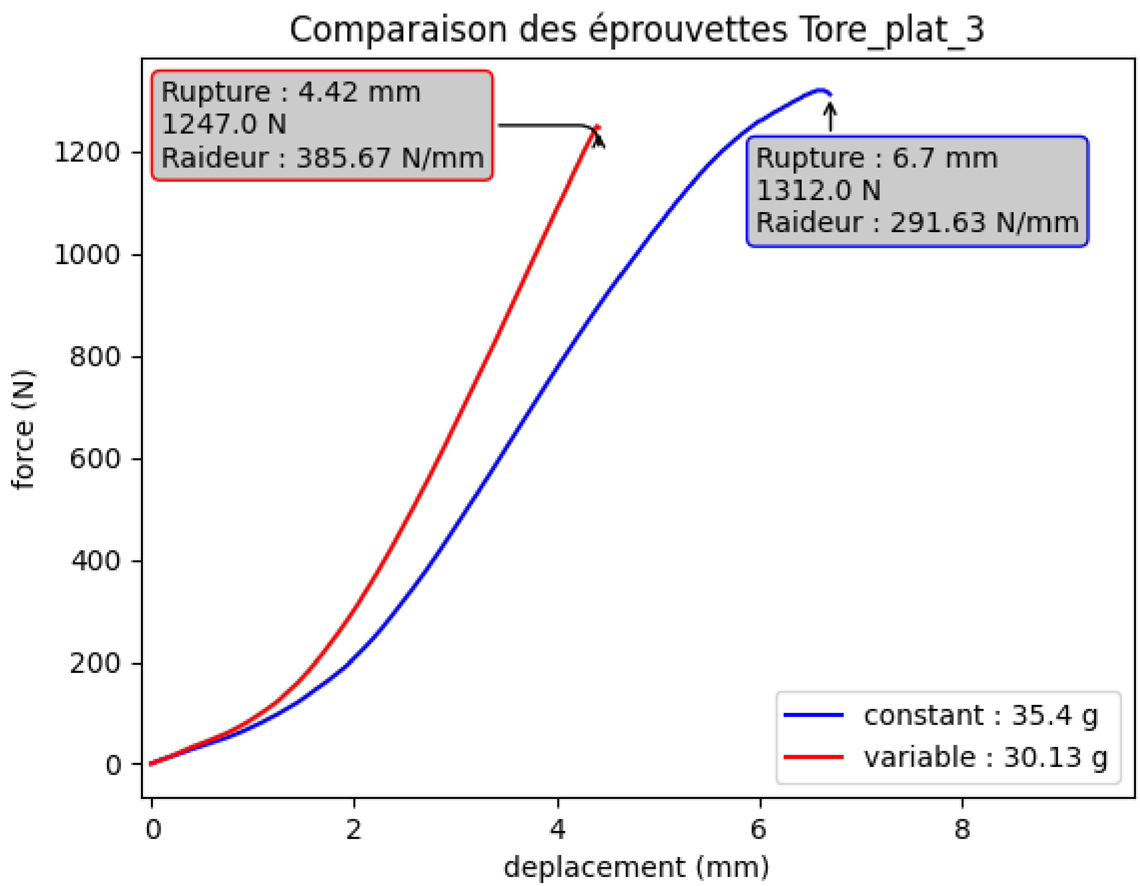
III.1 Présentation de l’expérience

Comportement local étudié en prenant une vidéo de la pièce. En analysant le déplacement de chaque point jaune, on en déduit le champ de déformation.

Comportement global de la pièce mesuré par la cellule de force et l’extensomètre.

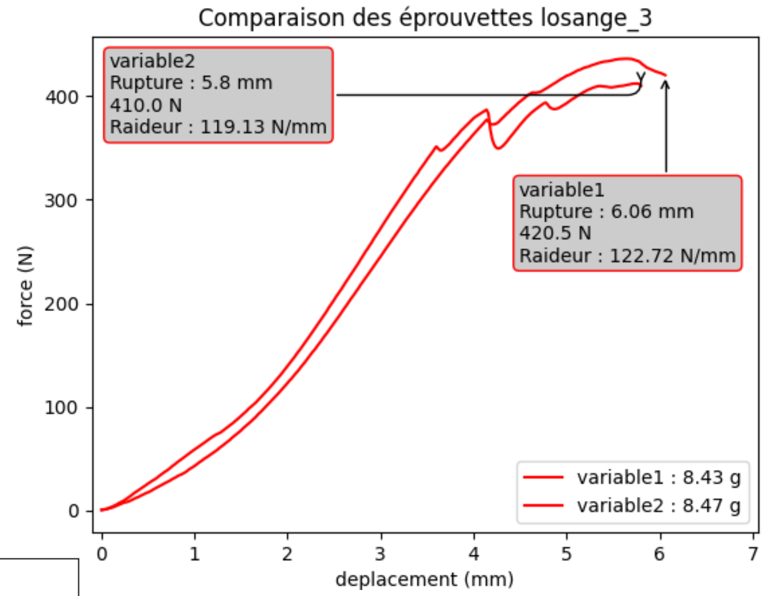
III.2 Résultats

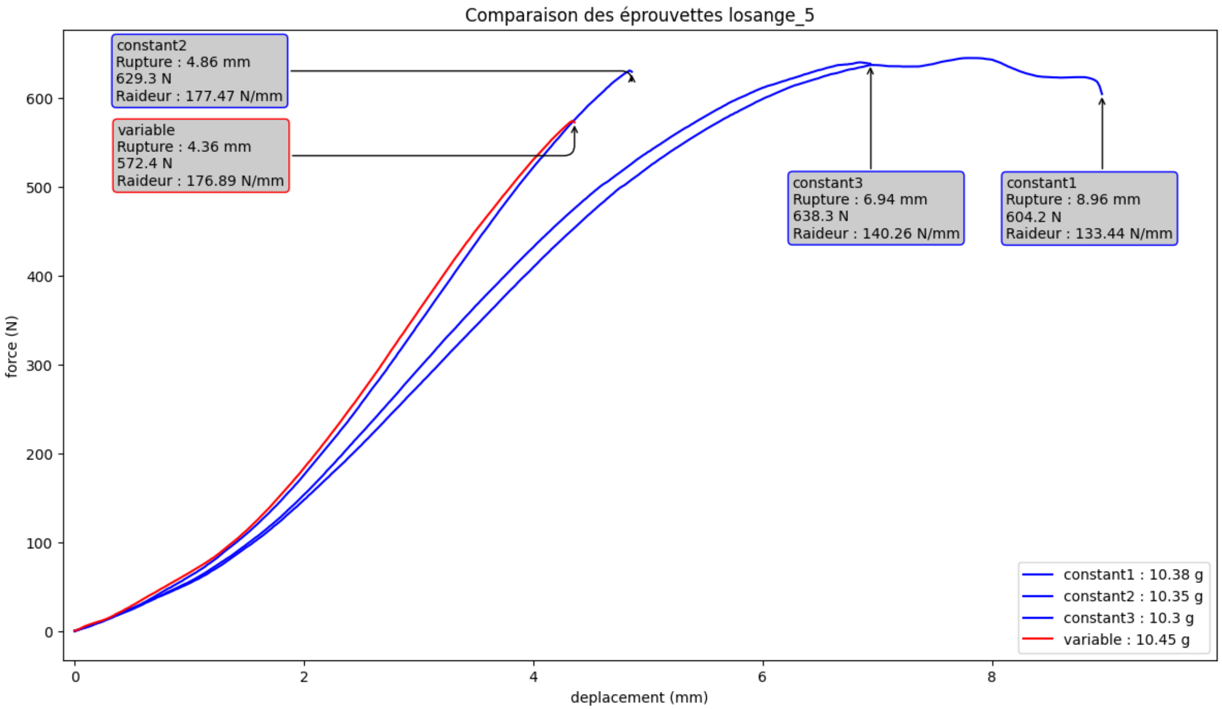
Les différents graphes montrent la réponse en traction de pièces similaires. Ils sont tracés grâce à Python.



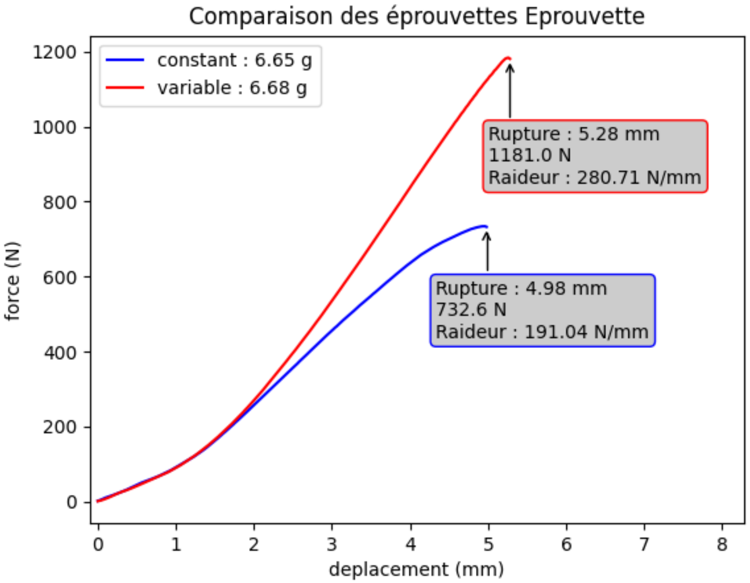
III.2 Résultats

A droite deux éprouvettes type losange identiques ont été testées.



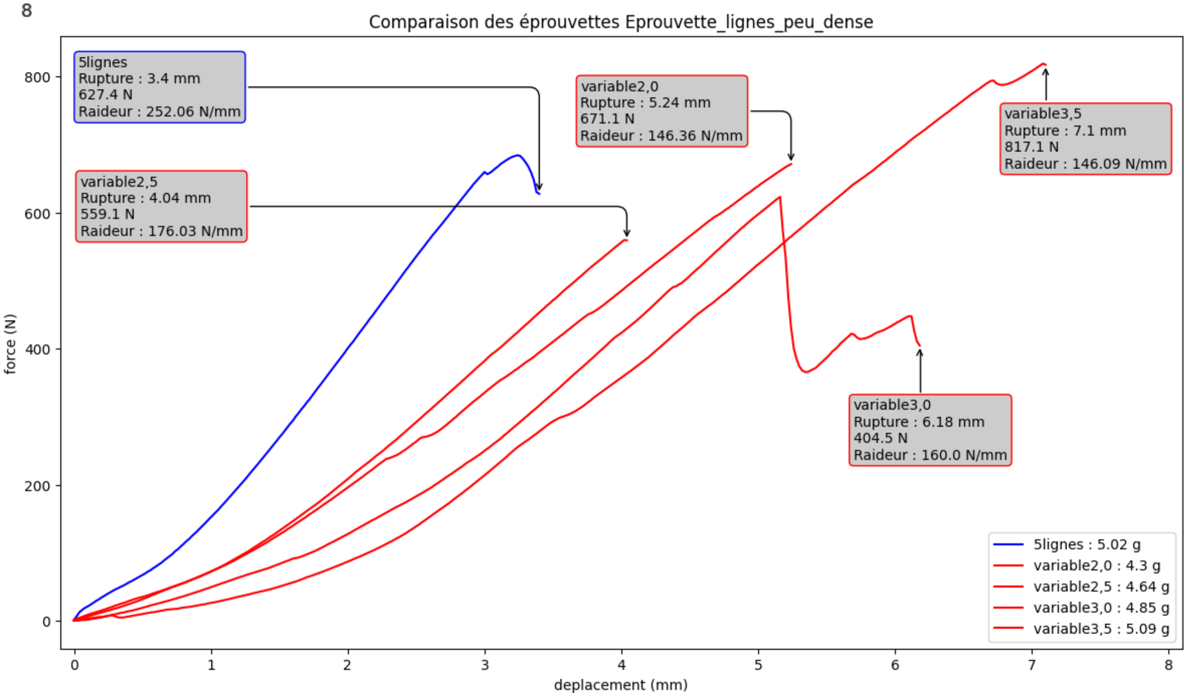


Sur les losanges, le changement de remplissage n'a pas de grande influence. En effet, le remplissage est peu dense

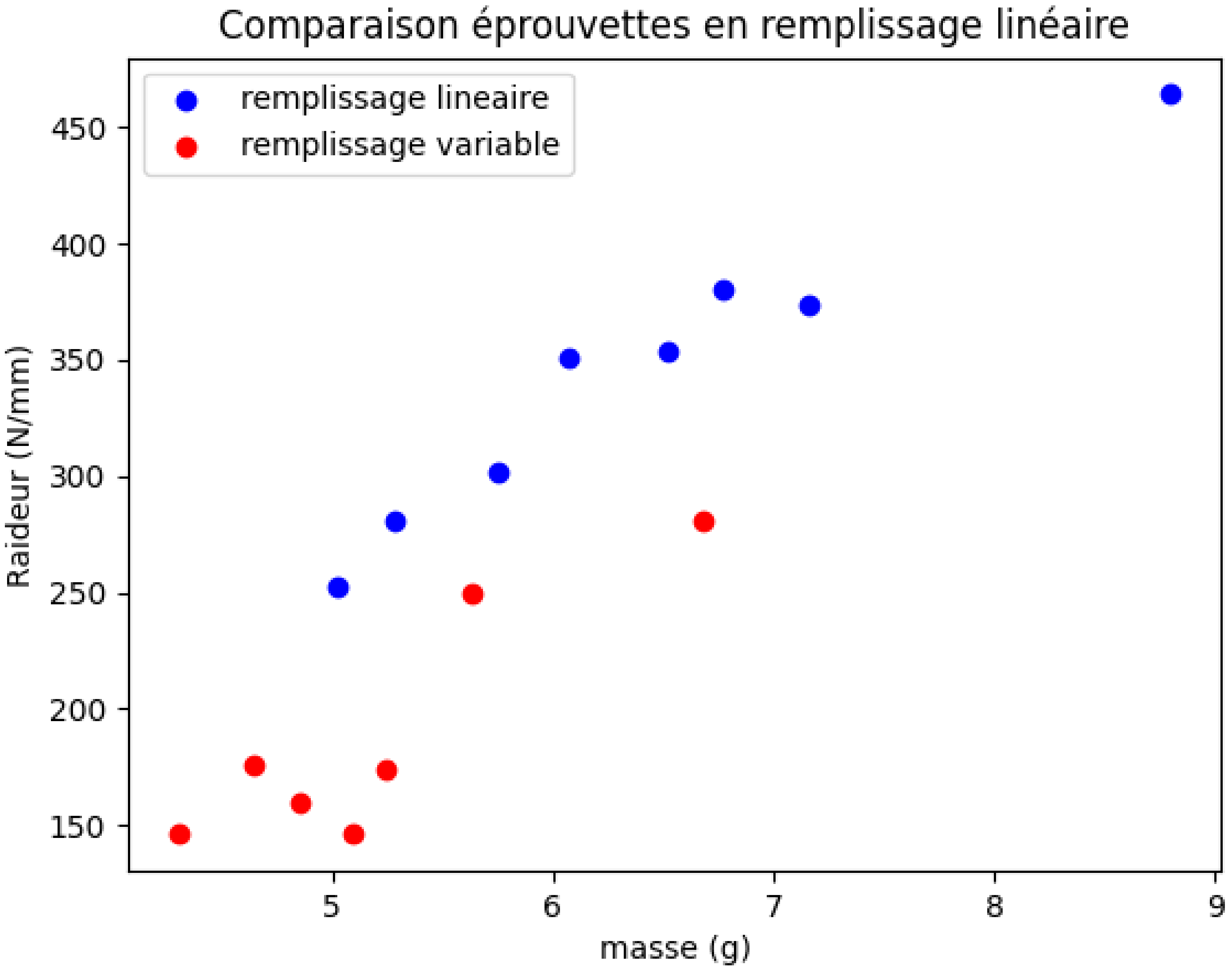


Le remplissage variable est, comparé au remplissage en grille, efficace pour améliorer la rigidité des éprouvettes.

On retrouve cette même efficacité en comparaison avec un remplissage linéaire (présenté dans I.)



III.2 Résultats



Les pièces imprimées avec la densité variable n'ont pas nécessairement un module de Young plus élevé (en fonction de la masse).

III.2 Bilan sur la comparaison des pièces

Le remplissage à densité variable se distingue des remplissages à densité constante.

-Il permet d'augmenter la limite de résistance à la traction.
-A masse égale, il peut aussi bien réduire qu'augmenter le module de Young (dépendance à la forme de la pièce).
-Cependant, il est nécessaire d'avoir une certaine densité pour pouvoir observer ces phénomènes.

Conclusion

<u>Objectifs du TIPE :</u>

- Vérifier les lois théoriques sur le PLA ☑
- Générer un remplissage à densité variable pour optimiser la pièce. ☑
- Comparer le remplissage ainsi généré avec des remplissages classiques. ☑

Conclusion globale :

Modifier le remplissage au niveau local a un effet significatif sur les propriétés mécaniques d'une pièce imprimée en 3D.

Merci pour votre attention.

Annexe : Affichage des graphiques avec python

Librairies :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from scipy import stats


def calculate_stiffness(file):
    lire=dict(pd.read_csv(file,delimiter=";"))
    force = list(lire["charge (N)"])
    deplacement = list(lire["deplacement (mm)"])
    force_reg=[]
    deplacement_reg=[]
    droit = True
    for i in range(len(force)):
            if ((deplacement[i] > 2.3 and deplacement[i] < 4.1) or force[i]>200 ) and droit:
                deplacement_reg.append(deplacement[i])
                force_reg.append(force[i])
            if deplacement[i]>4.1 or i==len(deplacement)-1 or force[i]>force[i+1]+5: droit = False

    X = np.array(deplacement_reg)
    Y = np.array(force_reg)
    slope, intercept, r_value, p_value, std_err = stats.linregress(X, Y)
    return slope
```

```python
def comparatif(folder) :
    path=folder
    files=os.listdir(path)
    os.chdir(folder)
    dico=dict(pd.read_csv("masse.txt",delimiter=";"))
    fig , ax = plt.subplots()
    xmin=-0.1
    xmax=0
    anotations=[]
    for file in files:
        if file!="masse.txt":
            couleur='blue'
            if file[:8]=='variable' : couleur = 'red'
            raid = round(calculate_stiffness(file),2)
            print('Raideur de '+file[:-4]+' : ', raid, 'N/mm')
            lire=dict(pd.read_csv(file,delimiter=";"))
            force = list(lire["charge (N)"])
            deplacement = list(lire["deplacement (mm)"])
            ax.plot(np.array(deplacement),np.array(force), label = file[:-4]+" : "+str(list(dico[file[:-4]])[0])+" g" , color = couleur)
            an=ax.annotate(
                    file[:-4]+'\n'+'Rupture : '+str(deplacement[-1])+' mm'+'\n'+str(force[-1])+" N"+"\n"+'Raideur : '+str(raid)+' N/mm',
                    xy=(deplacement[-1],force[-1]), xycoords='data',
                    xytext=(30, -40), textcoords='offset points',
                    bbox=dict(boxstyle="round", fc="0.8",color = couleur),
                    arrowprops=dict(arrowstyle="->",
                    connectionstyle="angle,angleA=0,angleB=90,rad=10"))
            anotations.append(an)
            an.draggable()
            if deplacement[-1]>xmax : xmax = deplacement[-1]
    ax.set_xlim(xmin,xmax+1)
    ax.set_xlabel("deplacement (mm)")
    ax.set_ylabel('force (N)')
    ax.legend()
    ax.set_title('Comparaison des éprouvettes '+folder)
    plt.show()
comparatif('losange_3')
```

Annexe : Calcul du polynôme à partir de la carte de contrainte

Librairies :

```
import pyvista as pv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import json
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import sympy as sp
```

```
def contraintes_to_densite(l_contrainte):
    l_contrainte = np.array(l_contrainte)
    return (l_contrainte-min(l_contrainte))/(max(l_contrainte)-min(l_contrainte))
```

```
def compute_densite(file):
    mesh = pv.read(file)
    lcontrainte= mesh.point_data['Contrainte:von Mises']
    l_densite = contraintes_to_densite(lcontrainte)
    l_points = mesh.points.tolist()
    l_points = [tuple(point) for point in l_points] # pour pouvoir les passer en clé de dictionnaire

    densite_dico = dict(zip(l_points, l_densite))
    #Ldens = set_density(dmin,dmax,Lstress,fct)
    return densite_dico.copy()
```

```
def compute_polynome(l_points,
l_densite, degree=2):
    """
    retourne les coefficient du
polynome a 3 inconnues obtenue par
methode des moindre carré :
    P(X, Y, Z) = densité
    """
    # Générer les termes polynomiaux
    poly =
PolynomialFeatures(degree=degree)
    P_poly =
poly.fit_transform(l_points)

    # Ajuster le modèle linéaire (sur
les termes polynomiaux)
    model = LinearRegression()
    model.fit(P_poly, l_densite)

    return model, poly
```

```
def polynomial_to_tensor(model, poly, var_names=['x', 'y', 'z']):
    """
    Convertit un polynôme sklearn en un tableau 3D où tab[i][j][k] correspond au coeff de
x^i * y^j * z^k.
    :param model: modèle sklearn entraîné (LinearRegression)
    :param poly: objet PolynomialFeatures utilisé pour générer les termes
    :param var_names: noms des variables, par défaut ['x', 'y', 'z']
    :return: tableau numpy 3D
    """
    from collections import defaultdict

    coef_dict = defaultdict(float)
    feature_names = poly.get_feature_names_out(var_names)

    max_deg = poly.degree
    # Remplir le dictionnaire des coefficients
    for coef, name in zip(model.coef_, feature_names):
        if name == "1":
            i = j = k = 0
        else:
            parts = name.replace("^", "**").split(" ")
            i = j = k = 0
            for part in parts:
                if part.startswith("x"):
                    i += int(part.split("**")[1]) if "**" in part else 1
                elif part.startswith("y"):
                    j += int(part.split("**")[1]) if "**" in part else 1
                elif part.startswith("z"):
                    k += int(part.split("**")[1]) if "**" in part else 1
        coef_dict[(i, j, k)] += coef
    # Créer le tableau 3D
    tab = np.zeros((max_deg + 1, max_deg + 1, max_deg + 1))
    for (i, j, k), val in coef_dict.items():
        tab[i][j][k] = val
    # Ajouter l’intercept à (0,0,0)
    tab[0][0][0] += model.intercept_

    return tab
```

```python
def save_polynme_as_json(tensor, l_points, filename):
    """
    tensor : polynome sous forme de tensseur
    -> exporte le polynome pour que ce dernier soit accessible par le programme C++
    """
    x_coords = [p[0] for p in l_points]
    y_coords = [p[1] for p in l_points]
    min_x, max_x = min(x_coords), max(x_coords)
    min_y, max_y = min(y_coords), max(y_coords)



    b_box = [[min_x, min_y], [max_x, min_y], [max_x, max_y], [min_x, max_y]]
    print(b_box)
    data = {
        "tensor": tensor.tolist(),
        "bbox": b_box
    }

    # Écriture dans le fichier JSON
    with open(filename, 'w') as f:
        json.dump(data, f, indent=2)
```

Annexe : Démonstration contrainte de Von Mises

Nous avons $dev(\underline{\sigma}) = \begin{pmatrix} \frac{1}{2}(\sigma_{11} - \sigma_{22}) & \sigma_{12} \\ \sigma_{12} & \frac{1}{2}(\sigma_{22} - \sigma_{11}) \end{pmatrix}$

$C = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & 1-2\nu \end{pmatrix}$ d'où

$C^{-1} = \frac{1+\nu}{E} \begin{pmatrix} 1-\nu & -\nu & 0 \\ -\nu & 1-\nu & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Ainsi $U_f = \frac{1}{2} tr(dev(\underline{\sigma}) \cdot \underline{\epsilon}^T)$

$$U_f = \frac{1+\nu}{2E}[\frac{1}{2}(\sigma_{11} - \sigma_{22})((1-\nu)\sigma_{11} - \nu\sigma_{22})$$
$$+ \frac{1}{2}(\sigma_{22} - \sigma 11)((1-\nu)\sigma_{22} - \nu\sigma_{11})$$
$$+ 2\sigma_{12}^2]$$
$$U_f = \frac{1+\nu}{2E}((\sigma_{11} - \sigma_{22})^2 + 2\sigma_{12}^2)$$

Ce qui donne le résultat attendu.