Séance 04 :

Essai de traction

1. Problématique et mise en situation

C.F. sujet.

2. Caractérisation des matériaux

1. ***Ecrire une fonction*** `transf(d, f)` ***qui prend en argument un allongement et une force, et qui retourne deux valeurs :***
   - `eps` ***, la valeur de l'allongement relatif***
   - `sigma` ***la valeur de la contrainte***

   Pour une meilleure organisation, je crée un dossier `ressources` dans lequel je placerais toutes les ressources. Pour l'instant je crée le fichier `ressources/donnees.py` :

```python
# Donnees de la seance 04

# longueur a vide de l'eprouvette
l0 = 50

# section de l'eprouvette
S = 2*12.5
```

   J'importe les librairies qui me seront utiles plus tard :

```python
# Importation of external modules
import numpy as np
import os.path
import matplotlib.pylab as plt
```

   Je peux maintenant répondre à la question :

```python
# Importation of constant data
from ressources.donnees import *

# Transform function that return epsilon and sigma


def Transform(m, f):
    return (m/l0, f/S)
```

2. ***Ecrire une fonction*** `lecture(essai)` ***qui prend en argument le nom de l'essai, et qui retourne :***
   - ***Le nom du matériau de l'essai***
   - ***Un tableau à 2 colonnes composées uniquement de données numériques :***
     - ***La*** $1^{ère}$ ***indiquant les allongements relatifs de l'essai***
     - ***La*** $2^{ème}$ ***indiquant les contraintes correspondantes à celles des allongements relatifs.***

   Je place les 4 résultats des essais `essai_1.csv`, `essai_2.csv`, `essai_3.csv` et `essai_4.csv` dans le dossier `ressources`. Je préfère décomposer la question en plusieurs fonctions :

```python
def isNumeric(inputList):
    """Return true if all elements in a list are numeric (int or float)"""
    for element in inputList:
        try:
            float(element)
        except:
            return False
    return True
```

   Le classique `isNumeric` qui n'existe pas simplement en Python !

```python
def clearEmptyCells(inputList):
    """
        @inputs  : list
        @returns : same list without empty cells
    """
    return [k for k in inputList if k != '']
```

`clearEmptyCells` pour supprimer d'éventuelles cellules vides.
Et enfin :

```python
def ConvertDataFromCSV(path):
    """Read data from the csv given and return an numpy array filled with
    this data"""
    materialName = ''  # Name of the tested material
    epsilon = []  # Movemennt of the test piece
    sigma = []  # Force applied on the test piece
    starting = True

    dirname = os.path.dirname(__file__)
    filename = os.path.join(dirname, path)

    if (os.path.exists(filename)):
        try:
            file = open(filename, 'r')
            for row in file.readlines():
                try:
                    line = clearEmptyCells(row.replace('\n', '').split(';'))
                    if (starting):
                        starting = not starting
                        materialName = line[0]
                    elif (isNumeric(line)):
                        # Data reading from the CSV file
                        movement = float(line[0])
                        force = float(line[1])

                        # Formating of data
                        tranformedData = Transform(movement, force)

                        # Adding data to result that will be returned
                        epsilon.append(tranformedData[0])
                        sigma.append(tranformedData[1])
                except:
                    print('error !')
        finally:
            file.close()
        return (materialName, sigma, epsilon)
    else:
        raise ValueError("File ({}) doesn't exists".format(path))
```

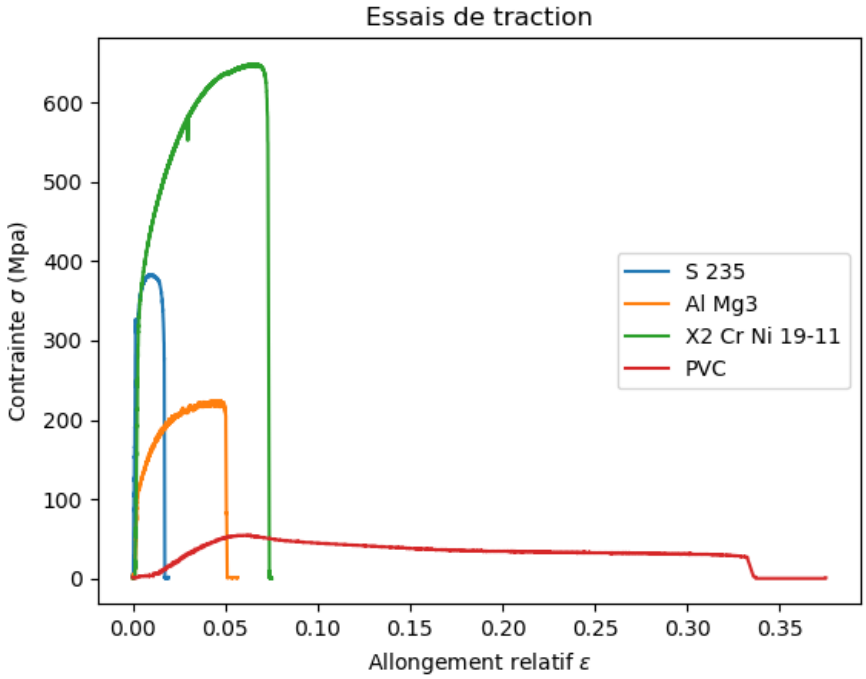
3. ***Ecrire une procédure*** `trace(nom, T)` ***qui prend en argument le nom du matériau, et un tableau à 2 colonnes de données numériques et qui trace la colonne 1 en fonction de la colonne 1 avec pour légende le nom du matériau et :***
   - ***Pour titre : "Essais de traction"***
   - ***Pour label d'abscisse : "Allongement relatif"***
   - ***Pour label d'ordonnée : "Contrainte (MPa)"***
   - ***La légende affichée au milieu à gauche de la figure***

```python
A = ConvertDataFromCSV('ressources/essai_1.csv')
plt.xlabel("Allongement relatif $\epsilon$")
plt.ylabel("Contrainte $\sigma$ (Mpa)")
plt.title("Essais de traction")
plt.plot(A[2], A[1], label=A[0])
plt.legend(loc='center left')
plt.show()
```

4. ***Ecrire une procédure*** `figure(n)` ***qui prend en argument le nombre d'essai*** `n` ***à tracer, et qui les trace sur la même figure.***

```
def figure(n):
    plt.xlabel("Allongement relatif $\epsilon$")
    plt.ylabel("Contrainte $\sigma$ (Mpa)")
    plt.title("Essais de traction")

    for n in range(1, n+1):
        A = ConvertDataFromCSV('ressources/essai_' + str(n) + '.csv')
        plt.plot(A[2], A[1], label=A[0])

    plt.legend(loc='center right')
    plt.show()
```

J'ai déplacé la légende à droite, car elle cachait les courbes. Voici le graphe obtenu :



5. ***Ecrire une fonction*** `rupture(T)` ***qui prend en argument un tableau à 2 colonnes*** (ϵ, σ) ***et qui retourne la valeur de la limite à la rupture*** R_m**.** Vu comme j'ai codé les fonctions précédentes, il faut que cette fonction prenne directement la liste des σ en paramètre :

```
def rupture(T):
    return max(T)
```

En exécutant :

```
A = ConvertDataFromCSV('ressources/essai_1.csv')
print(rupture(A[1]))
```

On trouve : `382.6092` Mpa

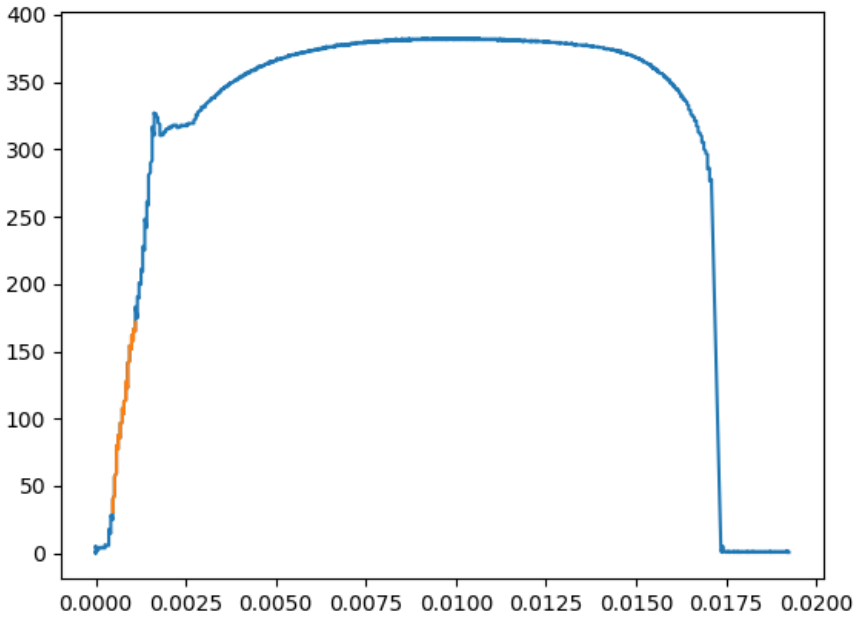
6. ***Ecrire une fonction*** `select(T)` ***qui prend en argument les données issues de l'essai et qui retourne la partie du tableau qui correspond à la plage définie plus haut.***
D'abord j'ai besoin d'une petite fonction de recherche :

```
def ind(L, e):
    i = 0
    while (L[i] < e and i < len(L)):
        i += 1
    return i
```

Puis je code la fonction demandée :

```python
def select(sigma, epsilon):
    # C'est pas optimise... mais c'est pour se servir de la fonction rupture

    # Limite a la rupture
    Rm = rupture(sigma)

    # Index a partir duquel on le depasse
    i = sigma.index(Rm)

    # On coupe les listes
    sigma = sigma[:i]
    epsilon = epsilon[:i]

    # Maintenant on selectionne les bonnes valeurs
    mini = ind(sigma, 0.08 * Rm)
    maxi = ind(sigma, 0.45 * Rm)

    # On coupe les listes
    sigma = sigma[mini:maxi]
    epsilon = epsilon[mini:maxi]

    return (sigma, epsilon)
```

En traçant pour le premier essai en orange la plage sélectionnée, et en bleu toute les valeurs on a :



7. *Ecrire une fonction* `somme(Ts)` *qui prend en argument le tableau de données sélectionnées et qui retourne un tableau* `Som` *dans lequel seront rangées les 5 valeurs* `Sxi`, `Syi`, `Sxiyi` *et* `Sxi2` *et* `n` *pour la plage de données concernée.*

```python
def somme(sigma, epsilon):
    sigma = np.array(sigma)
    epsilon = np.array(epsilon)
    n = sigma.shape[0]
    Sxi = np.sum(epsilon)
    Syi = np.sum(sigma)
    Sxiyi = np.sum(epsilon * sigma)
    Sxi2 = np.sum(epsilon**2)
    return (Sxi, Syi, Sxiyi, Sxi2, n)
```

On obtient :

`Sxi`	=	`0.07786759130836801`
`Syi`	=	`10925.483200000002`
`Sxiyi`	=	`9.156426879112345`
`Sxi2`	=	`6.375307483976406e-05`
`n`	=	`100`

8. ***Ecrire une fonction*** `Young(T)` ***qui prend en argument le tableau de données issues de l'essai de traction et qui retourne le module d'élasticité*** E ***et la valeur initiale*** $sigm_0$ ***pour la plage de donnée concernée.***

```
def young(sigma, epsilon):
    sigma, epsilon = select(sigma, epsilon)
    Sxi, Syi, Sxiyi, Sxi2, n = somme(sigma, epsilon)
    E = ((n * Sxiyi) - (Sxi * Syi)) / ((n * Sxi2) - (Sxi ** 2))
    sigm0 = (Syi - E * Sxi) / (n)

    return (E, sigm0)
```

On obtient :

`E`	=	`208054.24028098426`
`simg0`	=	`-52.75199352172678`

9. ***Ecrire une fonction*** `lim_el(T)` ***qui prend en argument les données issues de l'essai et qui retourne la limite élastique*** R_e***.***

Je définis d'abord une petite fonction test :

```
def test(A, B, E, sigm0):
    for i in range(10):
        valTh = E * A[i] + sigm0
        valRel = B[i]
        if(valRel < 1.05 * valTh):
            return False
        return True
```

Puis je code la fonction demandée :

```
def lim_el(sigma, epsilon, E, sigm0):
    iMax = len(epsilon) - 1
    i = 0
    while (not test(sigma, epsilon,  E, sigm0) and i <= iMax - 10):
        i += 1
    if (i != 0):
        return(sigma[i])
    else:
        return (0)
```

On obtient :

R_e	=	`161.8732`