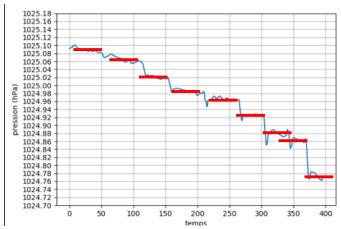
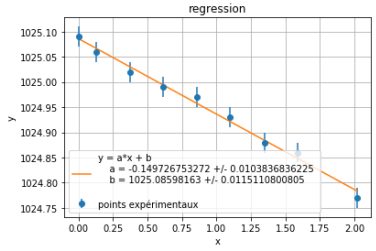
Grille de compétence	/1	/2	/3	/4
Analyser				
Mettre en équation la situation	Localement le modèle du GP et de l'atmosphère isotherme en équilibre est pertinent, on a : $P(z) = P_0 e^{-\frac{z}{\delta}}$ Avec $\delta = \frac{RT}{Mg}$ Avec $R \approx 8{,}31 J.K^{-1}.mol^{-1}, T \approx 294K$, $M = 29{,}0 g.mol^{-1}$ et $g \approx 9{,}81 m.s^{-2}$ Ici $\delta \gg z$ donc : $P(z) = P_0\left(1 - \frac{z}{\delta}\right)$			
Proposer le protocole pour obtenir la droite d'étalonnage	La lecture de la pression fait apparaître une variabilité que l'on peut moyenner en laissant le capteur dans une positions donné pendant 60s 			
Réaliser				
Savoir effectuer des mesures avec son smartphone	La régression linéaire donne alors la courbe $P(z)$  - Le comportement linéaire semble être vérifié			

	- On obtient une pente de $a = -15 Pa/m$ et une incertitude-type associée $u(a) = 1 Pa/m$. La valeur de référence est $a_{ref} = 12 Pa/m$ ce qui est en accord avec notre modèle - La pente est $b = (102509 \pm 1) Pa$ pour une valeur mesurée de $102509 Pa$
Valider	
Savoir utiliser la droite d'étalonnage	En utilisant le fichier utilisation_droite_etalon.py, je trouve une taille de $(157 \pm 8) cm$ au lieu de 181cm ce qui fait un écart de 3 incertitude-type.
Communiquer	
Proposer une analyse de ses résultats.	Il est impressionnant de pouvoir trouver des résultats qui corroborent la loi de la statique des fluides à notre échelle. Il est également surprenant de pouvoir connaître notre altitude à moins de 0,5m près. Si on cherche à connaître notre taille au cm près, ce capteur n'est pas adapté. On peut retrouver rapidement ce résultat en notant que : $\left\|\frac{dP}{dz}\right\| = \frac{P_0 e^{\frac{z}{\delta}}}{\delta} \approx \frac{P_0}{\delta}$ soit $\Delta z \approx 10 cm$

```
t,P,T=tableau_V()
plt.plot(t,P)
plt.grid()
plt.xlabel("t")
plt.ylabel("P(hPa)")
plt.show()
tab_z=[0,0.13,0.38,0.62,0.865,1.11,1.35,1.6,2.2]
tab_P=[1012.65,1012.63,1012.61,1012.56,1012.53,1012.5,1012.47
,1012.43,1012.33]
```

```
u_P=np.ones(len(tab_z))*0.05
u_z=np.ones(len(tab_z))*0.01
import numpy as np
import matplotlib.pyplot as plt

def tableau_V ():
    """le fichier csv contient des valeurs décimales écrite
avec des ,
    et séparées par par des ;
    dans l'exemple propsé, on récupère les données de la
première
    et de la deuxième colonne (la 1e ligne étant exclue)"""
    f=open("mesures.csv","r")
    tab=f.readlines()#tab est une liste des valeurs de chaque
ligne du tableau excel
    f.close()
    N=len(tab)-3
    table_P=np.zeros((N))#tableau vide
    table_T=np.zeros(N)
    table_t=np.zeros((N))

    for i in range(N):
        ligne=tab[i]
        ligne=ligne.split(",")
        P,T,t=ligne[0],ligne[1],ligne[2]
        P,T,t=float(P),float(T),float(t)
        table_P[i],table_T[i],table_t[i]=P,T,t#on remplit les
tableaux
    return table_t,table_P,table_T

def RegLin(X,u_X,Y,u_Y):
    """cette fonction prend pour argument 4 tableaux numpy,
    - le tableau X contient les valeurs en abscisse
    - le tableau des incertitudes-types de X (qui peuvent
être nulles !)
    - le tableau Y contient les valeurs en ordonnée
    - le tableau des incertitudes-types de Y
     cette fonction retourne Y(X) ainsi que l'équation de la
droite et les incertitudes types associées"""
    M=10000#nbr itérations pour Monte Carlos
    tab_a=np.zeros((M))
    tab_b=np.zeros((M))
    for i in range(0,M):
        tab_X=np.random.normal(X,u_X)#On génère des valeurs
aléatoires pour toutes les valeurs Xi
        tab_Y=np.random.normal(Y,u_Y)#On génère des valeurs
aléatoires pour toutes les valeurs Yi
        p=np.polyfit(tab_X,tab_Y,1)#Régression linéaire : on
trouve le polynôme d'ordre 1 (moindres carrés)
        tab_a[i]=p[0]#pente
        tab_b[i]=p[1]#Y(0)
    a = np.mean(tab_a)#moyenne des pentes
    b = np.mean(tab_b)#moyenne  des ordonnées à l'origine
    u_a=np.std(tab_a)#écart type ou incertitude-type
    u_b=np.std(tab_b)# écart type ou incertitude-type
    x=np.linspace(np.min(X),np.max(X),1000)
    y=a*x+b#Droite de régression

plt.errorbar(X,Y,xerr=u_X,yerr=u_Y,linestyle="",marker='o',la
bel="points expérimentaux")#points exp + barres d'erreur
    plt.plot(x,y,label="Y=aX+B \n\
    a = {0:.3f} +/- {1:.3f} \n\
    b = {2:.3f} +/- {3:.3f}".format(a,u_a,b,u_b))
    plt.grid(True)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("regression")
    plt.legend()
    plt.show()
RegLin(tab_z,u_z,tab_P,u_P)
def etalon(a,u_a,b,u_b,Yreport):
    """cette fonction prend pour argument les valeurs de a et
b d'une fonction linéaire y(x)=ax+b
    u_a et u_b sont les incertitudes-types associées à a et b
    Yreport est l'ordonnée du point P dont on veut l'abscisse
Xp (avec incertitude obtenue par MC)
    cette fonction retourne [Yreport, Xp, incertitude-type de
Xp]"""
    M=10**4
    Tab_Xpred=np.zeros((M))
    for i in range (M):
        b_new=np.random.normal(b,u_b)
        a_new=np.random.normal(a,u_a)
        Xpred=(Yreport-b_new)/a_new
        Tab_Xpred[i]=Xpred
    Xpred=np.mean(Tab_Xpred)
```

```
    u_Xpred=np.std(Tab_Xpred)
    return [Yreport,Xpred,u_Xpred]
print(etalon(-0.143,0.025,1012.654,0.028,1012.38))
```