Évolution physique d'un ballon d'air chaud dans la troposphère

SALOMON Marien

L'expérience est réalisée sur une montgolfière de l'association beaune-montgolfière.

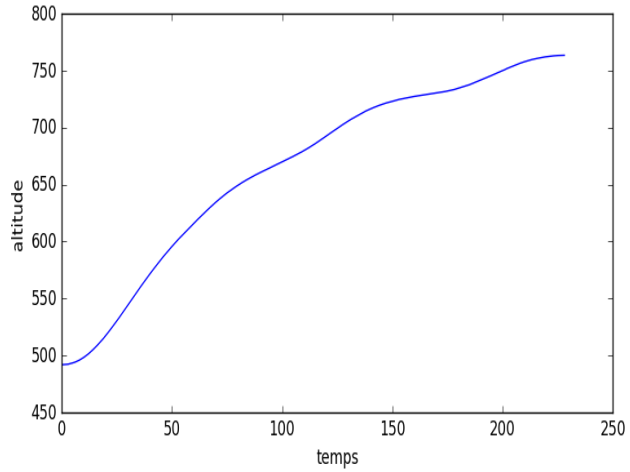
Grandeurs physiques mesurées :

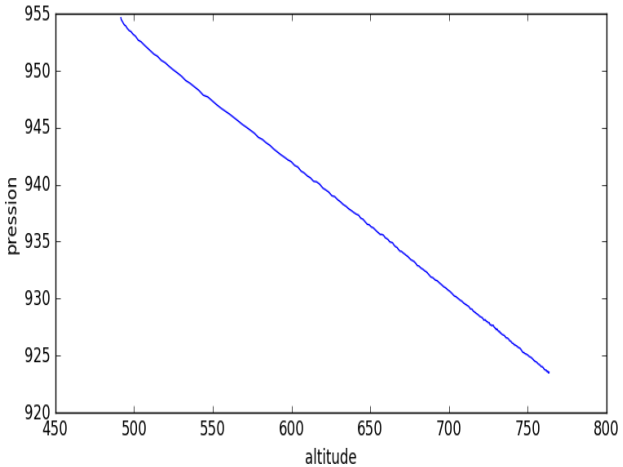
- Altitude (GPS)
- Pression (capteur Bosch BMP085 digital pressure sensor)
- Température

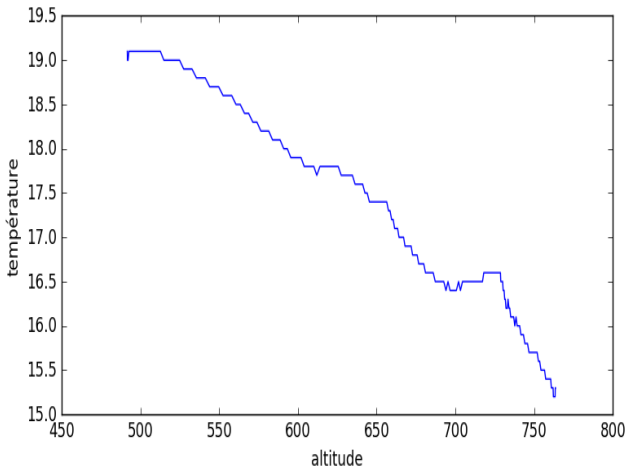
Le pas d'acquisition est fixé sur la fréquence de réception du signal GPS : $1s$.
On obtient un fichier au format .csv sous la forme suivante :

UTC time	altitude (m)	VDOP	nb of sat	pressure (hPa)	temperature (C)
15:06:06	405,1	1,7	9	964,03	20,4
15:06:07	405,1	1,7	9	964,02	20,4

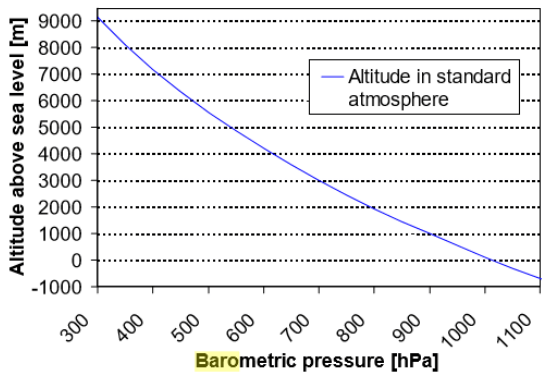
On traite le fichier obtenu grâce au langage de programmation python. Et on sélectionne une portion intéressante du vol. On obtient alors les courbes suivantes :


altitude
temps
800
750
700
650
600
550
500
450
0
50
100
150
200
250

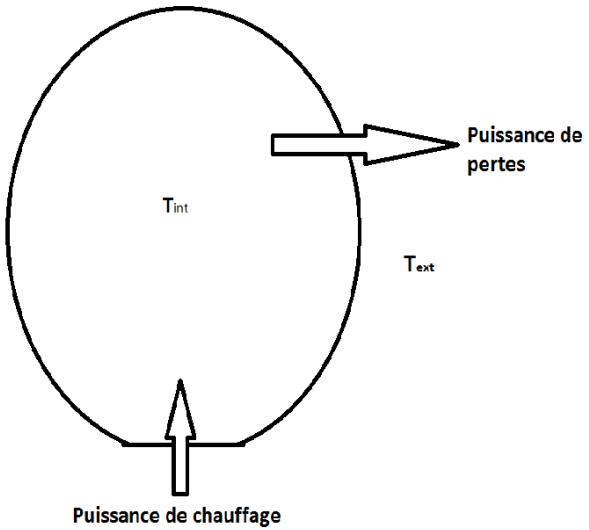

pression
altitude
955
950
945
940
935
930
925
920
450
500
550
600
650
700
750
800


température
altitude
19.5
19.0
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
450
500
550
600
650
700
750
800

Dans l'atmosphère adiabatique, on considère que la pression et la température sont données par :
$P(z) = P_{mer}(1 - \frac{0.0065z}{T_{mer}})^{5.225}$ (formule du nivellement barométrique)

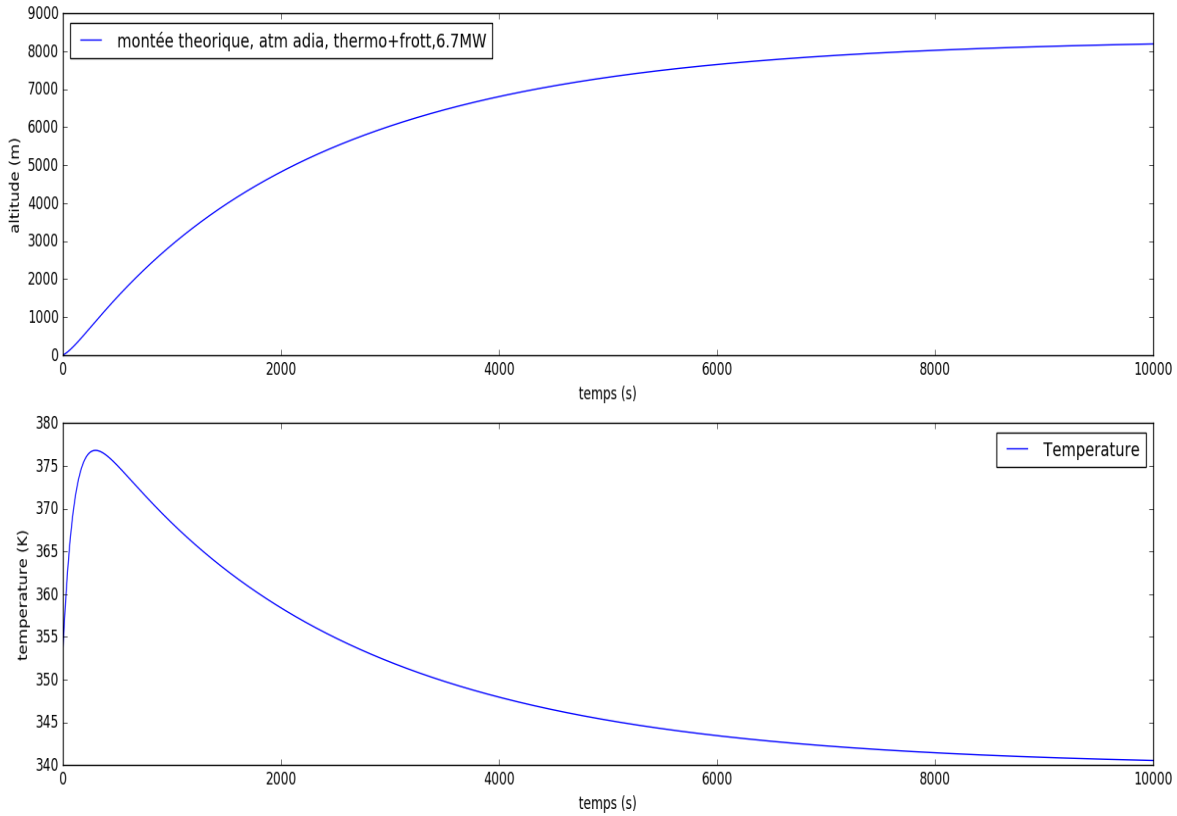


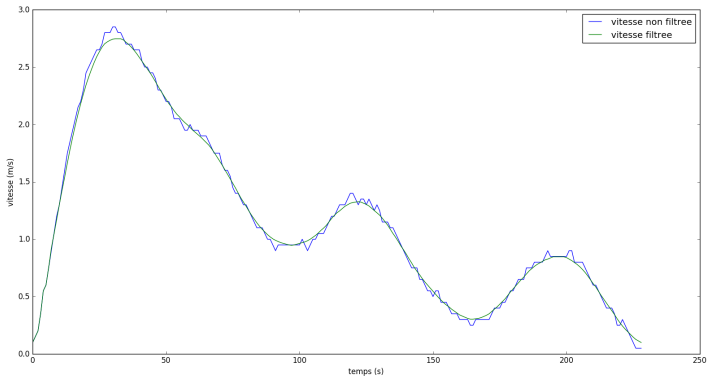
$T(z) = T_{mer} - 0.0065z$ Le modèle s'applique entre 0 et 16000m d'altitude.

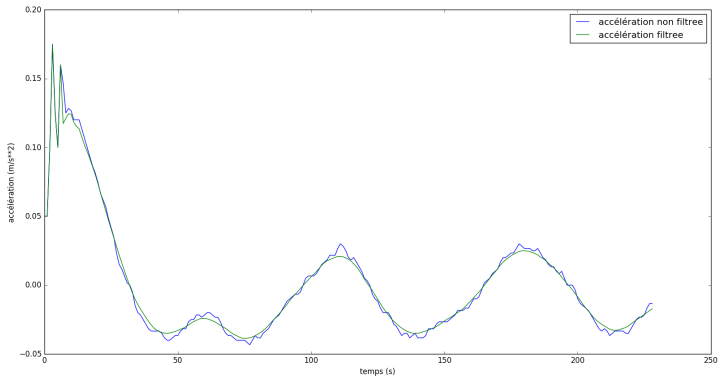


Puissance de pertes :

$$P_{pertes} = \sigma S T^4 + \lambda S \frac{T_{int} - T_{ext}}{e} + hS(T_{int} - T_{ext})$$

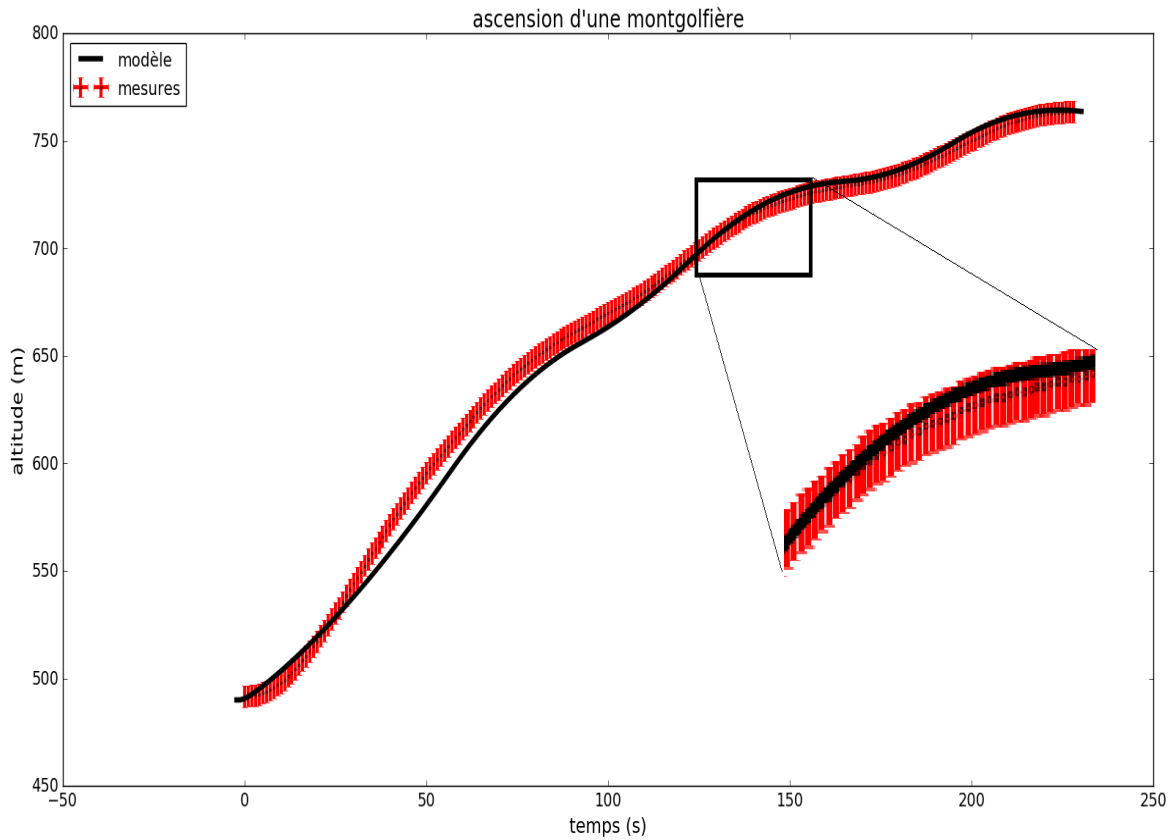

montée theorique, atm adia, thermo+frott,6.7MW
altitude (m)
temps (s)
Temperature
temperature (K)
temps (s)



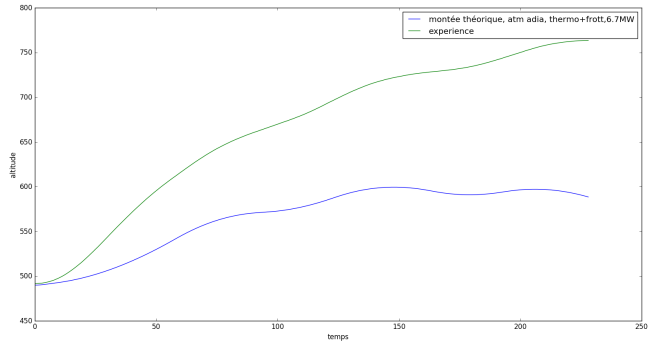


Le filtrage est réalisé par moyenne glissante.

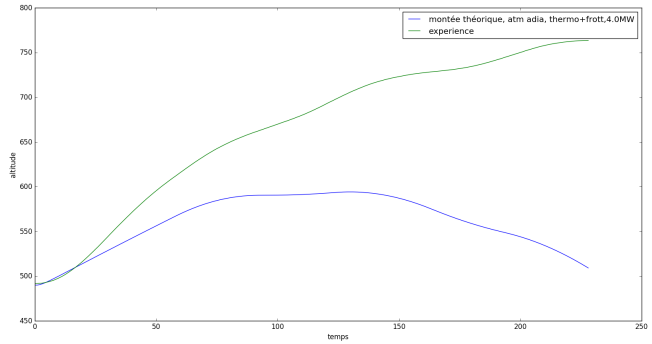
En utilisant ce modèle et en faisant correspondre les instants de chauffage avec l'expérience, on obtient :



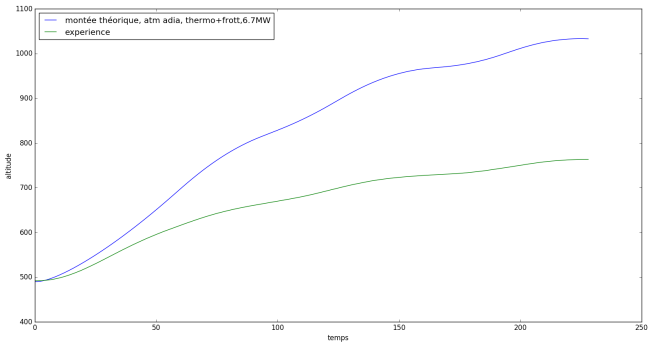
Paramètre	valeur utilisée	ordre de grandeur
frottement α	2000	?
Puissance brûleur	$6.7MW$	$6.3MW$ http ://www.cameronfrance.com, brûleur shadow
Température	$T_{mer} = 293K$	$288K$



Température



Puissance



α

Conclusion

La construction théorique et informatique réalisée par confrontation à l'expérience fournit un modèle prédictif.
Ainsi, une fois les paramètres réglés, le pilote peut disposer d'une prédiction en altitude pour les minutes à venir. Cela permet un confort de pilotage accru et une amélioration de la sécurité des vols.

Annexes I

```
def Ppertes(z,Tint, T_ext):
    """z l'altitude, Tint la temperature interieure,
    T_ext est une fonction de z renvoyant la temperature exterieure,
    Ppertes renvoie la puissance thermique des pertes"""
    sigma = 5.67 * 10**-8 #W m-2 K-4
    h = 10. #W m-2 K-1 il s'agit d'un ordre de grandeur
    lambd = 0.048 #W K-1 m-1
    e = 2*10**-3 #m (epaisseur de l'enveloppe)
    return sigma * S * Tint**4 + lambd * S * (Tint - T_ext(z)) / e +
           h * S * (Tint - T_ext(z))


#TIPE#

#conclusion de l'etude precedente et application Ã  l'experience

#hypotheses:
#On prend en compte :
#l'atmosphere adiabatique
#la thermodynamique du ballon et les transferts thermiques associees
#les frottements de l'air sur le ballon : frottements fluides :
#Ffrott = - alpha*v


#modules :
import numpy as np
import matplotlib.pyplot as plt
```

Annexes II

```
#variables globales :
g = 9.81 #m.s^-2
m_charge = 900. #kg
R = 8.314
Z_th0 = np.array([0, 0, 273+80]) #altitude, vitesse, Temperature
Pmer = 1013.25*100 #Pa
Tmer = 273. + 20 #K
M = 28.97 * 10**(-3) #kg/mol #masse molaire air sec
gamma = 1.4
Pmer = 1013.*100 #hPa
T = 90 + 273 #K (temperature interieure)
V = 6000. #m^3 volume de l'enveloppe
S = 1600. #m^2 surface de l'enveloppe
Puiss_brul = 2500000. #W
C = 1256. * V #capacite thermique totale
alpha = 2000. #coefficient de frottement visqueux dans l'air
##########################################################################

def euler_thermo(F, Z_th0, Ttot, pas, T = T, puiss = Puiss_brul):
    """renvoie la liste des temps et celle des altitudes
    pour les conditions initiales donnees"""
    Lz = []
    Lt = []
    LT = []
    T = T
    z = Z_th0
    t = 0
```

Annexes III

```
    while t < Ttot :
        if z[0] < 0 :
            z[0] = 0
            z[1] = 0
        Lz.append(z[0])
        LT.append(z[2])
        Lt.append(t)
        z = z + pas * F(z, t, T, puiss)
        T = z[2]
        t += pas
    return Lt, Lz, LT

def T_ext(z):
    """approximation de l'atmosphere adiabatique """
    if z < 16000 :
        return Tmer - (6.5/1000) * z #le modele n'est valable
                                #que dans la troposphÃ¨re, z < 16000m
    else : return Tmer - (6.5/1000) * 16000

def P(z):
    """approximation de l'atmosphere adiabatique,
    formule de nivellement barometrique"""
    return Pmer * ( 1 - (0.0065 * z / Tmer))**5.255

def Ppertes(z,Tint, T_ext):
    """renvoie la puissance thermique des pertes"""
    sigma = 5.67 * 10**-8 #W m-2 K-4
```

Annexes IV

```
    h = 10. #W m-2 K-1 il s'agit d'un ordre de grandeur
    lambd = 0.048 #W K-1 m-1
    e = 2*10**-3 #m (epaisseur de l'enveloppe)
    return sigma * S * Tint**4 + lambd * S * (Tint - T_ext(z)) / e +
            h * S * (Tint - T_ext(z))

def m_air(z, T):
    """renvoie la masse de l'air contenu dans le ballon Ã  l'altitude z"""
    return P(z) * V * M / (R * T)

def F_thermo(Z_th,t, T, Puiss_brul = Puiss_brul):
    z, vz, T = Z_th
    return np.array([vz, g * ( ( (T/T_ext(z)) * (P(z)*V*M) / ( P(z)*V*M + m_charge*R*T )

def courbe_montee_thermo(Ttot, pas, Puiss = Puiss_brul):
    """affiche la courbe de montee obtenue theoriquement
    en prenant en compte l'aspect mÃ©canique et thermodynamique"""
    Lt, Lz, LT = euler_thermo(F_thermo,Z_th0, Ttot, pas, T, Puiss)
    #altitude:
    plt.subplot(211)
    plt.plot(np.array(Lt), np.array(Lz), label =
             'montÃ©e theorique, atm adia, thermo+frott, ' + str(Puiss/1000000) +
             'MW')
    plt.xlabel('temps (s)')
    plt.ylabel('altitude (m)')
    plt.legend(loc = 'best')
    #plt.ylim((-1000, 16000))
```

Annexes V

```
    #Temperature:
    plt.subplot(212)
    plt.plot(np.array(Lt), np.array(LT), label = 'Temperature')
    plt.xlabel('temps (s)')
    plt.ylabel('temperature (K)')
    plt.legend(loc = 'best')
    plt.show()

courbe_montee_thermo(10000, 1, 6700000)


def courbes_montee_puissances(Ttot, pas, Puiss_min, Puiss_max, n):
    """affiche n courbes de montee pour des puissances de bruleur
    comprises entre Puiss_min et Puiss_max
    Donner les puissances en W"""
    Puiss = np.linspace(Puiss_min, Puiss_max, n)
    Lt = np.arange(0, Ttot, pas)
    for p in range(len(Puiss)):
        plt.plot(Lt, np.array(
            euler_thermo(F_thermo, Z_th0, Ttot, pas, T, Puiss[p])[1]),
                label = 'Puissance=' + str(int(Puiss[p])) + 'W')
    plt.legend(loc = 'best')
    plt.ylim([0, 1000])
    plt.xlabel('temps (s)')
    plt.ylabel('altitude (m)')
    plt.show()

courbes_montee_puissances(800, 1, 2000000, 6700000, 8)
```

Annexes VI

```
#exp22.10:
T_chauff = [False for k in range(5)] +
[True for k in range(50)] +
[False for k in range(35)] +
[True for k in range(30)] +
[False for k in range(40)] +
[True for k in range(30)] +
[False for k in range(35)] +
[True for k in range(5)]

###
def m_air(z, T):
    """renvoie la masse de l'air contenu dans le ballon Ã  l'altitude z"""
    return P(z) * V * M / (R * T)

def F_brul(Z_th,t, T, Puiss_brul = Puiss_brul):
    z, vz, T = Z_th
    return np.array([vz, g * ( ( (T/T_ext(z)) *
                                 (P(z)*V*M) / ( P(z)*V*M + m_charge*R*T ) ) - 1)
                     - alpha * vz /(m_air(z, T) + m_charge), (Puiss_brul - Ppertes(z, T)

def F_sans_brul(Z_th,t, T, Puiss_brul = Puiss_brul):
    z, vz, T = Z_th
    return np.array([vz, g * ( ( (T/T_ext(z)) *
                                 (P(z)*V*M) / ( P(z)*V*M + m_charge*R*T ) ) - 1)
                     - alpha * vz /(m_air(z, T) + m_charge), - Ppertes(z, T) / C ])
```

Annexes VII

```
def Ppertes(z,Tint):
    """renvoie la puissance thermique des pertes"""
    sigma = 5.67 * 10**-8 #W m-2 K-4
    h = 5 #W m-2 K-1 il s'agit d'un ordre de grandeur
    lambd = 0.048 #W K-1 m-1
    e = 2*10**-3 #m (epaisseur de l'enveloppe)
    return  sigma * S * Tint**4 + lambd * S * (Tint - T_ext(z)) / e
+ h * S * (Tint - T_ext(z))


def T_ext(z):
    """approximation de l'atmosphere standard """
    if z < 16000 :
        return Tmer - (6.5/1000) * z #le modele n'est valable
                                     #que dans la troposphÃ¨re, z < 16000m
    else : return Tmer - (6.5/1000) * 16000

def P(z):
    """approximation de l'atmosphÃ¨re standard,
formule de nivellement barometrique"""
    return Pmer * ( 1 - (0.0065 * z / Tmer))**5.255

def euler_thermo(F_brul, F_sans_brul, Z_th0, Ttot, pas, T = T,
                 puiss = Puiss_brul):
    """renvoie la liste des temps et celle des altitudes
```

Annexes VIII

```
    pour les conditions initiales donnÃ©es"""
    Lz = []
    Lt = []
    LT = []
    T = T
    z = Z_th0
    t = 0
    while t < Ttot :
        if z[0] < 0 :
            z[0] = 0
            z[1] = 0
        if T_chauff[t]:
            Lz.append(z[0])
            LT.append(z[2])
            Lt.append(t)
            z = z + pas * F_brul(z, t, T, puiss)
            T = z[2]
        else :
            Lz.append(z[0])
            LT.append(z[2])
            Lt.append(t)
            z = z + pas * F_sans_brul(z, t, T, puiss)
            T = z[2]
        t += pas
    return Lt, Lz, LT

def courbe_montee_thermo(Ttot, pas, Puiss = Puiss_brul):
    """affiche la courbe de montee obtenue theoriquement
```

Annexes IX

```
en prenant en compte l'aspect mÃ©canique et thermodynamique"""
Lt, Lz, LT = euler_thermo(F_brul, F_sans_brul, Z_th0, Ttot, pas, T, Puiss)
#affichage courbe modele :
plt.plot(np.array(Lt), np.array(Lz), label =
         'montee_thÃ©orique, atm_adia, thermo+frott, '
        +str(Puiss/1000000)+'MW')
plt.xlabel('temps (s)')
plt.ylabel('altitude (m)')
plt.legend(loc = 'best')

#creation donnees exp
fichier = open("R22_1546_montee.csv", 'r')
contenu = csv.reader(fichier, delimiter = ',')
L_tps = []
L_alt = []
L_P = []
L_T = []
excep_lin0 = 0 #titres
excep_lin1 = 0 #enregistre le temps t0
for line in contenu:
    if excep_lin0 == 0 :
        excep_lin0 = 1
        excep_lin1 = 1
    elif excep_lin1 == 1 :
        t0 = 0
        L_tps.append(t0)
        L_alt.append(float(line[1][2:-1]))
```

Annexes X

```
            L_P.append(float(line[4][2:-1]))
            L_T.append(float(line[5][2:-1]))
            excep_lin1 = 0
            t0 +=1
        else :
            L_tps.append(t0)
            L_alt.append(float(line[1][2:-1]))
            L_P.append(float(line[4][2:-1]))
            L_T.append(float(line[5][2:-1]))
            t0 += 1
    #affichage courbe exp :
    plt.plot(L_tps, L_alt, label = 'experience')
    plt.xlabel('temps')
    plt.ylabel('altitude')
    plt.legend(loc = 'best')
    plt.show()
    return L_tps, L_alt, Lz


L_tps, L_exp, L_mod = courbe_montee_thermo(229, 1, 6700000)

## Affichage avec barres d'erreur :
plt.close('all') #


## Donnees experimentales
N = 229  # N points expÃ©rimentaux
```

Annexes XI

```
x_expe = L_tps # abscisses
y_expe = L_exp  # ordonnÃ©es

# Barres d'erreurs
err_x = [0.001 for _ in range (N)]
err_y = [5 for _ in range (N)]

Npts = 229
Delta_x = (np.max(x_expe) - np.min(x_expe) ) /100 #
x_model = np.linspace(np.min(x_expe)-Delta_x ,np.max(x_expe) + Delta_x ,Npts)


y_model = L_mod


## tracÃ© des courbes
# points expÃ©rimentaux
plt.errorbar(x_expe, y_expe, yerr=err_y, xerr=err_x ,fmt ='ro',
             linewidth=1,markersize=2,label= "mesures")
# modÃ¨le en trait plein
plt.plot(x_model ,y_model ,linewidth=4,color = 'k',label = "modÃ¨le")

# polices des axes plus grandes
plt.tick_params(labelsize=14)
```

Annexes XII

```
# nom des axes et titres
plt.xlabel('temps (s)',fontsize = 16)
plt.ylabel('altitude (m)',fontsize = 16)
plt.title("ascension d'une montgolfiere",fontsize = 18)
plt.legend(loc='best')
plt.show()
```