# 00_Rapport_Essai_V_Hautiere

March 24, 2021

## 0.1 1. essai01 modélisation avec `curve_fit`

L'objectif est de reprendre un TP sur l'étude du rayonnement thermique d'une ampoule à incandescence $(6V, 100mA)$ en remplaçant le traitement fait auparavant sous Excel par un traitement sous Python. On étudie le rayonnement d'une ampoule (6V, 100mA). Le filament est constitué par du tungstène dont la résistivité est une fonction de la température $T$ (en Kelvin) : $\rho = aT + bT^2$ où $a = 2,5 \cdot 10^{-14}\Omega.m.K^{-2}$ et $b = 2,3 \cdot 10^{-10}\Omega.m.K^{-1}$. Le filament est assimilé à un cylindre de section $s = \pi r^2$ et de longueur $l$. On rappelle la valeur de la constante de Stefan-Boltzmann $\sigma = 5.67.10^{-8}W.m^{-2}.K-4$.

On alimente l'ampoule en régime continu, on relève la tension U à ses bornes et le courant qui la traverse (convention récepteur). On peut établir (voir le pdf joint que j'avais fait à l'époque) la relation entre la resistance $R$ du filament et la puissance reçue selon :

$$R = c \times (P)^{1/2} + d \times (P)^{1/4}$$

L'objectif est d'obtenir les valeurs de $c$ et $d$ en utilisant l'outil `curve_fit` de python. Les données $(U, I)$ sont stockées dans le fichier essai01_UI.csv

[1]:
```
import csv
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy.optimize import curve_fit

def readCSV(file,sep,n):
    with open(file,"r") as f:
        read=csv.reader(f,delimiter=sep)
        col=[]
        for row in read :
            try:
                col.append(float(row[n].replace(",","."))
                #print (col)
            except:
                pass
    return col
#listes des valeurs experimentales
U=readCSV("essai01_UI.csv",";",1)
I=readCSV("essai01_UI.csv",";",2)
```

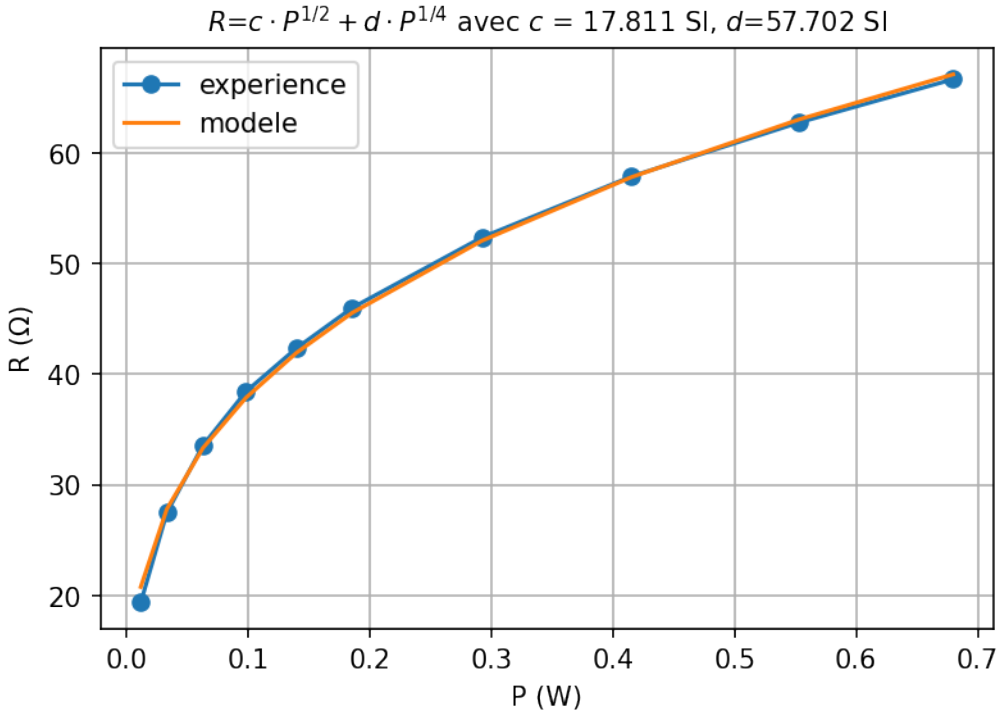
```
# generation des listes R=U/I et P=U*I
P=[float(x*y) for x,y in zip(U,I)]
R=[float(x/y) for x,y in zip(U,I)]

#definition du modele R=c*(P)^(0.5)+d*(P)^(0.25)
# Initialisation des paramètres du modele:
c=1
d=1

#definition du modele et optimisation
def func(P,c,d):
    return c*np.power(P,0.5)+d*np.power(P,0.25)
(c_opt,d_opt),_=curve_fit(func,P,R,p0=[c,d])

#traces de la courbe experimentale et du modele
print(c_opt,d_opt)
plt.figure(dpi=150)
plt.plot(P,R,'o-',label='experience')
plt.plot(P,func(P,c_opt,d_opt),label='modele')
plt.xlabel('P (W)')
plt.ylabel('R ($\Omega$)')
plt.title(r"$R$=$c\cdot P^{{1/2}}+d\cdot P^{{1/4}}$ avec $c$ = {:.3f} SI, $d$={:
 ↪.3f} SI""".format(c_opt, d_opt), fontsize=10)
#plt.title("""$R_e$ = {:.1f} $\Omega$, $R_m$ = {:.1f} $\Omega$, $f_0$ = {:.1f}␣
 ↪Hz, $Q$={:.1f}""".format(10,15,7,8), fontsize=10)
plt.legend()
plt.grid(True)
plt.show()
```

```
17.810650226734065 57.70173827442388
```




## 0.2 2. essai02 Filtrage numérique avec la méthode d'Euler explicite

L'objectif est de réaliser un filtrage numérique passe-bas (**circuit** $RC$) en mettant en entrée des signaux générés par le $GBF$ et enregistrés via **l'oscilloscope numérique**. A partir du GBF, j'ai généré successivement quatre signaux que j'ai enregistré ensuite via le port USB de l'oscilloscope : un signal sinusoïdal à $1kHz$ (essai02_sinus.csv), un signal triangle à $1kHz$ (essai02_triangle.csv), un signal carré à $1kHz$ (essai02_carre.csv) et un signal sinusoïdal bruité (essai02_signalbruite.csv) fabriqué à partir des deux sorties GBF du GBF Tektronix AFG1022 en mettant en sortie 1 un sinus à $1kHz$ d'amplitude $10V$ et en sortie 2 un sinus à $10kHz$ d' amplitude $0,1V$. Je les ai ensuite additionné avec la fonction math de l'oscilloscope.

Dans le programme ci-dessous, j'ai mis en entrée le signal carré et j'ai choisi une fréquence de coupure $f_0 = \dfrac{1}{2\pi\tau} = \dfrac{1}{2\pi \cdot 0,001} \approx 160Hz$. On observe bien le caractère intégrateur du filtres dans ces conditions avec les signaux d'entrée pré-cités.

[2]:
```
import csv
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy.optimize import curve_fit

#lecture du csv et recuperation des donnees experimentales
```

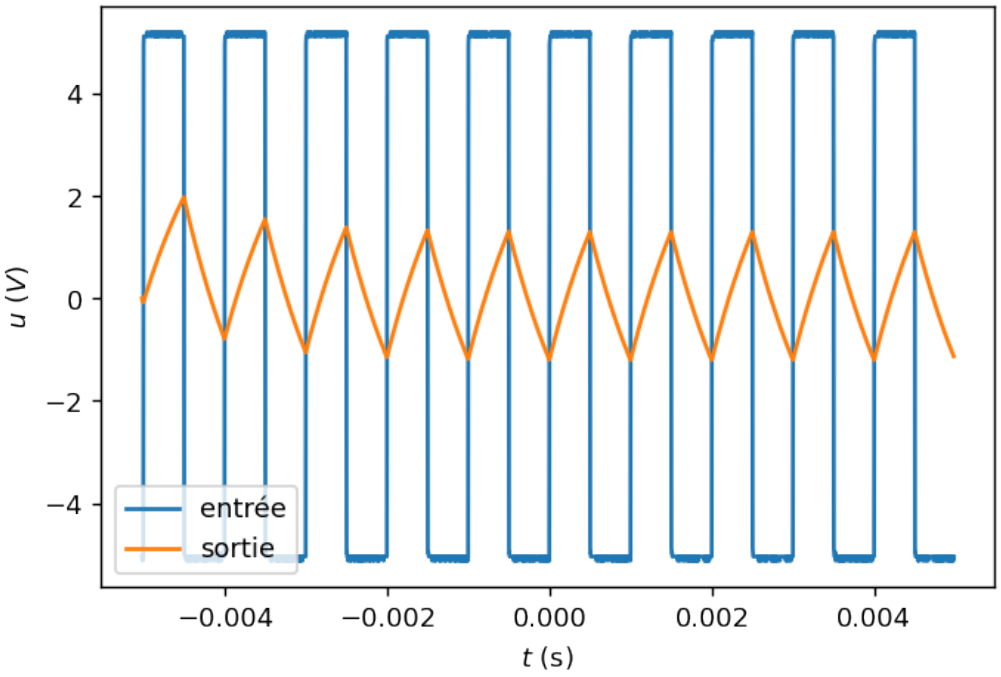
```
def readCSV(file,sep,n):
    with open(file,"r") as f:
        read=csv.reader(f,delimiter=sep)
        col=[]
        for row in read :
            try:
                col.append(float(row[n].replace(",","")))
            except:
                pass
    return col

#listes des valeurs experimentales
t_exp=readCSV("essai02_carre.CSV",",",3)
u_exp=readCSV("essai02_carre.CSV",",",4)

#methode d'Euler explicite, filtrage passe-bas
#initialisation des variables
N=len(t_exp) #nombre d'iterations
a=t_exp[0] #borne inferieure
b=t_exp[2499] #borne superieure
h=t_exp[1]-t_exp[0] #pas de discretisation
tn=t_exp[0] #valeur initiale de l'abscisse
un=0 #valeur initiale de l'ordonnee
t=[tn] #initialisation de la liste des abscisses
u=[un] #initialisation de la liste des ordonnees
tau=0.001 #valeur de tau

#algorithme d'Euler
for i in range(1,N):
    tn=tn+h
    un=un*(1-h/tau)+h/tau*u_exp[i]
#enregistrement des valeurs de tn et fn dans les listes t et f
    t=t+[tn]
    u=u+[un]

#traces de la courbe experimentale et du modele
plt.figure(dpi=130)
plt.plot(t_exp,u_exp,label="entrée")
plt.plot(t,u,label="sortie")
plt.xlabel('$t$ (s)')
plt.ylabel('$u$ ($V$)')
plt.legend()
plt.show()
```




## 0.3 3. essai03 Utilisation de la carte BBC Micro:Bit en accéléromètre avec récupération des données

Dans ce troisième essai, j'ai voulu tester la carte BBC micro:bit de manière autonome, ie, en téléversant un programme, en l'alimentant par pile et en la programmant pour qu'elle stocke les données dans sa mémoire flash afin de les exploiter par la suite. **Le code utilisé est en micropython (python pour microcontroleur) et donc n'est pas fonctionnel dans ce jupyter notebook et de plus nécessite de disposer d'une carte micro:bit connectée. Je l'ai donc mis entre commentaire. Je voulais juste montrer à ceux qui ne connaissent pas un exemple de syntaxe d'un code en micropython.**

Mon objectif est ici de récupérer les valeurs de l'accélération du capteur accéléromètre intégré à la carte sous forme d'un fichier ACC.csv. Ce capteur semble précis et travaille sur la gamme $[-2g, 2g]$. Je ne trouve pas pour l'instant les informations sur sa précision. Les valeurs récupérées sont en 'milli g'.

j'ai testé avec le train du TP DOPPLER en mouvement circulaire uniforme mais je n'obtiens pas pour l'instant de résultats probants (les valeurs d'accélérations ne mettent pas en évidence l'accérération centripète comme je m'y attendais).

[3]:
```
# import microbit as m
# while True:
#     if m.button_a.is_pressed():
#         m.display.clear()
#         m.display.scroll("GO")
#         with open('ACC.csv', 'w') as f:
```

```
#             t0 = m.running_time()
#             for i in range(100):
#                 t = m.running_time()-t0
#                 x = m.accelerometer.get_x()
#                 y = m.accelerometer.get_y()
#                 z = m.accelerometer.get_z()
#                 f.write(str(t)+";"+str(x)+";"+str(y)+";"+str(z)+"\n")
#                 m.sleep(50)
#         m.display.scroll('FIN')
#     else:
#         m.display.scroll("A")
```

Le fichier ACC.csv est enregistré sur la carte et on le récupère en utilisant un éditeur spécialisé. J'ai utilisé **l'éditeur Mu** qui est dédié aux cartes Micro:bit et qui possède un onglet fichier permettant de lire les données enregistrées sur la carte et de les récupérer sur le pc. . (Le csv n'est pas directement accessible sur la memoire flash de la carte car il doit etre décodé avant d'etre recuperable).