Moteur de Stirling avec équipement Arduino

Éric Brillaux et Benjamin Guiselin

29 mars 2022

1 Bibliographie

- *Débuter avec Arduino*, Delphine Chareyron & Antoine Bérut.
  `https://culturesciencesphysique.ens-lyon.fr/ressource/TP-Arduino-debut.xml`
- *Détection de 3 niveaux de luminosité et pilotage de sorties d'un micro-contrôleur Arduino*, Delphine Chareyron & Antoine Bérut.
  `https://culturesciencesphysique.ens-lyon.fr/ressource/TP-Arduino-controle-sorties.xml`
- *Mesure de la température à l'aide d'un montage Arduino et représentation graphique des données avec python*, Antoine Bérut.
  `https://culturesciencesphysique.ens-lyon.fr/ressource/TP-Arduino-temperature-berut.xml`

2 Mise en marche

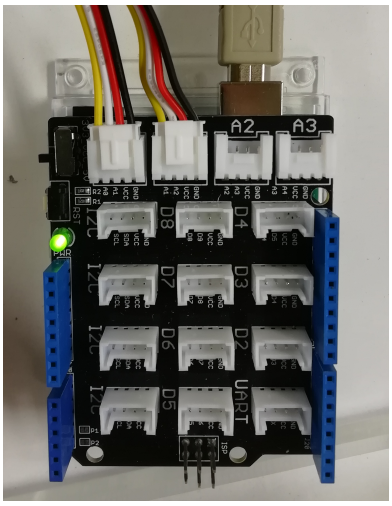
- On alimente la carte Arduino avec le port USB relié à l'ordinateur (Fig. 1a).
- On branche le capteur de pression et les deux fourches optiques sur trois entrées analogiques de la carte Arduino. La carte Arduino utilisée (modèle UNO) comprend quatre entrées analogiques (de A0 à A3, Fig. 1a).
- Le capteur de pression est le capteur différentiel linéaire MPX2100DP qui mesure la différence entre la pression du gaz dans la chambre et la pression atmosphérique dans une de gamme de pression 100 kPa. Il renvoie une tension égale à la moitié de la tension crête-à-crête ($V_{cc} = 40\,\text{mV}$) à différence de pression nulle. La sortie du capteur de pression est une valeur de tension U_p codée sur 10 bits [1] ($2^{10} = 1024$), à partir de laquelle on détermine la surpression p par rapport à la pression atmosphérique selon la loi affine

$$p(\text{kPa}) = 100 \times \frac{U_p(\text{bits})}{1023} - 50. \tag{1}$$

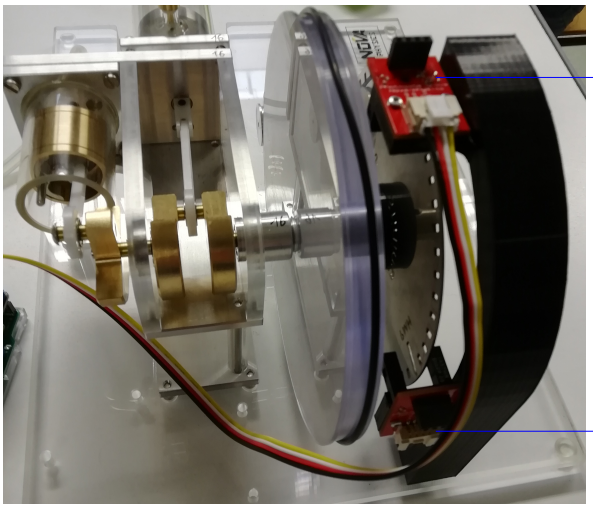
- La fourche optique du haut détecte le point mort haut (fente longue) ; elle permet de compter le nombre de tours du volant d'inertie attaché à l'arbre de rotation du moteur (Fig. 1b), et donc le nombre n de cycles effectués par le moteur. Au passage de la fente longue, la fourche optique renvoie une tension égale à 5 V (alors qu'elle est nulle sinon). Un code Arduino permet alors de compter le nombre n de fronts montants de la tension qui s'assimile au nombre de tours.
- La fourche optique du bas détecte le passage des fentes (courtes et longue) ; elle permet de déterminer la position angulaire du volant d'inertie, qui contient $N = 50$ fentes. Cette donnée permet de remonter au volume du gaz dans la chambre du moteur. Dans la notice, on lit que le volume minimal de la chambre vaut $V_{min} = 32\,\text{cm}^3$ et le volume maximal $V_{max} = 44\,\text{cm}^3$. Le volume minimal correspond au point mort (fente longue en face de la fourche optique inférieure). Au passage d'une fente, la tension aux bornes de la fourche devient égale à 5 V. Un code Arduino permet alors de remonter au nombre de fronts montants i de la tension depuis le point mort haut, et finalement au volume (pour $1 \leq i \leq N$)

$$V = \frac{V_{max} - V_{min}}{2} \sin\left(\frac{2\pi(i-1)}{N}\right) + \sqrt{\left(\frac{V_{min} + V_{max}}{2}\right)^2 - \left(\frac{V_{max} - V_{min}}{2}\right)^2 \cos^2\left(\frac{2\pi(i-1)}{N}\right)}. \tag{2}$$

1. Il s'agit du nombre de bits sur lesquels les entrées analogiques de la carte Arduino UNO sont codées.



(a) Carte Arduino.



(b) Vue d'ensemble du moteur de Stirling.

FIGURE 1 – (a) Carte Arduino (modèle UNO) fournie avec le moteur Stirling. Elle comprend plusieurs entrées analogiques (de A0 à A3), ainsi qu'un port d'alimentation USB. Sur cet exemple, deux capteurs sur les trois (les deux fourches optiques et le capteur de pression) sont branchés. (b) La grande poulie d'entraînement est reliée par un arbre de rotation à un volant d'inertie qui joue le rôle de hacheur optique. Le passage des fentes courtes est détecté par la fourche optique du bas ; le passage de la fente longue (point mort haut) est repéré par la fourche optique du haut.

Dans la formule précédente, c'est une fonction sinus qui intervient, car l'indice de la fente est réinitialisé à 1 lorsque la fente longue est détectée par la fourche supérieure, alors que le point mort haut (volume minimal) correspond au passage de la fente longue dans la fourche inférieure. Par ailleurs, les positions angulaires des deux fourches sont décalées de $\pi/2$.

3 Utilisation du moteur

- On se sert du réservoir d'éthanol comme source chaude en brulant la mèche.
- Il faut alors attendre un peu pour que l'extrémité du tube soit à température suffisamment haute. On peut alors lancer le moteur en faisant tourner manuellement le volant (dans le sens indiqué par une flèche sur le volant d'inertie), qui se met alors à tourner tout seul.
- Le code Arduino donné dans ci-joint permet alors de remonter au cycle parcouru par le gaz de la chambre dans le diagramme (p, V) et donc au travail mécanique, tandis que la mesure de la masse d'éthanol brûlée permet de remonter au transfert thermique de la source chaude, et finalement au rendement (connaissant l'enthalpie de combustion de l'éthanol). Le code Arduino produit un document texte à trois colonnes dont voici un exemple.

```
n   i   Up

315 46 373
315 47 358
315 48 351
315 49 350
315 50 359
316 1 378
316 2 410
316 3 456
```

- L'exploitation des données du port série de la carte Arduino peut se faire avec le code Python également ci-joint. Un exemple de tracé de cycle est donné ci-joint (Fig. 2), tandis qu'un exemple de sortie du code

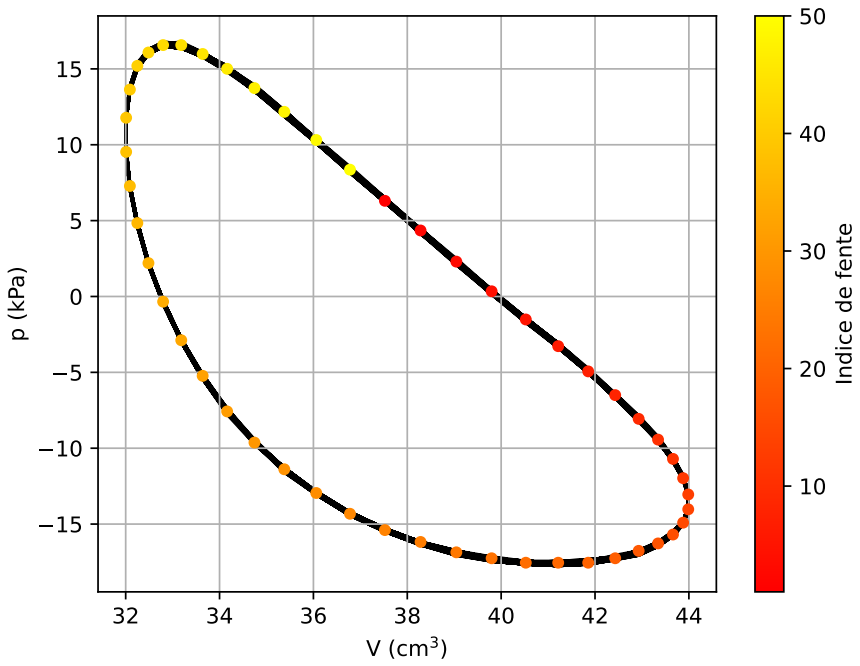


FIGURE 2 – Cycle du moteur de Stirling obtenu après traitement des données du port série de la carte Arduino.

Python est donnée ci-dessous. Le cycle est parcouru dans le sens horaire, comme cela est attendu pour un moteur. L'aire du cycle donne le travail W fourni par cycle par le moteur. Il est possible d'estimer le transfert thermique reçu pendant un cycle Q par l'enthalpie de combustion de la masse d'éthanol brûlée par cycle. On trouve typiquement un rendement $\eta = W/Q$ très faible (inférieur à ou de l'ordre de 1 %), parce qu'une grande partie de la chaleur libérée par la combustion de l'éthanol n'est pas captée par le gaz.

```
Travail moyen fourni durant un cycle = 0.18810838524431145 J.
Transfert thermique moyen reçu durant un cycle = 63.47672209026168 J.
Rendement moyen = 0.002963423110866183.
```

- Le moteur de Stirling, par un système de poulies, fait tourner une génératrice à courant continu, qu'on peut brancher à un composant électrique afin d'étudier le comportement en charge. On pourra, par exemple, utiliser une diode électroluminescente `P29.25`. Il faut alors placer l'interrupteur associé à la génératrice vers le bas (circuit fermé).

4 Précautions d'utilisation

- Le branchement des capteurs avec les fils de connexion à la carte Arduino est difficile : il est préférable de laisser les fils constamment branchés à la carte Arduino.
- Le moteur présente un sens privilégié de rotation, non indiqué dans la notice, mais que nous avons marqué par une flèche sur le volant d'inertie.
- La sensibilité du capteur de pression n'est pas indiquée correctement dans les notices. Il faut donc se référer uniquement à l'Éq. (1).
- Le code Arduino donné dans la notice n'est pas utilisable en l'état, car le *baud number* (nombre de bits par seconde pour la transmission de données vers le port série) indiqué est insuffisant pour un fonctionnement normal du moteur. Il faut donc utiliser le code ci-joint qui, lui, donne des résultats acceptables. Le *baud number* peut être diminué jusqu'à 57 600, mais pas en deçà.

5 Codes

5.1 Code Arduino

```
// Définition des constantes
const int PinPres = A0; // Déclaration de l'entrée analogique A0 comme celle du capteur de pression
const int PinPMH = A1; // Déclaration de l'entrée analogique A1 comme celle de la fourche supérieure
const int Pinbroche = A2; // Déclaration de l'entrée analogique A2 comme celle de la fourche inférieure
int nbrecycle = 0; // Compteur du nombre de cycles
int nbrefentes = 0; // Compteur du nombre de fentes parcourues dans un cycle
int etatphotogatePMH; // État de la fourche du point mort haut
int etatphotogate; // État de la fourche inférieure
int memoirePMH = LOW; // État garde en mémoire de la fourche du point mort haut
int memoire = LOW; // État garde en mémoire de la fourche inférieure

void setup() {
  // Initialisation
pinMode(PinPres,INPUT); // Déclaration de la variable comme variable d'entrée
pinMode(PinPMH,INPUT_PULLUP); // Déclaration de la variable comme variable d'entrée
// input_pullup permet d'éliminer le bruit de détection
pinMode(Pinbroche,INPUT_PULLUP); // Déclaration de la variable comme variable d'entrée
Serial.begin(115200); // Déclaration du port série
Serial.print("# Nbre cycles");
Serial.print('\t');
Serial.print("Nbre fentes");
Serial.print('\t');
Serial.print("Pres");
Serial.println();
}

void loop() {
  // Boucle
etatphotogatePMH=digitalRead(PinPMH); // Lecture de l'état de la fourche supérieure
etatphotogate=digitalRead(Pinbroche); // Lecture de l'état de la fourche inférieure
// Si l'état de la fourche supérieure est haut et qu'il était bas avant,
// alors incrémente le nombre de cycles de 1 et réinitialise le nombre de fentes à 0
if ((etatphotogatePMH != memoirePMH) && (etatphotogatePMH == HIGH))
{nbrecycle++;
nbrefentes = 0;
}
// Si l'état de la fourche inférieure est haut et qu'il était bas avant,
// alors incrémente le nombre de fentes de 1
// et affiche les valeurs des différents compteurs et celle donnée par le capteur de pression
if ((etatphotogate != memoire) && (etatphotogate == HIGH) && (nbrecycle>=1))
{nbrefentes++;
Serial.print(nbrecycle);
Serial.print('\t');
Serial.print(nbrefentes);
Serial.print('\t');
Serial.print(analogRead(PinPres));
Serial.println();
}
// Initialise les nouvelles valeurs à garder en mémoire pour les états des deux fourches
memoire = etatphotogate;
memoirePMH = etatphotogatePMH;
}
```

5.2 Code Python

```
# -*- coding: utf-8 -*-
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

# Paramètres à ne pas modifier
N=50 # Nombre de fentes
Vmin=32 # Volume minimal en cm3
Vmax=44 # Volume maximal en cm3
```

```
nb=10 # Nombre de bits
dp=100 # Différence de pression maximale en kPa
L=29693 # chaleur latente de combustion de l'éthanol en J/g

# Paramètres à modifier
m=0. # masse d'éthanol brûlée en g
nom_fichier='cycle.pdf' # nom du fichier d'enregistrement de la figure du cycle

# Extraction des données
# Première colonne = numéro du cycle
# Deuxième colonne = numéro de la fente dans le cycle
# Troisième colonne = pression codée sur 10 bits
data=np.genfromtxt('cycle_v2') # 'cycle_v2' est à remplacer par le nom du fichier dans le dossier
    courant
cycles=data[:,0]
fentes=data[:,1]
p=dp*(data[:,-1]/(2**nb-1.)-0.5) # Calcul de la pression en kPa
p*=1e3 # Conversion de la pression en Pa
V=0.5*(Vmax-Vmin)*np.sin(2.*np.pi*(fentes-1.)/N)+0.5*np.sqrt(
(Vmin+Vmax)**2-(Vmin-Vmax)**2*np.cos(2.*np.pi*(fentes-1.)/N)**2) # Calcul du volume en cm3
V*=1e-6 # Conversion du volume en m3

# Élimination des premières données qui ne correspondent pas à un cycle entier
j=0
while(data[j,1]>1):
    j+=1
p=p[j:]
V=V[j:]
fentes=fentes[j:]
cycles=cycles[j:]
cycles+=-cycles[0]+1 # Initialisation de l'indice du premier cycle à 1

# Élimination des dernières données qui ne correspondent pas à un cycle entier
j=-1
while (data[j,1]<50):
    j-=1
p=p[:j]
V=V[:j]
fentes=fentes[:j]
cycles=cycles[:j]

# Calcul des différents échanges d'énergie
W=np.trapz(p,V) # Calcul du travail fourni en J
Qc=m*L # transfert thermique reçu de la source chaude en J
eta=W/Qc # rendement du moteur
print('Travail moyen fourni durant un cycle = {0} J.'.format(W/cycles[-1]))
print('Transfert thermique moyen reçu durant un cycle = {0} J.'.format(Qc/cycles[-1]))
print('Rendement moyen = {0}.'.format(eta))

# Tracé des cycles dans le diagramme de Watt
# On représente également un cycle typique pour marquer le sens d'évolution du cycle
plt.grid()
plt.scatter(V[N*int(cycles[-1]/2.):N*int(cycles[-1]/2.)+N]*1e6,p[N*int(cycles[-1]/2.):N*int(cycles
    [-1]/2.)+N]*1e-3,c=fentes[N*int(cycles[-1]/2.):N*int(cycles[-1]/2.)+N],s=20,cmap='autumn',zorder=2)
plt.plot(V*1e6,p*1e-3,zorder=1,color='k')
plt.colorbar(label='Indice de fente')
plt.xlabel(r'V (cm$^3$)')
plt.ylabel('p (kPa)')
plt.savefig(nom_fichier)
plt.show()
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Utilitaires de post-traitement sous python de données échantillonnées par un Arduino et
transférées via le port série

Auteurs: C. Winisdoerffer (17/03/2022)
Modifications par S. Al-Jibouri, M. Viallet et M. Tandt (06/04/2022) :
    Correction du baudrate
    Correction du code d'oscilloscope pour qu'il s'affiche
    Upgrade du code d'ouverture du port arduino pour qu'il systématiquement
    Ajout d'une fermeture du port après les mesures pour éviter qu'il ne soit pas utilisable
    lors de l'execution suivante
    Ajout de mesure de temps d'execution pour l'acquisition en direct et en différé
"""

# Importation des librairies
import numpy as np
import matplotlib.pyplot as plt
import sys, time, argparse, collections
import serial

# Definition des fonctions
def progress_bar(progress):
    """ Barre de visualisation de la progression """
    barLength = 10              # longueur de la barre
    progress  = float(progress) # conversion (éventuelle)
    block = int(round(barLength*progress))
    text = "\rPourcentage: [{}] %".format( "#"*block + "-"*(barLength-block))
    #text = "\rPourcentage: [{}] {}%".format( "#"*block + "-"*(barLength-block), progress*100)
    sys.stdout.write(text)
    sys.stdout.flush()
    return

def get_serialport(baudrate, port, argv):
    """Creation de l'objet serialport,
    type(port) = str"""
    if argv[1:]:
        # Defining valid arguments and embed them into a parser
        parser = argparse.ArgumentParser()
        parser.add_argument("-r", "--baudrate",   type=int, help="baudrate")
        parser.add_argument("-p", "--port",       type=str, help="port serie")
        args = parser.parse_args()
        # Parse the arguments
        if args.baudrate:   baudrate = args.baudrate
        if args.port:       port = args.port
    # Création et spécification du port série
    serialport          = serial.Serial()
    serialport.baudrate = baudrate
    serialport.port     = port
    return serialport

def manage_SerialPort(argv):
   """ Gestion du port série """
   # Définition de valeurs par défaut
   baudrate = 115200   # doit être identique à celui défini dans le script Arduino
   port = ""
   i = 0
   while i < 10 and port == "":

       port     = 'COM{}'.format(i) # port série utilisé pour établir la comm. ordi-Arduino
       # Traitement des arguments optionnels passés sur la ligne de commande
       serialport = get_serialport(baudrate, port, argv)


       try :
           serialport.open()
       except :
           port = ""
           i+=1
   if port == "":
```

```
        port = input("Entrez un nom de port valide :")
        serialport = get_serialport(baudrate, port,argv)
    print('Port série: baudrate = {}, port = {}'.format(baudrate, port))
    return serialport

def check_SerialPort(serialport):
    """ Teste la communication avec le port série """
    # Ouverture du port série (le cas échéant)
    if not(serialport.isOpen()):
        serialport.open()                    # ouverture du port serie
        time.sleep(0.1)
    # Purge du port série
    serialport.reset_input_buffer()
    # Test du port série
    try :
         print("Lecture d'une ligne [affichage brut]: {}".format(serialport.readline()))
         time.sleep(1)
    except:
         print('Problème de réception des données')
    return

def detect_Fields(serialport, sentinel):
    """ Détermine le nombre de données présentes sur une ligne transmise par le port série """
    guess = []
    while len(guess) < 10:
        try:
            # Recuperation des donnees ligne a ligne
            data_byte   = serialport.readline()
            # Conversion binaire --> chaine de caract.
            data_string = data_byte.decode('ascii')
            if sentinel:
                if data_string[:len(sentinel)] == sentinel: # la ligne commence correctement
                    guess += [len(data_string[len(sentinel):].split('\t'))]
            else:
                guess += [len(data_string.split('\t'))]
        except:
            pass
    # Sanity check
    # todo
    Nfields = collections.Counter(guess).most_common(1)[0][0]
    print('Nfields = {}'.format(Nfields))
    return Nfields

def getdata_static(serialport, imax=100, sentinel=None):
    """ Récupère un jeu de données de longueur prédéfinie (default: imax=100) et le sauve
    dans un fichier.
        Idéal pour le post-traitement.
        L'argument optionnel 'sentinel' permet de vérifier qu'on récupère une ligne complète.
        """
    # Warming up
    check_SerialPort(serialport)
    Nfields = detect_Fields(serialport, sentinel)
    # Initialisation
    data = np.nan * np.ones((imax, Nfields))
    # Message
    print("* Récupération de {} données consécutives. Pour interrompre le programme, Ctrl+C".
    format(imax))
    # Récupération des lignes consécutives
    start = time.time()
    for i in range(imax):
        try:
            # Recuperation des donnees ligne a ligne
            data_byte   = serialport.readline()
            # Conversion binaire --> chaine de caract.
            data_string = data_byte.decode('ascii')
            if sentinel:
                if data_string[:len(sentinel)] == sentinel: # la ligne commence correctement
                    data[i,:] = np.fromstring(data_string[len(sentinel):], dtype=float, sep="\t")
            else:
                data[i,:] = np.fromstring(data_string, dtype=float, sep="\t")
            # Affichage d'une barre de progression
            progress_bar(float(i+1)/imax)
```

```
            except KeyboardInterrupt: # correspond a un Ctrl + C
                print("Lecture interrompue")
                return
        end = time.time()
        executiontime = end-start
        print("\nTemps d'acquisition : {:4f}".format(executiontime))
        # Sauvegarde
        serialport.close()
        fname = 'data.{}.txt'.format(time.strftime("%y%m%d.%H%M"))
        print('\n* Sauvegarde des donnees dans le fichier: {}'.format(fname))
        np.savetxt(fname, data)
        return data

def getdata_dynamic(serialport, imax=100, iblck=13, sentinel=None, yrange=None):
    """ Récupère les données et les visualise "comme sur un oscillo"
        Adapté de A. Bérut, https://github.com/aberut/arduino-oscillo) """
    # Warming up
    check_SerialPort(serialport)
    Nfields = detect_Fields(serialport,sentinel)
    # Initialisation
    blck = np.zeros((iblck, Nfields))    # les données sont affichées par blocs (de longueur
    iblck)
    data = np.zeros((imax, Nfields))
    # Oscilloscope
    fig   = plt.figure()
    ax    = fig.add_subplot(111)
    # Voies de l'oscillo
    channels = [[] for i in range(Nfields)]
    for ifield in range(Nfields):
        channels[ifield], = ax.plot(data[:,ifield])
    # Curseur repérant le dernier point tracé
    cursor, = ax.plot([0,0],[0,2**10-1],'r--')
    # min et max de l'axe des ordonnées
    if yrange:
        ax.set_ylim(yrange[0],yrange[1])
    else:
        ax.set_ylim(0,2**10-1)
    ax.grid()
    # Message
    print("* Récupération des données. Pour interrompre le programme, Ctrl+C")
    ibeg = 0
    iend = 0
    t0=time.time()
    while True:
        try:
            for i in range(iblck):
                # Recuperation des donnees ligne a ligne
                data_byte   = serialport.readline()
                # Conversion binaire --> chaine de caract.
                data_string = data_byte.decode('ascii')
                if sentinel:
                    if data_string[:len(sentinel)] == sentinel: # la ligne commence correctement
                        blck[i,:] = np.fromstring(data_string[len(sentinel):], dtype=float, sep=
                        "\t")
                else:
                    blck[i,:] = np.fromstring(data_string, dtype=float, sep="\t")
            # Sauvegarde du dernier bloc de données à la suite du préc. avec le modulo
            ibeg = iend
            iend += iblck
            if iend <= imax:
                data[ibeg:iend,:] = blck[:,:]
            else:
                delta_i = imax-ibeg
                data[ibeg:imax,:]       = blck[:delta_i,:]
                data[0:iblck-delta_i,:] = blck[delta_i:,:]
                iend = iblck-delta_i
            # Visualisation
            for ifield in range(Nfields):
                channels[ifield].set_ydata(data[:,ifield])
            cursor.set_xdata([iend,iend])
            plt.pause(0.01)
            t1=time.time()-t0#on mesure le temps écoulé depuis le début de l'acquisition des
```

```
            données
            plt.title('Temps écoulé : {:.4} s'.format(t1))
            fig.canvas.draw()
        except KeyboardInterrupt: # correspond a un Ctrl + C
            print("Lecture interrompue")
            return
    serialport.close()
    return

# Programme principal
if __name__ == "__main__":

    pass
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Traitement des données échantillonnées par un Arduino et transférées via le port série
Usage: python python_AnalogReadSerial.py
       python python_AnalogReadSerial.py -p '/dev/ttyACM0' [Linux]

Auteurs: C. Winisdoerffer (21/03/2022)
Modifications par S. Al-Jibouri, M. Viallet et M. Tandt (06/04/2022)
    Epuration du code : Mise en commentaire des morceaux d'analyse pour ne garder que la
    génération de données
"""

# Importation des librairies
import numpy as np
import matplotlib.pyplot as plt
import sys
import python_ArduinoUtils as AU
import scipy.optimize as scopt
import time


# Definition des fonctions
def convertdata(data, Nbits=10):
    """ Conversion des données brutes (issues de l'arduino) en données physiques """
    # Transformation affine
    data = 5.* data/(2**Nbits-1.)
    return data

def postprocess(data):
    """ Exemple de post-traitement des données : recherche d'un ajustement harmonique """
    x = data[:,0]
    y = data[:,1]
    # Initial guess
    coeff = np.array([np.mean(y), np.mean(y)/10, 1.e-4, 0.])
    # Fit
    lsq = scopt.least_squares(residuals, coeff, args=(x,y))
    lsq = lsq.x # keep only the optimized parameters among the outputs
    print('Best fit parameters: {}'.format(lsq))
    # Visualisation
    plt.plot(x,y,'-', label='data')
    plt.plot(x,functionnal(x, coeff), '-', label='initial guess')
    plt.plot(x,functionnal(x, lsq), '-', label='best fit')
    plt.legend()
    plt.show()
    return

def functionnal(x, coeff):
    """ Forme fonctionnelle pour le fit """
    # cste + A * sin(2\pi \nu x + \phi)
    res = coeff[0] + coeff[1]*np.sin(2.*np.pi*coeff[2]*x+coeff[3])
    return res

def residuals(coeff, x, y):
    """ Residuals to be used in lsq fitting """
    err = y - functionnal(x, coeff)
    return err

# Programme principal
if __name__ == "__main__":
    # Accès au port série par le script python
    serialport = AU.manage_SerialPort(sys.argv)

    # Récupération dynamique des données (pour visualisation)
    if False:
        #AU.getdata_dynamic(serialport)
        AU.getdata_dynamic(serialport,yrange=[0,50])
     # Récupération statique des données (pour post-traitement)
    if True:
        start = time.time()
        data = AU.getdata_static(serialport)
```

```
# data = AU.getdata_static(serialport,sentinel="#")
# # Conversion
# #data = convertdata(data)
# data[:,1:] = convertdata(data[:,1:])
# # Visualisation
# #plt.plot(data)
# plt.plot(data[:,0],data[:,1])
# plt.show()
# # Post-traitement des données
# if True:
#    postprocess(data)
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Gestion d'un moteur Stirling equipe d'un capteur de pression et de 2 fourches optiques

Auteurs: B. Guiselin & E. Brillaux (27/02/2022)
         C. Winisdoerffer (17/03/2022)
         Modifications par S. Al-Jibouri, M. Viallet et M. Tandt (06/04/2022) :
             Correction de la formule pour le volume à partir du nombre de fentes (pour que
             i = 1, 50 soit le volume minimal),
             Correction de l'affichage du numero d'indice pour que la reinitialisation du
             volume se fasse à i = 1 et pas i = 26
             Corrections additionnelles pour que l'indice de départ soit le bon sur le cycle.
             Correction de style pour l'affichage du cycle car jaune -> rouge est plus
             intuitif que rouge -> jaune
"""

# Importation des librairies
import numpy as np
import matplotlib.pyplot as plt
import time

# Definition des fonctions
def basic_parameters():
    """ Définition des paramètres techniques du moteur Stirling de l'ENS Lyon """
    # Paramètres à ne pas modifier
    Nfentes = 50      # Nombre de fentes
    Vmin    = 32.     # Volume minimal en cm3
    Vmax    = 44.     # Volume maximal en cm3
    deltaP  = 100.    # Différence de pression maximale en kPa
    Nbits   = 10      # Nombre de bits du CAN de la carte arduino
    return Nfentes, Vmin, Vmax, deltaP, Nbits

def read_datafile(fname):
    """ Récupération des données contenues dans le fichier fname """
    try:
        data = np.genfromtxt(fname)
    except ImportError:
        raise ImportError("Impossible d'accéder au fichier {}".format(fname))
    return data

def process_data(data):
    """ Traitement des données et conversion """
    # Le format des données est supposé être:
    #    1ère colonne = numéro du cycle
    #    2ème colonne = numéro de la fente dans le cycle
    #    3ème colonne = pression (codée sur 10 bits)
    cycles = data[:,0]
    fentes = data[:,1]
    press  = data[:,2]
    # Recuperation des paramètres du dispositif mis en œuvre
    Nfentes, Vmin, Vmax, deltaP, Nbits = basic_parameters()
    # Conversion de la pression "sur 10 bits" en pression "en Pa"
    press  = 1.e3 * deltaP * (press/(2**Nbits-1.)-0.5)
    # Calcul du volume de la chambre
    arg = 2.*np.pi*(fentes-1.)/Nfentes
    V = -0.5*(Vmax-Vmin)*np.cos(arg) + 0.5*np.sqrt((Vmin+Vmax)**2-(Vmin-Vmax)**2*np.sin(arg)**
    2)
    V *= 1.e-6  # conversion cm3 --> m3
    # Extraction de cycles entiers
    ibeg = np.where(fentes == 1)[0][0]          # debut du 1er cycle complet
    iend = np.where(fentes == Nfentes)[0][-1]   # fin du dernier cycle complet
    Ncycles = (iend-ibeg+1)/Nfentes
    return Ncycles, Nfentes, press[ibeg:iend], V[ibeg:iend], fentes[ibeg:iend]

def compute_thermo(Ncycles, press, V):
    """ Considérations thermodynamiques """
    # Calcul du travail fourni (en J)
    W = np.trapz(press,V)
    print('Travail moyen fourni durant un cycle = {} J (moyenne sur {} cycles)'.format(W/
    Ncycles, Ncycles))
```

```
    # Transfert thermique reçu de la source chaude (en J)
    m = input("Masse d'éthanol brûlé (en g): ")
    try:
        m   = float(m)
    except ValueError:
        print('Réponse non valable --> 10 g par défaut')
        m = 10.
    L   = 29693. # chaleur latente de combustion de l'éthanol en J/g
    Qc  = m*L    # transfert thermique reçu de la source chaude en J
    eta = W/Qc   # rendement du moteur
    print('Transfert thermique moyen reçu durant un cycle = {} J (moyenne sur {} cycles)'.
    format(Qc/Ncycles, Ncycles))
    print('Rendement moyen = {}.'.format(eta))
    return

def plot_data(fname, Nfentes, press, V):
    """ Visualisation des données dans le diagramme de Watt """
    plt.grid()
    # Visualisation de l'ensemble des données
    plt.plot(1.e6*V,1.e-3*press,zorder=1,color='k')
    # Visualisation d'un cycle typique (celui 'du milieu')
    ibeg = 0 #int(Nfentes*Ncycles/2)
    iend = ibeg + Nfentes
    plt.scatter(1.e6*V[ibeg:iend],1.e-3*press[ibeg:iend],c=np.arange(Nfentes),s=40,cmap=
    'autumn_r',zorder=2)
    # Decorum
    plt.colorbar(label='Indice de fente')
    plt.xlabel(r'V (cm$^3$)')
    plt.ylabel('p (kPa)')
    if True:
        plt.savefig(fname+time.strftime("%y%m%d.%H%M")+".png")
    plt.show()
    return

# Programme principal
if __name__ == "__main__":
    # Interaction avec l'utilisateur
    fname = input("Nom du fichier contenant les données: ")
    # Récupération des données brutes
    data = read_datafile(fname)

    # Couples (pression, volume) (en unités SI) sur des cycles complets
    Ncycles, Nfentes, press, V, fentes = process_data(data)

    # Calculs de considérations thermodynamiques
    compute_thermo(Ncycles, press, V)

    # Visualisation
    plot_data(fname, Nfentes, press, V)
```