Vade-mecum Python
Importation de données expérimentales

Dans cette partie, on cherche à importer dans notre programme des données issues de l'expérience.
Plusieurs cas sont à envisager :

— Les données proviennent d'un microcontrôleur (type ArduinoTM, Micro :bitTM...) ; Python est alors capable de communiquer via le port série avec la carte et de réceptionner les mesures faites par la carte.
Cela va même plus loin puisque la communication peut se faire dans les deux sens : Python peut commander le microcontrôleur (par exemple pour démarrer une acquisition) et ensuite réceptionner les données qui proviennent de la carte.
Ce cas ne sera pas étudié ici : le code est beaucoup plus lourd.

— On peut aussi rentrer directement les données expérimentales dans notre programme : on construit des listes contenant nos mesures. Cela peut allonger énormément le nombre de lignes de notre programme.

— En fait le cas qui va nous intéresser concerne l'importation de données issues d'un fichier `'xxx.csv'` par exemple en provenance d'AviMeca ou autre.
À noter que le fichier n'a pas besoin de l'extension en `".csv"`.

On présente ici deux méthodes :

— la première (très simple) utilise la bibliothèque `numpy` qui est présente dans toutes les distributions scientifiques de Python. Elle me vient de Sébastien Crozes.

— la seconde (très simple aussi) utilise la bibliothèque `pandas` et est très performante, merci à Vincent Chapelle pour me l'avoir présentée.

— Il existe d'autres méthodes comme celle utilisant la bibliothèque `csv` ; elle est un peu plus longue.

— Python pourrait traiter les données sans bibliothèque supplémentaire, mais l'ensemble du code est beaucoup plus long.

1 Utilisation de la bibliothèque `numpy`

Le fichier de données `data` est placé dans le même dossier que notre fichier Python.

Imaginons un fichier de données `data` se présentant ainsi :

```
# fichier de données
# donnée d1 en fonction de t
t;d1
0;1
1;3
2;5
3;9
...;...
```

Et le code Python pour en extraire les tableaux de données :

code Python

```
import numpy as np# on importe la bibliothèque numpy
file = np.loadtxt("data", skiprows = 3, delimiter = ";")
# on précise le nom du fichier de données, le nombre de lignes à "sauter" (ça peut être 0), et le délimiteur entre les champs
T = file[ :,0]# on garde tous les nombres de la première colonne
D1 = file[ :,1]# on garde tous les nombres de la seconde colonne
print(T)# on peut afficher T...
print(T/D1)# ou faire des calculs à partir de ces tableaux...
plt.plot(T,D1)# ou tracer un graphe
```

REMARQUES SUR CE PROGRAMME :

1. Par défaut, le paramètre skiprows est à 0 et le "delimiter" est un espace. Dans ce cas, on a un fichier de données brutes dont les éléments de chaque ligne sont séparés par des espaces, et la seconde ligne de notre code peut devenir alors : `file = np.loadtxt("data")`... dur de faire plus simple.

2. Pour cet exemple et pour la suite, Python reconnaît tout type de chaîne de caractères comme délimiteur ; les plus classiques sont :
   — la virgule `","`
   — la tabulation `"\t"`
   — le point-virgule `";"`
   — l'espace `" "`
   Ce sont les mêmes délimiteurs qui sont compris par la plupart des tableurs lors de l'ouverture d'un fichier `.csv`.

3. Que signifie `T = file[:,0]` ? Les `" :"` permettent de découper une liste (ou tableau) entre 2 limites ; `L[1:3]` découpe une liste L pour ne garder que les éléments 1 et 2 (3 exclu). Quand on ne précise pas les limites, cela correspond aux extrémités, `L[ : ]` garde toute la liste.
   Pour notre code, `T = file[:,0]` garde donc tous les éléments de la première colonne et les place dans T.

2 Utilisation de la bibliothèque pandas

2.1 Cas le plus simple : noms des colonnes, séparateur = virgule

On dispose d'un fichier de données nommé `data1` (l'extension .csv n'est pas obligatoire). Celui-ci est composé d'une ligne donnant le nom des colonnes et les colonnes sont séparées par des virgules :

```
col1,col2
0,1
1,2
4,5
7,8
10,11
20,30
```

On souhaite placer ces colonnes dans des tableaux qu'on nommera `T1` et `T2` pour chaque colonne et effectuer des calculs "naturels" à partir de ces derniers, comme on ferait sous un tableur (par exemple, la première colonne divisée par la seconde).

Un code possible est le suivant (ce programme est placé dans le même dossier que le fichier de données).

code Python

```
import pandas as pa
table = pa.read_csv("data1") # on ouvre en lecture le fichier de données
print(table) # pour voir le fichier de données
T1 = table["col1"] # on construit le premier tableau contenant les éléments de la première colonne
T2 = table["col2"]
print(T1) # pour afficher la première colonne
d = T1/T2 # on crée un tableau correspondant à la division des termes de la première colonne par ceux de la seconde
print(d) # on affiche le résultat
```

2.2 Cas 2 : pas de nom de colonnes, séparateur = virgule

On dispose d'un fichier de données nommé `data2`.

```
0,1
1,2
4,5
7,8
10,11
20,30
```

On souhaite placer ces colonnes dans des tableaux qu'on nommera `T1` et `T2` pour chaque colonne et effectuer des calculs "naturels" à partir de ces derniers, comme on ferait sous un tableur (par exemple, la première colonne divisée par la seconde).

Un code possible est le suivant (ce programme est placé dans le même dossier que le fichier de données).

code Python

```
import pandas as pa
table = pa.read_csv("data2", header = None) # on ouvre en lecture le fichier de données en précisant qu'il n'y a pas d'en-tête
print(table) # pour voir le fichier de données
T1 = table[0]
T2 = table[1]
print(T1) # pour afficher la première colonne
d = T1/T2 # on crée un tableau correspondant à la division des termes de la première colonne par ceux de la seconde
print(d) # on affiche le résultat
```

2.3 Cas 3 : en-tête + noms des colonnes, séparateur = ";"

On dispose d'un fichier de données nommé `data3` :

```
ceci est le fichier de données
col1;col2
0;1
1;2
4;5
7;8
10;11
20;30
```

On souhaite placer ces colonnes dans des tableaux qu'on nommera `T1` et `T2` pour chaque colonne et effectuer des calculs "naturels" à partir de ces derniers, comme on ferait sous un tableur (par exemple, la première colonne divisée par la seconde).

Un code possible est le suivant (ce programme est placé dans le même dossier que le fichier de données).

code Python

```
import pandas as pa
table = pa.read_csv("data3", delimiter=";", header = 1) # on ouvre en lecture le fichier de données en précisant qu'il y a 2 lignes d'en-tête (Python numérote à partir de zéro)
print(table) # pour voir le fichier de données
T1 = table["col1"]
T2 = table["col2"]
print(T1) # pour afficher la première colonne
d = T1/T2 # on crée un tableau correspondant à la division des termes de la première colonne par ceux de la seconde
print(d) # on affiche le résultat
```

3 Conclusion

Ces morceaux de code fonctionnent sur tout type de fichiers de données ; les choses à modifier au cas par cas sont : le nombre de lignes du préambule (dans le paramètre header pour la méthode 2 ou dans le paramètre skiprows dans la méthode 1).

Ces méthodes sont les plus intéressantes pour 2 raisons :
— elles sont les plus courtes
— elles transforment directement les colonnes en tableaux ou assimilés ce qui permet de réaliser des calculs type "tableur" sur les différentes colonnes.

L'énorme avantage du traitement par Python plutôt que par un tableur pour les fichiers de données est le suivant : imaginez un fichier de données comportant des centaines voire des milliers de lignes de données. Le traitement par Python est instantané et ne nécessite aucune manipulation sur le fichier de données contrairement au traitement sur tableur qui serait beaucoup plus laborieux. C'est notamment dans ce cas de figure que le traitement par un un langage de programmation type Python prend tout son sens.

Puis, à la suite de ces manipulations, on peut représenter les graphes d'une grandeur en fonction d'une autre par la fonction `plot` du paquet `matplotlib.pyplot`.