TP11 : Analyse de données avec Python

1 Objectif du TP

L'objectif de ce TP est de montrer comment on peut utiliser le langage de programmation python avec les bibliothèques `numpy` et `matplotlib` pour analyser les données d'une expérience physique.

2 Les données

Les données que nous allons analyser dans ce TP correspondent à celles qui auraient pu être collectées lors de la mesure de la caractéristique d'une photorésistance. Pour plusieurs valeurs de l'éclairement, on a mesuré la tension aux bornes de la photorésistance en fonction de l'intensité qui y circule. Il va falloir commencer par regarder les données et la manière dont elles sont stockées.

- Télécharger le fichier `mesures.zip` et le décompresser dans un répertoire connu sur votre ordinateur. Vous devriez obtenir une série de fichiers nommés `mesure_nnnnn.csv` où le nombre `nnnnn` correspond à l'éclairement en lux pour cette mesure. Dans la suite, on supposera que les fichiers ont été extraits dans le répertoire

  `C:\Utilisateurs\nono\Documents\mesures\`

Le format `csv` (*Comma Separated Values* ou *Valeurs séparées par des virgules*) est un format de fichier très simple qui est souvent pris en charge par les logiciels de tableurs comme LibreOffice (ou OpenOffice) Calc ou Microsoft Excel, vous devriez donc pouvoir ouvrir ces fichiers dans le tableur de votre choix. il est cependant intéressant d'ouvrir ce fichier avec un éditeur de texte simple (le Bloc-Notes de Windows ou autre logiciel équivalent). Si vous ouvrez le fichier `mesure_00010.csv`, vous devriez voir ceci :

```
-5.006 ; -0.249
-4.952 ; -0.246
-4.903 ; -0.246
-4.848 ; -0.243
-4.797 ; -0.240
...
```

Le fichier est composé d'un grand nombre de lignes, chacune contenant deux nombres séparés par un point-virgule (et pas un virgule comme on aurait pu s'y attendre). Le premier nombre correspond à la tension en V et le second à l'intensité en mA.

3 Affichage des mesures

On va utiliser les bibliothèques `numpy` et `matplotlib` de Python pour charger et afficher le graphique qui représente la tension en fonction de l'intensité.

3.1 La bibliothèque `numpy`

`numpy` est une bibliothèque Python qui permet de manipuler des données numériques (surtout mais pas seulement) de façon très efficace (rapidité de calcul) et pratique (complexité du code). Elle permet notamment d'appliquer des opérations mathématiques directement à une liste de nombre sans avoir à écrire de boucle explicitement.

On peut faire par exemple

```
import numpy as np
x = np.array([1,2,3,4,5])
y = x**2
```

Le tableau `y` contiendra alors les nombres `[1, 4, 9, 16, 25]`.

3.2 Chargement des données

Ensuite, nous allons charger les données du fichier de mesures dans une variable python, pour cela on utilise la fonction `genfromtx` de `numpy` :

```
BASE = "C:\\Utilisateurs\\nono\\Documents\\mesures\\"
ui = np.genfromtxt(BASE+"mesure_00010.csv", delimiter=";")
```

La variable `ui` est un tableau `numpy` a deux dimensions. On peut accéder à la liste des tensions et des intensités par :

```
u = ui[:,0]
i = ui[:,1]/1000 # On divise par 1000 pour convertir l'intensité en A
```

La syntaxe `ui[A,B]` permet d'extraire l'élément B de la mesure A. Par exemple `ui[0,0]` correspond à la tension de la première ligne du fichier de mesures et `ui[0,1]` est l'intensité correspondante. On peut extraire des sous-tableaux, par exemple `ui[0:5, 0]` renvoie la liste des 5 premières valeurs de tension. et `ui[:, 0]` renvoie la liste de toutes les valeurs de tension.

3.3 Affichage du graphique

Pour afficher le graphique qui représente $u = f(i)$, on utilise la bibliothèque `matplotlib` avec :

```
import matplotlib.pyplot as plt

plt.plot(i,u)
plt.show()
```

Une fenêtre devrait s'ouvrir avec le graphique.

Nous allons maintenant ajouter des barres d'erreur sur ce graphique, car évidemment les mesures expérimentales sont accompagnées d'une incertitude. Nous allons supposer (ça n'est pas forcément vrai) que l'incertitude sur u est $\delta u = 1\,\% + 50\,\text{mV}$ et l'incertitude sur i est $\delta i = 1\,\% + 10\,\mu\text{A}$. On affiche le graphique précédent avec les barres d'erreur en faisant :

```
u_err = np.abs(0.01*u+50e-3)
i_err = np.abs(0.01*i+10e-6)
plt.errorbar(i, u, yerr=u_err, xerr=i_err, capsize=3) # Essayer sans capsize pour voir l'effet
plt.show()
```

Attention, il faut fermer la précédente fenêtre pour que celle-ci puisse apparaitre.

4 Analyse des données

C'est déjà très bien de pouvoir créer un graphique, mais nous pouvons aller beaucoup plus loin !

4.1 Ajustement linéaire

Clairement les points ont l'air de s'aligner sur une droite. Nous allons déterminer son équation. `numpy` permet de faire un ajustement des données par un polynôme de degré ajustable avec la fonction `polyfit`. Comme nous voulons faire passer une droite par les points, nous allons utiliser un polynôme de degré 1.

```
a,b = np.polyfit(i, u, 1)
print("Équation de la droite : y=ax+b")
print("a = {}".format(a))
print("b = {}".format(b))
```

Pour vérifier que les résultats de cet ajustement correspondent bien aux mesures que nous avons faites, on peut tracer sur le même graphique les données et la droite calculée :

```
# Ici on utilise zorder pour s'assurer que la droite apparaitra au dessus
# des points de mesure
plt.errorbar(i, u, yerr=u_err, xerr=i_err, capsize=3, zorder=0)
x = np.array([i[0], i[-1]])   # On choisit les deux valeurs extrêmes d'intensité
y = a*x + b # On calcule les tensions correspondantes
plt.plot(x,y,linewidth=3, zorder=1) # On affiche la droite
```

```
plt.show()
```

Ici, a correspond au coefficient directeur de la droite et donc à la valeur de la résistance.

4.2 Incertitudes sur R

Nous avons maintenant accès à la valeur de la résistance mais nous aimerions avoir une idée de l'incertitude associée à cette valeur. Pour l'estimer nous allons utiliser une méthode de Monte-Carlo.

Le principe de cette méthode est de considérer que chacun des points de mesure aurait pu être à une position aléatoire de l'intervalle d'incertitude. On va alors écrire un programme qui, à partir de nos mesures et de leur incertitude, simule d'autres mesures possibles et détermine les paramètres a et b de la droite qui passe au mieux par ces points.

On obtient alors une liste de valeurs de a et b dont la moyenne nous donnera la valeur la plus probable du paramètre, et l'écart-type nous indiquera l'incertitude associée.

Pour tirer aléatoirement les points de mesure, nous considérerons une distribution gaussienne dont l'écart-type est donnée par l'incertitude associée à chaque point. (L'hypothèse d'une distribution gaussienne est discutable mais c'est souvent un bon point de départ). On écrit donc le programme suivant :

```
a=[]  # La liste des valeurs de a
b=[]  # La liste des valeurs de b
# On effectue 500 tirages aléatoire, plus ce nombre est élevé, plus
# l'estimation des incertitudes sera fiable
for k in range(500):
    # um est la valeur de u calculée à partir de la mesure et de son incertitude
    # on utilise une distribution gaussienne ("normal") centrée sur u
    # et d'écart-type égale à u_err
    um = np.random.normal(u, u_err)

    # idem pour im
    im = np.random.normal(i, i_err)

    # ajustement des données simulées
    at,bt = np.polyfit(im, um, 1)

    # ajout à la liste
    a.append(at)
    b.append(bt)

aa = np.average(a) # valeur moyenne de a
da = np.std(a)     # incertitude sur a

bb = np.average(b) # valeur moyenne de b
db = np.std(b)     # incertitude sur b
print("a={}+-{}".format(aa, da))
print("b={}+-{}".format(bb, db))
```

4.3 Analyse de toutes les données

Nous avons maintenant tous les outils pour analyser l'ensemble des fichiers de données et plutôt que de répéter une à une les opérations précédentes, nous pouvons écrire un programme qui fera tout d'un coup. Le corps du programme peut ressembler à ça :

```
import numpy as np
import matplotlib.pyplot as plt

# Liste des éclairements mesurés
lum = [10, 50, 100, 300, 400, 1000, 3000, 10000]

# Liste des valeurs de résistances et incertitudes
lR = []
ldR = []
for l in lum:
```

```
    nom_fichier = BASE+"mesure{:05d}".format(l)
    R, dR = analyse_fichier(nom_fichier)
    lR.append(R)
    ldR.append(dR)

plt.errorbar(lum, R, y_err=dR, capsize=3)
plt.show()
```

On pourra aussi essayer de tracer le graphique représentatif de $\log(R)$ en fonction de log(lum) pour établir une modélisation de l'évolution de la résistance avec l'intensité lumineuse.