Représentations graphiques avec Python™

Dans ce document, nous utilisons les bibliothèques **numpy** et **matpltotlib.pyplot** : la première permet de faire des calculs numériques et des tableaux de nombres ou « array » ; la seconde regroupe des outils permettant de tracer des graphiques.

Tracer un graphique cartésien avec Python™

- Importation des bibliothèques :

```
import matplotlib.pyplot as plt                #Pour tracer des graphiques
import numpy as np                             #Pour faire divers calculs
```

- Entrée des données expérimentales dans un tableau ou array dans le langage python. Grâce à la bibliothèque **numpy** on crée un **np.array** pour chacune des grandeurs (abscisses et ordonnées).

Par exemple, si l'on souhaite tracer l'évolution d'une grandeur y en fonction d'une grandeur x :

```
x = np.array([rentrer les valeurs de x séparées par des virgules])
#contient les valeurs x auxquelles ont été mesurées celles de y
y = np.array([rentrer les valeurs de y séparées par des virgules])
#ce sont les valeurs de y correspondantes
```

Il est important de rentrer le même nombre de valeurs de x et de y et dans le bon ordre ;

```
plt.plot(x, y, 'bo')                    #Représenter y en fonction de x avec
des points bleus ('g+' avec des croix vertes …)
plt.xlabel('légende à adapter')         #Légende axe de abscisses
plt.ylabel('légende à adapter')         #Légende axe des ordonnées
plt.title('titre à adapter')            #Titre
plt.grid()                              #Quadrillage
plt.show()                              #Affiche le graphique
```

Faire une régression linéaire avec Python™

Il y a plusieurs possibilités pour faire une régression linéaire. Nous présentons ici le script permettent de faire une régression linéaire avec la bibliothèque **numpy**.

```
#Renseigner les grandeurs x et y dans des array au préalable.

#Rajout des barres d'incertitude sur la grandeur portée en y

u_y = valeur                #Incertitude-type sur la grandeur portée en y
plt.errorbar(x, y, yerr=u_y, fmt='go') #Tracé du nuage de points en vert
avec les barres d'incertitude en y.
```

```
#Régression linéaire avec np.polyfit

p=np.polyfit(x, y, 1)    #Régression linéaire de y en fonction de x
(équation y = p[0]*x+p[1])
plt.plot(x,np.polyval(p,x))          #Tracé de la droite de régression
plt.xlabel('à adapter')
plt.ylabel('à adapter')
plt.title('à adapter')
plt.grid()
plt.show()
print('pente',p[0])                  #Affiche la pente de la droite
print('oo', p[1])                    #Affiche l'ordonnée à l'origine
```

Tracer un histogramme avec Python™

Pour tracer un histogramme avec Python™, on utilise la fonction **`plt.hist`**.

- **Nombre de classes** : le nombre de classes est optimisé avec l'argument optionnel bins='rice' *(voir infra)*. Si on veut fixer le nombre de classes à 5 (par exemple), alors on écrit bins=5 à la place.
- **Type d'histogramme** : par défaut l'histogramme tracé est un histogramme des *effectifs* : chaque ordonnée représente l'effectif dans la classe. En mathématiques, on réserve l'appellation histogramme aux diagrammes dont l'ordonnée est la *densité de fréquence*. On les interprète comme une densité de probabilité expérimentale. Dans ce cas il faut utiliser l'option density = True. On l'utilisera notamment lorsqu'on veut superposer des histogrammes.

```
#Représentation d'un histogramme des effectifs

plt.hist(x, bins='rice',color='pink')    #Représentation de l'histogramme
des valeurs x / avec optimisation du nombre de classes.
plt.title('Histogramme de x')
plt.xlabel('x en - … (préciser l'unité)')
plt.ylabel('Effectifs')
plt.show()
```