ON 2 - Régression linéaire

Ce qu'il faut savoir et savoir faire

→ Utiliser un logiciel de régression linéaire afin d'obtenir les valeurs des paramètres du modèle.
→ Analyser les résultats obtenus à l'aide d'une procédure de validation : analyse graphique intégrant les barres d'incertitude ou analyse des écarts normalisés.

I. Faire une régression linéaire

1. Algorithme Python

→ On importe la bibliothèque `numpy` avec l'alias `np`. On appellera alors les différentes fonctions de cette bibliothèque avec le préfixe np.

→ On entre les valeurs des entrées x et y sous la forme de deux **tableaux de valeurs**.

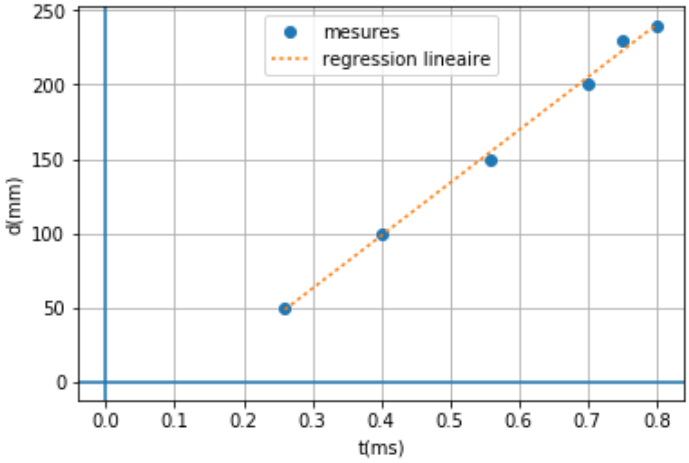
Dans la bibliothèque `numpy` se trouve la fonction **`polyfit`**, qui permet de trouver le polynôme qui colle le mieux à la courbe expérimentale. Pour une régression linéaire il s'agit de chercher un polynôme de degré 1.

→ La commande **`[a,b] = np.polyfit (x,y,1)`** renvoie les coefficients a et b qui collent le plus à la courbe y fonction de x, $y = ax + b$.

2. Exemple

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

#les mesures
x=np.array([0.26,0.40,0.56,0.70, 0.75,0.80])
y=np.array([50,100,150,200,230,240])
plt.plot(x,y,'o',label='mesures')

#regression linéaire
[a,b]=np.polyfit(x,y,1)
print("a = {:.5f}".format(a))# 5 chiffres après la virgule
print("b = {:.5f}".format(b))

#tracé
ymodel=a*x+b
plt.plot(x,ymodel,':',label='regression lineaire')
plt.legend()
plt.grid()
plt.axhline()
plt.axvline()
```

```
plt.xlabel('t(ms)')
plt.ylabel('d(mm)')
plt.show()

a = 354.88136
b = -43.57306
```



II. Validation du modèle

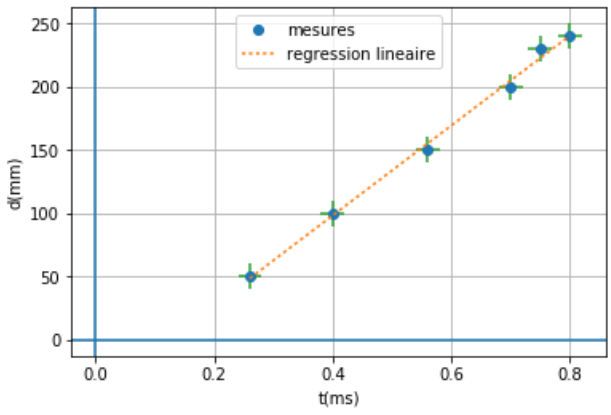
Un premier contrôle visuel permet de valider le modèle.

1. Barres d'incertitudes

La commande **`plt.errorbar( xValues, yValues, xerr = xErrorValues, yerr = yErrorValues)`** permet de tracer les barres d'incertitudes.

Dans l'exemple étudié, on suppose les précisions constantes. On rajoute à l'algorithme précédent :

```
#barres d'incertitudes
delta_x=0.02  #précision de x (ici valeur constante)
delta_y=10    #précision de y (ici valeur constante)
plt.errorbar(x, y, xerr = delta_x, yerr = delta_y,linestyle='none')
plt.show()
```



2. Estimation des incertitudes par la méthode de Monte Carlo

On veut déterminer les incertitudes-types sur a et b du modèle $y = ax + b$.

On considère une série de m mesures de couples (x_i,y_i), chacun étant dans un intervalle de demi-largeur $(\Delta x_i, \Delta y_i)$. Pour estimer l'incertitude-type du coefficient directeur a et de l'ordonnée à l'origine b, il faudrait réaliser des ensembles de nouvelles mesures $\{x_i\}$ et $\{y_i\}$ puis réaliser une nouvelle régression linéaire. En réalisant un grand nombre de fois cette opération, on obtiendra, en prenant les écarts-types, les valeurs des incertitudes-types sur les paramètres a et b. Un tel procédé est bien trop long et donc peu pratique. On utilise alors la méthode de Monte-Carlo.

1. **Lister les grandeurs expérimentales** utiles pour le calcul et associer à chacune, un intervalle au sein duquel on peut raisonnablement penser que celle-ci appartient
2. Créer deux listes vides pour stocker les pentes et les ordonnées à l'origine des régressions.
3. Réaliser un **très grand nombre N de simulations**, pour chacune :
   - **Réaliser un tirage aléatoire de m couples (x_i,y_i),** donné par une loi de probabilité uniforme, dans les intervalles $[x_i - \Delta x_i , x_i + \Delta x_i]$ et $[y_i - \Delta y_i , y_i + \Delta y_i]$.
   - **Réaliser une régression linéaire** sur cet ensemble de valeurs (m couples (x,y)) avec `polyfit`.
   - **Stocker** dans les listes la pente et l'ordonnée à l'origine de cette régression.
4. **Calculer les écarts-types** des deux listes des pentes et des ordonnées à l'origine pour **obtenir les incertitudes-types de a et b.**

On reprend les données de l'exemple précédent.

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

# Entrée des données
x=np.array([0.26,0.40,0.56,0.70, 0.75,0.80])
y=np.array([50,100,150,200,230,240])
delta_x = 0.02
delta_y = 10

# Nombre de simulations
N = 10000

# Listes des valeurs du coefficient directeur et de l'ordonnée à l'origine
A = []
B = []

# Réalisation de la simulation
for k in range(N):
    Xsim = x+np.random.uniform(-delta_x,delta_x,len(x))
    Ysim = y+ np.random.uniform(-delta_y,+delta_y,len(x))
    a,b = np.polyfit(Xsim,Ysim,1)  #régression linéaire
    A.append(a)
    B.append(b)

# Calcul des incertitudes-type
u_a=np.std(A,ddof=1)
u_b=np.std(B,ddof=1)
```

```
# Affichage des résultats avec 5 chiffres après la virgule
print("u(a) = {:.5f} ".format(u_a))
print("u(b) = {:.5f} ".format(u_b))

#Génération d'un histogramme pour voir les distributions (optionnel)
plt.hist(A,bins='rice')
plt.title('distribution des valeurs de a')
plt.figure(2)
plt.title('distribution des valeurs de b')
plt.hist(B,bins='rice')
plt.show()

u(a) = 14.76266
u(b) = 8.98458
```