

**Capacité numérique : simuler, à l'aide d'un langage de programmation ou d'un tableur, un processus aléatoire de variation des valeurs expérimentales de l'une des grandeurs – simulation Monte-Carlo – pour évaluer l'incertitude sur les paramètres du modèle. 🍃 🍃 🍃**

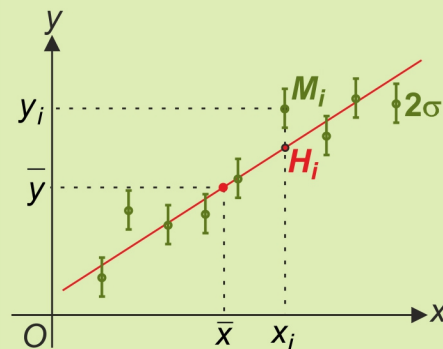
## Régression linéaire (ajustement affine)

On peut vérifier expérimentalement que deux grandeurs physiques  $X$  et  $Y$  sont compatibles avec un modèle  $Y = f(X)$ . Plaçons-nous dans le cas où on modélise la relation entre deux grandeurs physiques  $X$  et  $Y$  par  $Y = aX + b$ .

On réalise une série de  $N$  mesures indépendantes de ces grandeurs, notées  $(X_i, Y_i)$ ,  $i \in \llbracket 1, N \rrbracket$ .

On obtient  $(x_i, y_i)$ ,  $i \in \llbracket 1, N \rrbracket$ , avec des écarts-types  $(u(X_i), u(Y_i))$ ,  $i \in \llbracket 1, N \rrbracket$ . On se pose la question suivante : La relation  $Y = aX + b$  est-elle bien vérifiée ? Quelles sont alors les valeurs expérimentales de  $a$  et  $b$  et les incertitudes sur ces coefficients ?

Même si le modèle affine est valide, les points de coordonnées  $(x_i, y_i)$   $i \in \llbracket 1, N \rrbracket$  ne sont pas parfaitement alignés du fait des incertitudes de mesure.



Nous nous placerons par défaut dans le cas suivant (régression linéaire ordinaire) :

- **L'incertitude-type sur  $X$  est négligeable, soit  $|a| \cdot u(X_i) \ll u(Y_i) \forall i \in \llbracket 1, N \rrbracket$ .**
- **Les incertitudes-type  $u(Y_1) = u(Y_2) = \dots = u(Y_N) = \sigma$  sont toutes identiques.**

Ce dernier critère n'est pas souvent vérifié, ne serait-ce que parce que l'on ne se ramène souvent à une relation affine qu'après un changement de variable  $Y = g(\Gamma)$ , où  $\Gamma$  est la grandeur dont le mesurage fournit des valeurs  $\gamma_i$ ,  $i \in \llbracket 1, N \rrbracket$ . Même si  $u(\Gamma_i) = \sigma \forall i \in \llbracket 1, N \rrbracket$ , les incertitudes-type sur  $Y_i = g(\Gamma_i)$  sont différentes.

La droite recherchée est telle que la grandeur  $\chi^2 = \frac{1}{\sigma^2} \sum_{i=1}^N [y_i - f(x_i)]^2 = \frac{S}{\sigma^2}$  est minimale.

Graphiquement,  $S$  représente la somme des carrés des distances  $H_i M_i$ , où  $M_i$  est le point de coordonnées  $(x_i, y_i)$  et  $H_i$  le projeté de  $M_i$  sur la droite d'équation  $Y = aX + b = f(X)$  selon la direction  $Oy$ . Cette méthode porte en conséquence le nom de **méthode des moindres carrés**.

La fonction  $(a, b) \mapsto S = \sum_{i=1}^N [y_i - ax_i - b]^2$  passe par un minimum si  $\frac{\partial S}{\partial a} = 0$  et  $\frac{\partial S}{\partial b} = 0$ , soit :

$$\begin{cases} \sum_{i=1}^N [ax_i^2 - x_i y_i + bx_i] = 0 \\ \sum_{i=1}^N [b - y_i + ax_i] = 0 \end{cases} \Leftrightarrow \begin{cases} a \sum_{i=1}^N x_i^2 + b \sum_{i=1}^N x_i = \sum_{i=1}^N x_i y_i \\ a \sum_{i=1}^N x_i + Nb = \sum_{i=1}^N y_i \end{cases}$$

En divisant ces deux dernières relations par  $N$ , on obtient  $\begin{cases} \overline{ax^2} + b\overline{x} = \overline{xy} \\ a\overline{x} + b = \overline{y} \end{cases}$ , en notant avec une

barre les grandeurs moyennes *expérimentales*. Les meilleurs estimateurs de la pente et de l'ordonnée à l'origine de la droite de régression cherchée (droite de régression de  $y$  en  $x$ ) sont donc

$a = \frac{\overline{xy} - \overline{x} \overline{y}}{\overline{x^2} - \overline{x}^2}$  et  $b = \overline{y} - a\overline{x}$ . L'équation de la droite recherchée est  $y - \overline{y} = a(x - \overline{x})$ . Elle passe

par le point moyen  $(\overline{x}, \overline{y})$  du nuage de points.

## Déterminer l'incertitude-type sur $a$ et $b$ par la méthode de Monte-Carlo

La fonction python **polyfit** du module numpy permet de réaliser un ajustement d'une série de valeurs avec un polynôme, ici de degré 1. Elle renvoie sous forme de tableau les valeurs de la pente  $a$  et de l'ordonnée à l'origine  $b$ .

Utilisons une simulation de Monte-Carlo pour obtenir  $u(a)$  et  $u(b)$  à partir d'une série de mesures  $(x_i, y_i)$ . On se place dans le cadre où la distribution des  $Y_j$  est quelconque, avec des incertitudes-types  $\sigma$  *identiques*.

— (i) On calcule  $a$  et  $b$  grâce à **polyfit**. La droite d'ajustement a pour équation :

$$Y_{\text{ajusté}} = aX + b.$$

— (ii) On simule  $N\_MC$  jeux de mesures  $(x_i^j = x_i, y_i^j = ax_i + b + \varepsilon_i^j)$  où  $\varepsilon_i$  suit la distribution choisie, centrée sur 0 et d'incertitude-type  $\sigma$ .

— (iii) On effectue une régression linéaire sur chaque jeu de mesures et on obtient  $N\_MC$  valeurs des coefficients de la droite d'ajustement  $(a_j, b_j)$ ,  $j \in [1, N\_MC]$ .

— (iv) On en déduit les valeurs moyennes  $a^*$  et  $b^*$  et les écarts-types  $u(a^*)$  et  $u(b^*)$ .

Le code python suivant met en œuvre cette démarche et permet de tracer le nuage des points expérimentaux  $M_j(x_i, y_i)$  avec les barres d'incertitude de taille  $2\sigma$ , ainsi que les histogrammes des valeurs simulées de  $a$  et  $b$ . Les résidus normalisés  $Z_j = [y_i - (a_0^* x_i + b_0^*)] / \sigma$  sont étudiés en traçant les points de coordonnées  $(x_i, Z_j)$ .

```
import numpy as np
from matplotlib import pyplot as plt
import numpy.random as rd

def AjustAff_MC(Lx, Ly, sigma, fig = True): # partie à connaître
    p = np.polyfit(Lx, Ly, 1) # fonction numpy pour effectuer un ajustement affine, 1 correspond au degré du polynôme utilisé
    a, b = p[0], p[1] # valeurs des coefficients basés sur la réalisation expérimentale
    def ajustement(x):
        return a*x + b
    Ly_ajuste = ajustement(Lx) # valeurs ajustées de Y
    N_MC = 100000 # nombre d'expériences simulées
    a_MC, b_MC = np.zeros(N_MC), np.zeros(N_MC) # initialisation des tableaux dans lesquels on va stocker les valeurs simulées des pentes et ordonnées à l'origine
```

```

for i in range(N_MC): # pour chaque expérience simulée, on simule
une mesure yi pour chaque valeur xi
    Ly_MC = Ly_ajuste + rd.uniform(-sigma*3**0.5,sigma*3**0.5, size
= len(Lx)) # ici avec une loi uniforme, mais on peut aussi prendre une
loi normale :
    #Ly_MC = Ly_ajuste + rd.normal(0, sigma, size = len(Lx))
    # on reprend l'ajustement affine avec cette série de valeurs
    p = np.polyfit(Lx, Ly_MC, 1)
    # on stocke les valeurs des paramètres d'ajustement dans les
listes a_MC et b_MC
    a_MC[i],b_MC[i] = p[0],p[1]
    a_moy = np.mean(a_MC) # on calcule la moyenne des pentes
    u_a = np.std(a_MC, ddof = 1) # et leur écart-type
    print('paramètre a : ', a_moy)
    print('incertitude-type : u(a) = ', u_a)
    b_moy = np.mean(b_MC) # on calcule la moyenne des ordonnées à l'ori-
gine
    u_b = np.std(b_MC, ddof = 1) # et leur écart-type
    print('paramètre b : ', b_moy)
    print('incertitude-type : u(b) = ', u_b)
    if fig == True: # partie graphique ; il n'est pas nécessaire de la
mémoriser
        # nuage de points avec barres d'erreurs et droite d'ajustement
        X = [Lx[0],Lx[-1]] # valeurs extrêmes de X
        Y = [a_moy*Lx[0]+b_moy,a_moy*Lx[-1]+b_moy] # valeurs extrêmes
de Y

        plt.figure('points et ajustement affine')
        plt.plot(Lx,Ly, 'o', label = 'points expérimentaux',color =
'red')
        plt.errorbar(Lx, Ly, yerr = sigma, fmt = 'm.',ecolor =
'black',elinewidth = 3) # les barres d'incertitudes de taille 2 sigma
        plt.xlabel('$X$')
        plt.ylabel('$Y$')
        plt.legend()
        plt.plot(X,Y, "r-", label = 'ajustement affine',color = 'b',al-
pha = 0.6)
        plt.show()
        # tracé des résidus normalisés
        Z = (Ly-(a_moy*Lx+b_moy))/(sigma)
        plt.figure('résidus normalisés ajustement affine Monte Carlo')
        plt.plot(Lx, Z,'o',color = 'red')
        plt.fill_between([np.min(Lx),np.max(Lx)], y1 = -2, y2 = 2, color
= 'green', alpha = .3)
        plt.xlim(.9*np.min(Lx), 1.02*np.max(Lx))
        plt.xlabel('$X$')
        plt.ylabel('résidus normalisés de $Y$' ), plt.ylim(-3,3)
        plt.ticklabel_format(axis = 'x', style = 'sci', scilimits =
(0,0))

        plt.grid()
        plt.show()
        # tracé des histogrammes pour a et b
        plt.figure('histogrammes',figsize = (15, 4))
        plt.subplot(1, 2, 1)
        plt.hist(a_MC, bins = 'rice', color = 'red')
        plt.xlabel('valeurs de $a$ obtenues')

```

```

plt.ylabel("nombre d'apparitions")
plt.subplot(1, 2, 2)
plt.hist(b_MC, bins = 'rice', color = 'green')
plt.xlabel('valeurs de $b$ obtenues')
plt.ylabel("nombre d'apparitions")
plt.show()
return (a_MC,b_MC) # la fonction renvoie l'ensemble des valeurs simulées
pour une utilisation ultérieure

```

On prend pour exemple l'étude d'un réseau optique en incidence normale.

La formule utilisée est  $\sin i' = n\lambda$ , où  $i'$  est l'angle entre la direction pour laquelle on trouve la raie d'ordre 0 et la raie d'ordre 1 correspondant à la longueur d'onde  $\lambda$ .

$n$  est le nombre de traits par mètre que comprend le réseau. On réalise pour chaque raie de longueur d'onde connue les mesures de  $i'$ .

$\lambda(\text{nm})$	$i'(^{\circ})$	$n = \frac{\sin i'}{\lambda}$ (traits/mm)	$u(n)$
404,7	13,35	570,5	
407,8	13,45	570,4	
435,8	14,37	569,5	
491,6	16,27	569,9	
546,1	18,17	571,0	
577,0	19,23	570,8	
579,1	19,27	569,9	
623,4	20,8	569,6	

On cherche à valider (ou non) la loi affine théorique  $Y = \sin i' = n\lambda = aX + b$ , avec  $X = \lambda$ , et, si le modèle est validé, à donner une valeur de  $n$  avec son incertitude-type.

**Q.1)** Les angles  $i'$  sont mesurés grâce au goniomètre avec une tolérance de 1', ce qui signifie par exemple que lorsqu'on mesure  $i' = 13^{\circ}21'$ , on considère que la valeur réelle se trouve dans l'intervalle  $[13^{\circ}20', 13^{\circ}22']$ . En déduire l'incertitude-type sur  $i'$  en radian.

**Q.2)** Relier par la méthode analytique l'incertitude-type  $u(n)$  à  $n$ ,  $i'$ , et  $u(i')$ . Remplir la colonne du tableau donnant  $u(n)$ . Pourquoi faut-il nécessairement que  $u(i')$  soit exprimé en radian pour les applications numériques ?

$u(n)$  est-il constant ? Quelles sont les longueurs d'onde permettant d'obtenir une incertitude-type sur  $n$  la plus faible ?

Par la suite, on considérera, conformément au programme, que  $u(n)$  est constant, égal à la plus grande des valeurs du tableau.

**Q.3)** Déterminer, grâce à la méthode de Monte-Carlo, la valeur  $n^*$  du nombre de traits par mm du réseau que l'on peut proposer, ainsi que l'incertitude-type  $u(n^*)$ .

Comparer  $u(n^*)$  aux valeurs  $u(n)$  du tableau. Commenter.

Tracer le nuage de points expérimentaux avec les barres d'incertitude et la droite d'ajustement affine, les histogrammes des valeurs simulées de  $a$  et  $b$ , ainsi que les résidus normalisés en fonction de  $X = \lambda$ .

Le modèle affine est-il compatible avec les mesures effectuées ?

## Ajustement linéaire

Dans le modèle affine utilisé, le coefficient  $b$  n'est pas fixé, alors que la loi physique est une loi linéaire, et pas affine :  $Y = \sin i' = n\lambda = aX$ .

C'est ce mauvais choix de modèle qui est responsable d'une surestimation de  $u(n^*)$ , comme nous allons le voir.

Si le modèle physique choisi est linéaire :  $Y = aX = f(X)$ , on obtient, en reprenant la méthode

des moindres carrés :  $a^* = \frac{\overline{xy}}{\overline{x^2}}$ .

**Q.4)** Comme `polyfit` ne permet pas d'effectuer des ajustements linéaires, coder sous Python une fonction `linfit(Lx, Ly)` qui renvoie le coefficient  $a^*$  à partir des listes `Lx` et `Ly` contenant respectivement les valeurs  $x_i$  et  $y_i$ , avec  $i \in \llbracket 1, N \rrbracket$ .

Adapter la fonction `AjustAff_MC` en `AjustLin_MC` afin d'obtenir  $a^*$  et son incertitude-type par la méthode de Monte-Carlo. Cette fonction doit également permettre de tracer le nuage de points expérimentaux avec les barres d'incertitudes et la droite d'ajustement linéaire, les histogrammes des valeurs simulées de  $a$  et  $b$ , ainsi que les résidus normalisés.

**Q.5)** Déterminer, grâce à la méthode de Monte-Carlo, la valeur  $n^*$  du nombre de traits par mm du réseau que l'on peut proposer, ainsi que l'incertitude-type  $u(n^*)$ . Le modèle linéaire est-il compatible avec les mesures effectuées ?

Comparer l'ajustement linéaire à l'ajustement affine.

En conclusion, notamment pour les TIPE :

**Il ne faut pas prendre plus de paramètres dans le modèle utilisé que ceux présents dans la théorie.**

Si aucune théorie n'est disponible, des modèles différents peuvent être adaptés aux valeurs expérimentales. On en choisit un et on obtient alors une loi *empirique*.

Comme le programme impose d'utiliser `polyfit`, on utilisera l'ajustement affine  $Y = aX + b$  bien que la loi théorique soit linéaire, mais il est bienvenu de critiquer cette démarche.

*On reprend donc par la suite l'ajustement affine dans le cas du réseau.*

**Q.6)** Rajouter dans la fonction `AjustAff_MC` le tracé du nuage des points de coordonnées  $(a_j, b_j)$ ,  $j \in \llbracket 1, N_{MC} \rrbracket$ .

Tracer ce nuage dans le cas du réseau et commenter.

**Q.7)** On cherche maintenant à mesurer une abscisse inconnue, ainsi que son incertitude-type, à partir de la valeur expérimentale d'une ordonnée. Dans le cas du réseau optique en incidence normale, on cherche à déterminer la longueur d'onde correspondant à la raie trouvée dans la direction  $i' = i'_0 = 17,21^\circ$ .

Justifier qu'on utilise pour cela l'ensemble des valeurs simulées de  $a$  et de  $b$ .

Écrire sous python une fonction `Mesure_x_inconnu(y0, sigma, a_MC, b_MC)` qui renvoie l'abscisse inconnue et son incertitude-type à partir des deux tableaux `a_MC` et `b_MC` ainsi que de l'incertitude-type sur l'ordonnée mesurée.

Appliquer cette fonction au cas du réseau. On rappelle que l'incertitude-type est prise constante, soit ici la plus grande des valeurs obtenues pour  $u(Y)$  avec  $Y = \sin i'$ .