

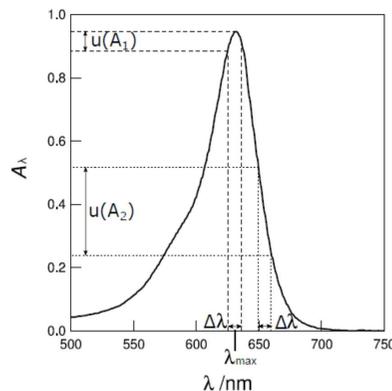
FICHE 3 : RÉGRESSION LINÉAIRE

La régression linéaire est présentée dans le cas concret du dosage par étalonnage du bleu brillant. Pour plus de détails sur la spectrophotométrie UV-visible, on se reportera à la fiche 12 « Spectrophotométrie UV-visible ».

DOSAGE PAR ÉTALONNAGE DU BLEU BRILLANT

Le bleu brillant est un colorant alimentaire (E133). Afin de déterminer sa concentration c dans un sirop de Curaçao, on effectue un **dosage par étalonnage** par spectrophotométrie. Le constructeur du spectrophotomètre UV-visible utilisé indique une précision sur l'absorbance $p_A = 0,025$.

Le spectre d'absorption d'une solution aqueuse de bleu brillant (colorant alimentaire E133) de concentration $1,0 \cdot 10^{-5} \text{ mol.L}^{-1}$ est reproduit sur la figure ci-contre :



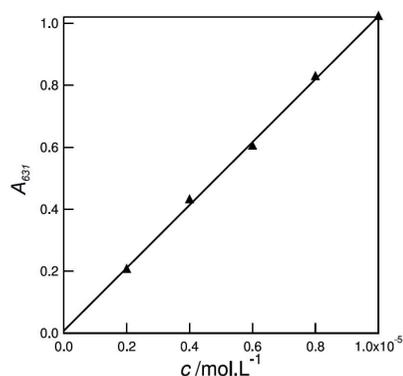
La longueur d'onde d'analyse est choisie à $\lambda_{\text{max}} = 631 \text{ nm}$.

PRÉPARATION D'UNE GAMME ÉTALON

Une **gamme étalon** (ou **échelle de teinte**) est réalisée : à partir d'une solution mère de bleu brillant à $c_0 = 1,0 \cdot 10^{-5} \text{ mol.L}^{-1}$ (avec une incertitude-type relative de 0,1 %), des solutions filles sont préparées par dilution en utilisant de la verrerie de précision (pipettes jaugées, fioles jaugées). On obtient une série de solutions de bleu brillant de concentrations c_i connues. L'absorbance de chaque solution est mesurée dans une cuve en plastique de 1 cm de long.

Les valeurs obtenues sont reportées en fonction de la concentration sur la figure ci-contre :

Afin d'avoir une meilleure précision, il est nécessaire que la gamme de concentration choisie pour faire l'échelle de teinte encadre la valeur de la concentration à déterminer. Il convient donc de mesurer l'absorbance de l'espèce inconnue au préalable et de faire l'échelle de teinte en conséquence.



VÉRIFICATION DE LA VALIDITÉ DE LA LOI DE BEER-LAMBERT AVEC UN TABLEUR

On peut réaliser une **régression linéaire** de la forme $A_{631} = a \cdot c$ à l'aide d'un tableur (type Excel, LibreOffice ou Regressi). On obtient :

$$\text{coefficient directeur : } a = 1,03 \cdot 10^5 \text{ L} \cdot \text{mol}^{-1}$$

Les points expérimentaux sont **bien répartis de part et d'autre de la droite de régression**. Cela confirme la validité de la loi de Beer-Lambert avec un coefficient d'absorption molaire $\epsilon_{631} = 1,03 \cdot 10^5 \text{ L} \cdot \text{mol}^{-1} \cdot \text{cm}^{-1}$.

VÉRIFICATION DE LA VALIDITÉ DE LA LOI DE BEER-LAMBERT AVEC PYTHON

Les bibliothèques **Numpy** et **Matplotlib.pyplot** sont importées pour les calculs numériques sur les tableaux et pour les tracés des graphes.

```
# Importation des bibliothèques
import numpy as np
import matplotlib.pyplot as plt
```

Les données numériques $X = \{x_i\}$ et $Y = \{y_i\}$ sont placées dans des listes ou des tableaux Numpy. L'incertitude-type sur les grandeurs peut être stockée dans une liste si elle change pour chaque point, ou sous forme d'un nombre unique si elle a toujours la même valeur.

```
# Données du problème
c = np.array([2e-6, 4e-6, 6e-6, 8e-6, 1e-5])
A = np.array([0.2041, 0.4257, 0.6022, 0.8267, 1.0206])
p_A = 0.025 # précision du spectrophotomètre
u_A = p_A/np.sqrt(3) # incertitude-type sur A (nombre)
u_c = 0.1/100*c # incertitudes-types sur c (liste)
```

La fonction `polyfit` de Numpy $Y = aX + b$ entre les tableaux de données X et Y . Elle renvoie un tableau de deux éléments : le **coefficient directeur** a et l'**ordonnée à l'origine** b de la droite de régression.

La fonction `polyfit` réalise la régression linéaire de la forme est plus générale : elle permet de réaliser des régressions polynomiales. Le nombre 1 entré comme dernier argument indique que le polynôme choisi est de plus haut degré 1, ce qui correspond à une droite.

```
# Régression linéaire
RL = np.polyfit(c, A, 1)
a = RL[0] # coefficient directeur
b = RL[1] # ordonnée à l'origine
print("coefficient directeur : ", a, "L/mol/cm")
print("ordonnée à l'origine : ", b)
```

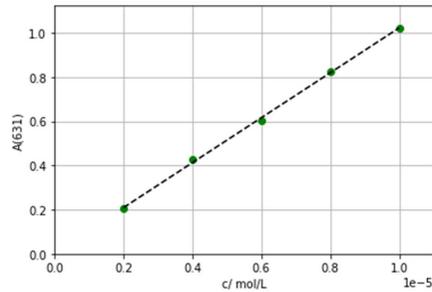
La compilation du programme renvoie :

```
coefficient directeur : 101699.99999999997 L/mol/cm
ordonnée à l'origine : 0.00566000000000000756
```

La **droite de régression** peut être superposée au graphe des données à l'aide des valeurs de a et b .

```
plt.plot(c, A, 'go') # données
```

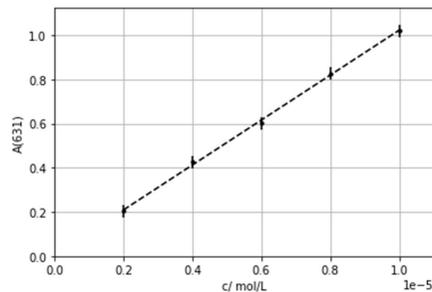
```
plt.plot(c, a*c+b,'k--') # droite de régression
plt.xlabel('c/ mol/L'), plt.ylabel('A(631)') # axes
plt.xlim(0, 1.1*np.max(c)) # échelle horizontale
plt.ylim(0,1.1*np.max(A)) # échelle verticale
plt.grid() # fait apparaître la grille
plt.show()
```



Des **barres d'erreur** sur c et A peuvent être ajoutées avec la fonction `errorbar`. Elles sont centrées sur la valeur mesurée et ont une demi-largeur (ou demi-hauteur) valant **deux fois l'incertitude-type**. Le modèle est validé si, pour chaque point, la droite de régression traverse une ellipse dont les barres d'erreur sont les axes.

Dans le code, la forme des points doit parfois être adaptée pour améliorer la lisibilité.

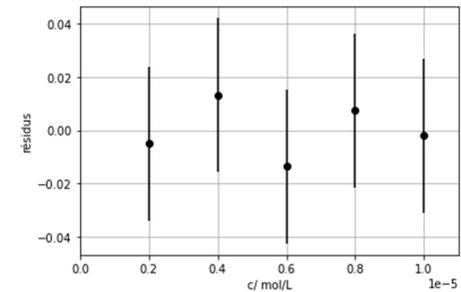
```
plt.errorbar(c, A, xerr = 2*u_c, yerr = 2*u_A, fmt ='k.')
plt.plot(c, a*c+b,'k--') # droite de régression
plt.xlabel('c/ mol/L'), plt.ylabel('A(631)') # axes
plt.xlim(0, 1.1*np.max(c)) # échelle horizontale
plt.ylim(0,1.1*np.max(A)) # échelle verticale
plt.grid() # fait apparaître la grille
plt.show()
```



Il est possible de tracer les **résidus** pour chaque point. Ils sont définis comme l'écart entre la **valeur mesurée** et la **valeur de référence**. Cette dernière est calculée à l'aide des paramètres a et b issus de la régression linéaire. On peut ajouter des barres d'erreur verticales dont la hauteur vaut deux fois l'incertitude-type. Le modèle est validé si, pour chaque point, la barre d'erreur **coupe l'axe des abscisses**.

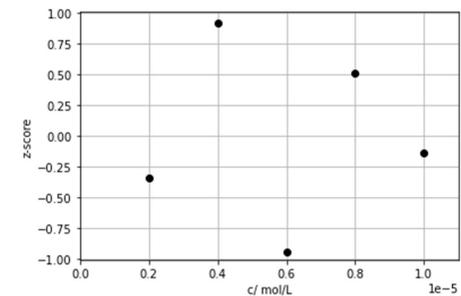
```
r = A-(a*c+b) # calcul des résidus
plt.errorbar(c, r, yerr = 2*u_A, fmt ='ko')
```

```
plt.xlabel('c/ mol/L'), plt.ylabel('résidus')
plt.xlim(0, 1.1*np.max(c)) # échelle horizontale
plt.grid() # fait apparaître la grille
plt.show()
```



Enfin, il est possible de tracer pour chaque point le **z-score** (voir définition dans la fiche 3 « Mesures et incertitudes »). Le modèle est validé si chaque point est **compris entre -2 et 2**.

```
z = (A-(a*c+b))/u_A # calcul des z-scores
plt.plot(c, z, 'ko')
plt.xlabel('c/ mol/L'), plt.ylabel('z-score') # axes
plt.xlim(0, 1.1*np.max(c)) # échelle horizontale
plt.ylim(-1.1*np.max(z),1.1*np.max(z)) # échelle verticale
plt.grid() # fait apparaître la grille
plt.show()
```



Les valeurs de **z-scores** obtenues sont **toutes comprises entre -2 et 2** et **aléatoirement réparties de chaque côté de 0**. Cela confirme qu'il existe une relation affine entre A_{631} et c .

Les **incertitudes-types** sur a et b peuvent être déterminées par la **méthode de Monte-Carlo**. Pour cela, on simule pour chaque ensemble de mesures des valeurs pour les $\{x_i\}$ et les $\{y_i\}$ en tenant compte des incertitudes-types. Puis, **une régression linéaire est réalisée pour chaque simulation** et les valeurs obtenues du coefficient directeur a et de l'ordonnée à l'origine b sont stockées dans des listes. En calculant l'**écart-type expérimental** sur ces listes, on obtient $u(a)$ et $u(b)$.

```
# Nombre de simulations
N = 10000
# simulation des distributions de données
```

```

a_sim, b_sim = [], []
for i in range(N):
    A_sim = np.random.normal(A,u_A) # simule A
    c_sim = np.random.normal(c,u_c) # simule c
    RL = np.polyfit(c_sim, A_sim, 1) # regression linéaire
    a_sim.append(RL[0]) # stockage de a simulé
    b_sim.append(RL[1]) # stockage de b simulé

# Calcul des incertitudes-types
u_a = np.std(a_sim, ddof = 1)
u_b = np.std(b_sim, ddof = 1)
print("incertitude-type sur a : u(a) = ",u_a,"L/mol")
print("incertitude-type sur b : u(b) = ",u_b)

```

La compilation du programme renvoie :

```

incertitude-type sur a : u(a) = 2305.4313359025373 L/mol
incertitude-type sur b : u(b) = 0.015331859911522638

```

On peut donc écrire : $a = (1,017 \pm 0,023) \cdot 10^5 \text{ L} \cdot \text{mol}^{-1}$ et $b = (6 \pm 15) \cdot 10^{-3}$

Dans l'hypothèse d'un modèle linéaire entre A et c , la valeur de référence de b est 0. Le calcul du z -score sur la valeur mesurée de b est ici :

$$z = |6 - 0|/15 = 0,4 < 2$$

La loi de Beer-Lambert est donc bien valide avec un coefficient d'absorption molaire $\varepsilon_{631} = (1,017 \pm 0,023) \cdot 10^5 \text{ L} \cdot \text{mol}^{-1} \cdot \text{cm}^{-1}$.

DÉTERMINATION DE LA CONCENTRATION DU BLEU BRILLANT

L'absorbance du sirop de Curaçao est trop élevée (supérieure à 1,5) pour utiliser la courbe d'étalonnage réalisée. Le sirop est donc dilué 10 fois. Pour cela, on prélève un volume $V_{\text{pipette}} = 10 \text{ mL}$ de sirop de Curaçao à l'aide d'une pipette jaugée de précision 0,03 mL qui est versé dans une fiole de volume $V_{\text{fiole}} = 100 \text{ mL}$ de précision 0,1 mL. La solution diluée présente une absorbance de $A_{631} = 0,7803$.

En prenant en compte la dilution, la loi de Beer-Lambert donne :

$$c = \frac{V_{\text{fiole}}}{V_{\text{pipette}}} \frac{A_{631}}{\varepsilon_{631} l} = \frac{100}{10} \frac{0,7803}{1,017 \cdot 10^5 \times 1} = 7,673 \cdot 10^{-5} \text{ mol} \cdot \text{L}^{-1}$$

X	x	Matériel/instrument	p(x)	u(x)
$V_{\text{pipette}} / \text{mL}$	10	Pipette	0,03	
$V_{\text{fiole}} / \text{mL}$	100	Fiole	0,1	
A_{631}	0,7803	Spectrophotomètre	0,025	
$\varepsilon_{631} / \text{L} \cdot \text{mol}^{-1} \cdot \text{cm}^{-1}$	$1,017 \cdot 10^5$	Régression linéaire		$2,3 \cdot 10^3$
l / cm	1	Cuve		0

À l'aide d'une simulation de type Monte-Carlo (faite avec Python ou avec le logiciel Gum_MC), on détermine que $u(c) = 2,3 \cdot 10^{-6} \text{ mol} \cdot \text{L}^{-1}$.

Le code Python est donné ci-dessous :

```

# Importation des bibliothèques
import numpy as np

# Nombre de simulations
N = 1000000

# Données du problème

Vfiole = 100 # mL (volume de la fiole jaugée)
p_fiole = 0.1 # mL (précision de la fiole jaugée)
Vpipette = 10 # mL (volume de la pipette jaugée)
p_pipette = 0.03 # mL (précision de la pipette jaugée)
A631 = 0.7803 # (absorbance de la solution diluée)
p_A631 = 0.025 # (précision de la mesure d'absorbance)
eps631 = 1.017e5 # L/mol/cm (coeff d'absorption molaire)
u_eps631 = 2300 # L/mol/cm (incertitude-type sur le coeff d'absorption molaire)
l = 1 # cm (largeur de la cuve)

# Simulation des distributions de données

Vfiole_sim = Vfiole + np.random.uniform(-p_fiole,p_fiole,N)
Vpipette_sim = Vpipette + np.random.uniform(-p_pipette,p_pipette,N)
A631_sim = A631 + np.random.uniform(-p_A631,p_A631,N)
eps631_sim = np.random.normal(eps631,u_eps631,N)

c_sim = Vfiole_sim*A631_sim/(Vpipette_sim*eps631_sim*1)

c = np.average(c_sim)
u_c = np.std(c_sim, ddof=1)

print("cmoyen = ",c,"mol/L")
print("u(c) = ",u_c,"mol/L")

```

La compilation du programme renvoie :

```

cmoyen = 7.676683581993027e-05 mol/L
u(c) = 2.2496055017216335e-06 mol/L

```

D'où : $c = (7,67 \pm 0,22) \cdot 10^{-5} \text{ mol} \cdot \text{L}^{-1}$

Les codes Python sont disponibles sur le site de la classe (PC → chimie → Informations et documents divers de chimie) :

<http://pc.bginette.com/LerouxR/Divers/index.php>

Identifiant : *ginette* ; mot de passe : *toto*

LU DANS LES RAPPORTS DU JURY...

« La maîtrise des logiciels par les candidat·e·s est trop lacunaire et cela les pénalise. Les étapes suivantes posent souvent des difficultés : choix des axes, superposition de courbes issues d'expériences différentes, qualité des régressions ou possibilité de ne faire des régressions que sur un sous-ensemble de points, méthode des tangentes, de la dérivée, affichage simultané de la courbe et des valeurs numériques. (ENS2018 à 2023) »