TP n°2 : Focométrie

▷ Dans ce TP, nous allons étudier deux méthodes expérimentales permettant de déterminer la distance focale image de lentilles convergentes.

Ce que vous devez savoir faire

▷ Estimer l'ordre de grandeur d'une distance focale de lentille par auto-collimation ;

▷ Déterminer de façon rigoureuse une distance focale à l'aide de la méthode de Bessel ;

▷ Estimer une incertitude de mesure et sa propagation dans une formule donnée (incertitude de type B) ;

Matériel :

▷ Banc d'optique et accessoires ;

▷ Objet "F" ;

▷ Lumière blanche ;

▷ Lentilles convergentes ;

▷ Miroir plan ;

▷ Écran avec papier millimétré.

I. Détermination de la distance focale de l'objectif par auto-collimation

▷ Pour cette première partie, se munir de la lentille convergente donnée par le professeur. Le but est de vérifier par une mesure très rapide la valeur de sa vergence. Pour cela, on utilise l'autocollimation (cf. TD O2, EX 3 pour la théorie).

Protocole : Autocollimation

1. Positionner la lentille en face d'un objet bien éclairé.
2. Tenir ensuite à la main un miroir collé immédiatement derrière la lentille (le miroir ne doit pas forcément être très droit).
3. Déplacer l'ensemble (lentille+miroir) jusqu'à ce que l'image de l'objet se forme dans le même plan que l'objet.
4. La distance objet lentille est alors égale à la distance focale de la lentille. Ne pas hésiter à incliner légèrement le miroir pour que l'image soit bien visible.

Remarques

— La méthode d'autocollimation ne s'utilise que pour les lentilles convergentes.

— Pour les lentilles divergentes, on peut utiliser le principe d'accolement des lentilles, avec une lentille convergente de grande vergence pour que la somme des deux vergences soit positive et que l'on puisse utiliser la méthode d'autocollimation.

Q 1. Réaliser le protocole de l'autocollimation pour déterminer la distance focale de la lentille convergente donnée par le professeur. Faire un traitement des incertitudes et comparer avec la valeur donnée par le professeur à l'aide notamment d'un écart normalisé.

II. Méthode de Bessel

▷ Dans le cas où la distance D objet-écran fixée est telle que $D > 4f'$, comme vu en cours, il y a deux positions de la lentille convergente, distantes de d, pour lesquelles on observe une image nette de l'objet sur l'écran.

Q 2. Faire un schéma de la situation en représentant D, d.

Q 3. Montrer que la distance focale image de la lentille a pour expression :

$$f' = \frac{D^2 - d^2}{4D}$$

Q 4. Montrer que l'on peut tracer la droite d'équation $Y = aX + b$ avec $X = \dfrac{1}{D}$, $Y = \dfrac{d^2}{D^2}$ et a et b à exprimer en fonction de f'. C'est ce que l'on va faire par la suite sur Python.

Q 5. Faire une mesure de D et d pour la lentille donnée par le professeur, en prenant en compte les incertitudes $u(d)$ et $u(D)$. Quelle incertitude domine ?

APPEL PROFESSEUR.

Q 6. Montrer que :

$$u(X) = \frac{u(D)}{D^2} \quad \text{et} \quad u(Y) = \frac{2D^2}{d^2}\sqrt{\left(\frac{u(d)}{d}\right)^2 + \left(\frac{u(D)}{D}\right)^2}$$

Q 7. Prendre une dizaine de mesures de D et d, puis tracer sur Python la droite modélisée par l'équation $Y = aX + b$ avec les barres d'erreur. Se servir de l'annexe pour mettre en place votre code. Le modèle paraît-il satisfaisant ?

APPEL PROFESSEUR.

Q 8. Prendre un des couples (D, d) de votre mesure et appliquer la méthode de Monte-Carlo afin de déterminer f' et son incertitude $u(f')$.

Q 9. Comparer la valeur avec la valeur donnée par le constructeur, en utilisant l'écart normalisé.

Annexe

▷ Cette annexe recense les seules fonctions dont vous avez besoin pour ce TP. Les deux bibliothèques se chargent une fois pour toutes en début de fichier :

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

Les tableaux numpy

▷ Un tableau `numpy` ressemble à une liste, mais les opérations s'y appliquent *terme à terme*, ce qui évite toute boucle. Par exemple :

— `L**2` crée un nouveau tableau élevant au carré puis stockant chacune des valeurs de `L`.

— `1/L` fonctionne de façon similaire : il crée un tableau en stockant cette fois-ci l'inverse de toutes les valeurs de `L`.

— etc. on imagine bien la puissance de cet outil !

Voici la syntaxe de cet outil :

```
L = np.array([valeur 1, valeur 2, ... , valeur n])
```

Remarques

— Les valeurs sont entre **crochets**, séparées par des virgules : `np.array(1, 2, 3)` provoque une erreur.

— Écrivez toujours vos mesures avec un point décimal (`80.0` et non `80`) pour éviter tout problème de division entière.

La régression linéaire

```
a, b = np.polyfit(X, Y, 1)
```

▷ `np.polyfit(X, Y, n)` cherche le polynôme de degré `n` qui passe au plus près des points, au sens des moindres carrés. Avec `n = 1`, il s'agit d'une droite et la fonction renvoie **deux nombres, dans cet ordre** : la pente `a`, puis l'ordonnée à l'origine `b`, de sorte que la droite ajustée a pour équation $Y = a\,X + b$.

Le tracé

```
plt.errorbar(X, Y, xerr=uX, yerr=uY, fmt='o', capsize=3, label='mesures')
plt.plot(X, a*X + b, label='regression')

plt.xlabel("Titre de l'axe des abscisses")
plt.ylabel("Titre de l'axe des ordonnées")
plt.title("Titre du graphique")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

▷ Le détail de chaque commande :

- ▷ `plt.errorbar(X, Y, ...)` place les points **avec leurs barres d'erreur**. `xerr` et `yerr` reçoivent les demi-longueurs des barres : chaque barre s'étend donc de $-u$ à $+u$ autour du point, et sa longueur totale vaut $2u$. L'option `fmt='o'` demande des points ronds **non reliés** entre eux : sans elle, les points seraient joints par une ligne brisée. `capsize=3` ajoute les petits traits qui ferment les barres.
- ▷ `plt.plot(X, a*X + b)` trace la droite ajustée. `a*X + b` calcule l'ordonnée de la droite pour chaque abscisse du tableau `X`.
- ▷ `plt.xlabel` et `plt.ylabel` nomment les axes, `plt.title` donne un titre à la figure. **Une figure sans axes légendés n'est pas évaluable.**

▷ `plt.legend()` affiche la légende : elle reprend les textes passés en `label` dans les commandes précédentes, et n'affiche rien s'il n'y en a aucun.

▷ `plt.grid(alpha=0.3)` ajoute une grille discrète, utile pour lire l'ordonnée à l'origine.

▷ `plt.show()` affiche la fenêtre graphique. À placer **en dernier**, une fois tous les tracés effectués.

L'incertitude par simulation (méthode de Monte-Carlo)

▷ Le principe est de *rejouer* la mesure un grand nombre de fois : on tire au hasard des valeurs de A (avec A une grandeur mesurée qui nous intéresse dans une expérience, mais on peut tout à fait en avoir plusieurs) compatibles avec les incertitudes, on recalcule la grandeur voulue, et on regarde à quel point les résultats se dispersent.

```
N = 10000                              # nombre de tirages

Asim = np.random.normal(A, uA, N)    # tirage aleatoire autour de la valeur A
Asim.mean()
Asim.std(ddof=1)
```

▷ `np.random.normal(valeur, incertitude, N)` renvoie `N` tirages aléatoires répartis autour de `valeur`, avec une dispersion égale à `incertitude`.

▷ `Asim.mean()` donne la **moyenne des valeurs du tableau** `Asim` : c'est le **résultat de la mesure**.

▷ `Asim.std(ddof=1)` donne l' **écart-type expérimental** du tableau `Asim`, celui du cours (division par $n-1$) : c'est l'**incertitude-type** cherchée.

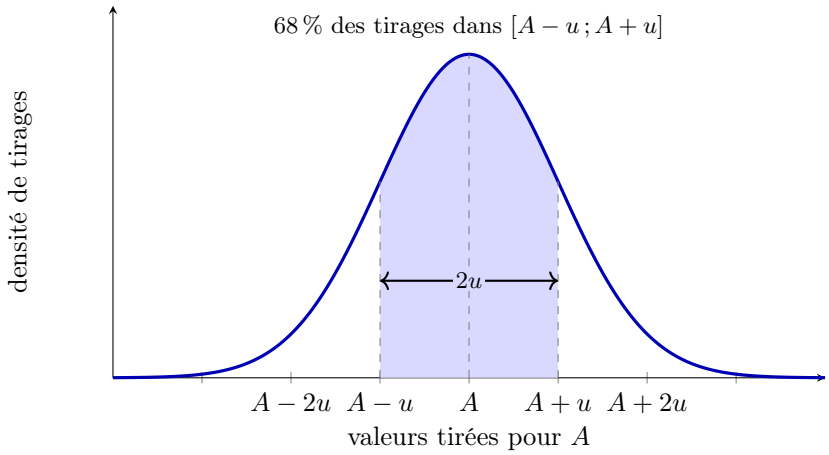

FIGURE 1 – Distribution des N valeurs produites par `np.random.normal(A, uA, N)` : elles se concentrent autour de A, avec une dispersion fixée par $u = u(A)$.

▷ Chaque tirage représente une valeur de A que l'on aurait pu mesurer, compte tenu de l'incertitude : les valeurs proches de A sortent souvent, les valeurs éloignées rarement. L'intervalle de largeur $2u$ (avec $u = u(A)$) centré sur A, en bleu sur la figure, contient environ 68 % des tirages : c'est en ce sens que $u(A)$ quantifie la dispersion de la mesure.

Remarques

Prenez au moins $N = 10\,000$ tirages pour que le résultat soit stable. Relancez deux fois votre programme : les deux valeurs d'incertitude doivent coïncider sur les deux premiers chiffres significatifs.