TD 2

Traitement des données

L'objectif de ce chapitre est d'examiner les aspects statistiques qui apparaissent lors d'une série de mesures d'une grandeur physique et comment cela impacte les grandeurs dérivées que l'on veut déduire, ainsi que la validation des modèles interprétatifs des expériences réalisées. La première section traite de l'aspect purement statistique d'un ensemble de données réelles ou simulées, la seconde de la propagation des erreurs dans le cas de grandeurs calculées et enfin la dernière section traite de la comparaison d'un jeu de données avec un loi physique, linéaire dans une première étape puis un exemple non linéaire dans une seconde et dernière étape.

A Statistique d'un ensemble de mesures

Considérons une grandeur physique X dont on veut réaliser la mesure. La répétition de ces mesures conduit à un ensemble de résultats $(x_1, x_2, ..., x_N)$ qu'on peut le plus souvent réduire à la donnée de deux nombres : la valeur moyenne $\overline{X}$ (souvent la valeur la plus probable) et la dispersion de ces valeurs ou "*incertitude*" $u(X)$. Où

$$\overline{X} = \frac{\sum_{i=1}^{N} x_i}{N}$$

et

$$u(X) = \sqrt{\frac{\sum_{i=1}^{N} \left(x_i - \overline{X}\right)}{N-1}}$$

On note alors le résultat sous la forme :

$$X = \overline{X} \pm u(X)$$

En réalité l'ensemble des mesures obéit à une répartition caractérisée par une loi de probabilité $p(x)$ (le plus souvent une loi dite *normale* ou gaussienne [1])

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

où $\overline{X} \to \mu$ et $u(X) \to \sigma$ lorsque $N \to \infty$.

On se propose dans cette première partie d'examiner sur un cas réel la répartition d'un ensemble de mesures d'une grandeur physique (ici une durée) à l'aide de ce qu'on appelle un *histogramme* et ensuite de comparer cette répartition avec une loi de probabilité gaussienne centrée sur la valeur moyenne et la largeur de celle-ci évaluée à partir de l'écart-type de ce jeu de données.

L'expérience que l'on examinera est celle de la chute d'une bille roulant le long d'un plan incliné où l'on a mesuré la durée de chute en suivant 2 protocoles :

1. mais pas toujours...

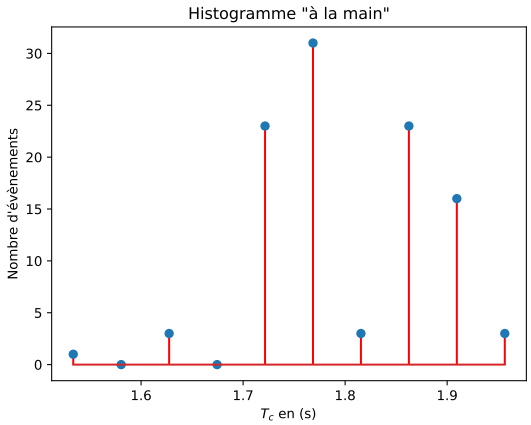
- dans le premier, on utilise un chronomètre qui est mis en marche au moment du lâché, puis est arrêté lorsque la bille parvient à l'extrémité de la rampe. Le chronomètre fournit des résultats au centième de seconde mais comme on le verra sur le jeu de données récupéré, c'est le temps de réponse de l'expérimentateur qui va *in fine* déterminer la précision de la mesure.
- dans le second, on utilise un système de fourches optiques pour déclencher puis arrêter un "timer", lors du passage de la bille dans les fourches. Les détecteurs optiques étant reliés à un microcontrôleur Arduino, le "timer" de celui-ci permet d'avoir des résultats à la microseconde. Néanmoins, on verra que là encore c'est l'expérimentateur qui, par les conditions initiales du lâcher, impose des fluctuations aléatoires qui détermineront la précision de cette durée.

A.1 1ère expérience

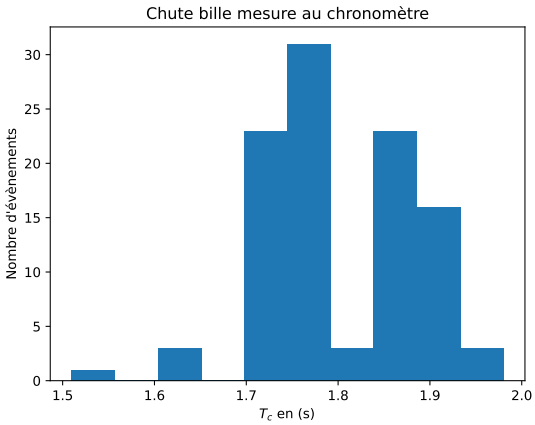
Exercice n° 1 *Mesure au chrono de la durée de chute de la bille.*

1. Récupérer le fichier de données `data_acquis_chrono.csv` sur l'ENT.
2. Récupérer les durées de chute dans un tableau (noté `dt` par exemple) à l'aide la fonction `read_csv` de `pandas`.
3. Tracé de l'histogramme :
   (a) Dans une première étape on va construire l'histogramme "à la main" en comptant le nombre d'évènements du tableau qui se situent dans un intervalle donné. Le principe est le suivant : on découpe l'intervalle des valeurs possibles en N boîtes de même dimension et on compte le nombre d'éléments du tableau de données contenus dans chaque sous-intervalle et que l'on stocke au final dans un tableau appelé histogramme.
      - Définir l'intervalle de travail en extrayant les valeurs minimale et maximale du tableau `dt`. *Indications* : les fonctions `T.min()` et `T.max()` d'un array `T` permettent de récupérer facilement ces valeurs.
      - On notera `nbin` le nombre de "boîtes" qui subdivisent l'intervalle de travail (prendre typiquement ici 10 boîtes) et construire un tableau `vbin` contenant la position centrale de chacune de ces boîtes dans l'hypothèse où toutes les boîtes ont la même largeur.
      - Construire alors un tableau `hist_v` de même dimension que `vbin` dont chaque élément contient le nombre d'éléments du tableau données initial `dt` dont les valeurs sont comprises dans l'intervalle.$
      - Tracez enfin l'histogramme `hist_v` en fonction du tableau des boîtes `vbin` à l'aide des fonction `plot()` ou `stem()` de `matplotlib`.
   (b) Dans une seconde étape, on peut obtenir plus rapidement ce résultat à l'aide de la fonction `hist` de `matplotlib`. Comparez (cf. figure 2.1)
4. Calculer la valeur moyenne et l'écart type de ce tableau de valeurs.
   Indication : les fonctions `mean` et `std` de `numpy` permettent un calcul rapide de ces valeurs. Comparez avec un calcul "à la main" [2].
5. Tracer sur une même figure l'histogramme des données et la distribution normale associée.
   Indications :
   - On choisira de normaliser l'histogramme en représentant la densité fréquentielle des évènements plutôt que le nombre dans chaque intervalle (option `density=True` de la fonction `hist()`) afin de comparer directement avec la loi gaussienne qui est normalisée.
   - On pourra utiliser la fonction `norm` de la bibliothèque `scipy.stat` afin de générer directement la courbe de $p(x)$ dans l'intervalle demandé.

2. En python quand même ! mais à l'aide d'une boucle for.



(a) *Réalisé "à la main" (fonction* `stem()` *de* `matplotlib`*)*



(b) *Réalisé avec la fonction* `hist()` *de* `matplotlib`

FIGURE 2.1 – *Histogrammes*

A.2 2ème expérience

Exercice n° 2 *mesure avec des barrières optiques*

On reprend le même traitement que précédemment avec le fichier correspond.

1. Récupérer le fichier de données `data_acquis_arduino.csv` sur l'ENT.
2. Récupérer les durées de chute dans un tableau, puis tracer l'histogramme de ces données.
3. Calculer la valeur moyenne et l'écart type de ce tableau de valeurs et tracer ensuite sur une même figure l'histogramme des données et la distribution normale associée.
4. Ecrire le résultat sous la forme $X = \overline{X} \pm u(X)$ et comparer avec le résultat de la mesure au chronomètre.

A.3 Modélisation Monte Carlo

Exercice n° 3 *Simulations*

On cherche à présent à simuler numériquement la dispersion des résultats suivant une loi de distribution gaussienne.

1. Effectuer une tirage aléatoire à l'aide de la fonction `normal()` de `numpy.random` qui renvoie un tableau (que l'on notera `Xmc`) de N valeurs centrées sur une valeur moyenne μ et un écart type σ. On prendra $\mu = 5$ et $\sigma = 0,2$.
2. Tracer les histogrammes de ce tableau pour $N = 20, 100, 5000$. Commentaires ?
3. Comparer $\overline{X}$ et $u(X)$ pour ces différentes valeurs de N. Que peut-on en conclure ?

B Propagation des erreurs

Le plus souvent en physique on mesure plusieurs variables pour en déduire une autre. Par exemple, en mesurant la tension aux bornes d'un dipôle et l'intensité du courant qui le parcourt, on peut déduire sa résistance en supposant que celui-ci obéit à la loi d'Ohm, ou bien en mesurant le volume d'un gaz et sa pression, on peut en déduire sa température si on suppose que celui-ci obéit à l'équation d'état du gaz parfait. Le problème est donc dans ces situations d'estimer l'incertitude sur la variable calculée qui est une mesure de la dispersion des valeurs possibles de cette grandeur en fonction des incertitudes des grandeurs mesurées. On parle alors de propagation des erreurs.

B.1 Cas standard

Exercice n° 4 *Somme et produit de variables aléatoires*

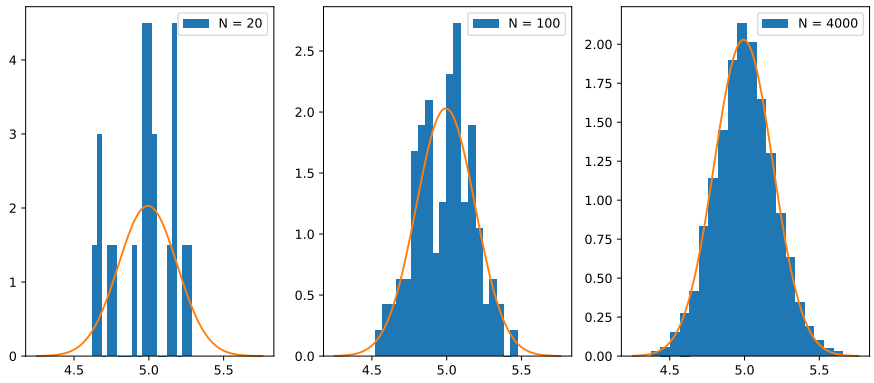


FIGURE 2.2 – *Comparaison d'histogrammes d'une même distribution pour un nombre N de tirages différents*

Soit 2 variables aléatoires X et Y de valeur moyenne et d'écart type respectivement $\mu_X = 2$, $\sigma_X = 0,05$ et $\mu_Y = 3$, $\sigma_Y = 0,1$

1. Tirer aléatoirement 2 tableaux `Xmc` et `Ymc` à l'aide de la fonction `normal()` de `numpy.random` en prenant un nombre de points $N = 500$. Vérifier comme précédemment que l'on a bien $\overline{X} \simeq \mu_X$ et $u(X) \simeq \sigma_X$ (et de même pour la variable Y).
2. Calculer le tableau somme : `Zmc = Xmc+Ymc`. En déduire la valeur moyenne $\overline{Z}$ et l'incertitude $u(Z)$ et comparer ce résultat avec la "formule" du lycée
$$u(Z) = \sqrt{u(X)^2 + u(Y)^2}$$
3. Calculer le tableau produit : `Zmc = Xmc*Ymc`. En déduire la valeur moyenne $\overline{Z}$ et l'incertitude $u(Z)$ et comparer ce résultat avec la "formule" du lycée
$$u(Z) = \overline{Z}\sqrt{\left(\frac{u(X)}{\overline{X}}\right)^2 + \left(\frac{u(Y)}{\overline{Y}}\right)^2}$$

B.2 Application à une expression non triviale

Dans le cas général d'une relation de type $Z = f(X,Y)$ la linéarisation de cette expression autour des valeurs moyennes de X et Y et l'hypothèse de loi normale des fluctuations autour de ces valeurs conduit à l'expression
$$u(Z) = \sqrt{\left(\frac{\partial f}{\partial X}u(X)\right)^2 + \left(\frac{\partial f}{\partial Y}u(Y)\right)^2}$$
où $\partial f/\partial X$ et $\partial f/\partial Y$ sont les dérivées partielles de la fonction f de X et Y. Ce calcul étant en général assez compliqué, on se propose d'écrire un programme qui simule la distribution des variables aléatoires X et Y et par cet échantillonnage d'en déduire les valeurs de $\overline{Z}$ et $u(Z)$. On travaillera ici dans le cas particulier de la mesure de l'indice d'un milieu en forme de prisme par la méthode du minimum de déviation.

Exercice n° 5 *Indice de réfraction*

On considère un prisme fait d'un matériau transparent d'indice de réfraction n et d'angle au sommet A. On envoie sur une des face du prisme un rayon lumineux monochromatique qui ressort par l'autre face en faisant un angle de déviation D par rapport à la direction initiale. En appliquant les lois de Snell-Descartes de la réfraction et quelques règles de géométrie, on peut montrer [3] que pour un angle de déviation minimale D_m l'indice n est relié à cet angle et à l'angle A par la relation
$$n = \frac{\sin\left(\frac{A+D_m}{2}\right)}{\sin\left(\frac{A}{2}\right)}$$

3. Faites-le !

1. Tirer aléatoirement 2 tableaux `Amc` et `Dmc` à l'aide de la fonction `normal()` de `numpy.random` en prenant un nombre de points $N = 500$ avec $\mu_A = 60°$, $\sigma_A = 2'$, $\mu_{D_m} = 53°35'$ et $\sigma_{D_m} = 2'$.
2. Calculer un tableau de valeur de l'indice à partir de ces 2 tableaux d'angle et en déduire $\overline{n}$ et $u(n)$. Attention, on prendra soin de convertir les degrés et minutes d'angle en radian pour calculer les sinus.
3. Pour les plus courageux, comparer ce résultat avec la formule rappelée plus haut.

C Ajustements

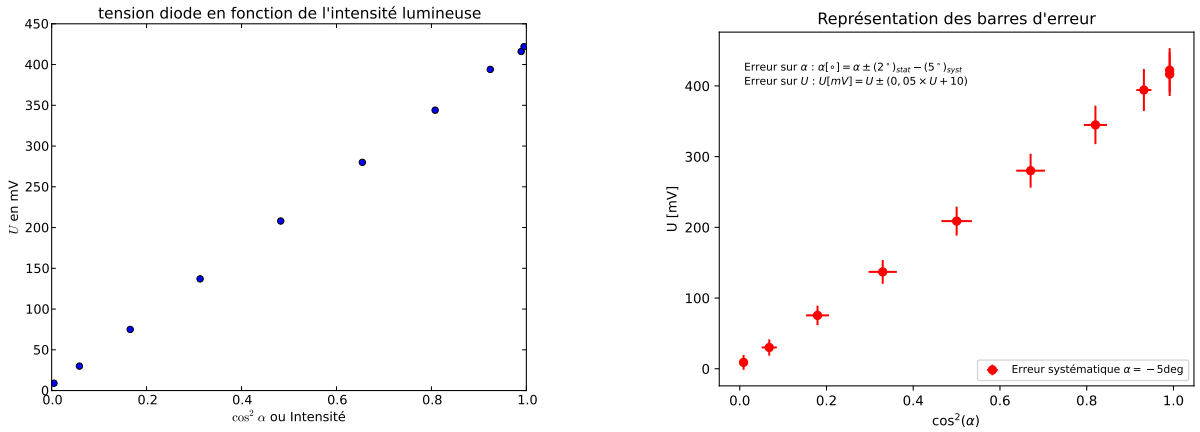
La démarche expérimentale initiée par Galilée il y a près de 400 ans consiste pour l'essentiel à confronter des données issues d'une expérience avec un modèle mathématique supposé décrire le phénomène physique étudié. On se propose ici d'étudier 2 expériences :

- la première consiste à vérifier qu'une lumière polarisée rectilignement traversant un analyseur faisant un angle α avec la polarisation initiale produit une intensité lumineuse I proportionnelle au cosinus de l'angle α. C'est la loi de Malus
$$I = I_0 \cos^2 \alpha$$
- avec la seconde, on exploite l'évolution temporelle de la position angulaire d'un pendule amorti afin d'en extraire la période et la durée d'amortissement.

C.1 Préparation des données

Lors de l'expérience de vérification de la loi deMalus une photodiode fournit en sortie de l'analyseur, une tension U proportionnelle à l'intensité lumineuse. Les données sont représentées dans le fichier sous la forme de deux colonnes correspondant aux angles α et aux tensions U correspondantes. Une ligne d'en-tête permet d'identifier à quelle grandeur physique se rapporte chaque colonne ainsi que les unités utilisées.

Exercice n° 6 *Loi de Malus : représentation des données*



(a) *Tracé simple* (b) *Tracé avec barres d'erreurs*

FIGURE 2.3 – *Tracé des données expérimentales du fichier* `dataMalus.csv`

1. Écrire un programme utilisant la fonction `read_csv()` de la bibliothèque `pandas` qui récupère les données du fichier `dataMalus.csv` et les stocke dans des tableaux (`array`) qu'on pourra noter `U` et `alpha` .
2. Tracer avec la fonction plot, le graphe $U = f(\cos^2 \alpha)$ (cf. figure 2.3a). On prendra soin de convertir les degrés en radians.
3. Rajouter un décalage systématique α_0 sur les angles. Combien faut-il prendre pour que les données semblent « bien alignées » ? Comment appelle-t-on ce type d'erreurs ?

4. Reprendre le tracé des données en rajoutant des barres d'erreurs de 5% sur la mesure de la tension plus une erreur de 10 mV due au calibre) à l'aide de la fonction de `matplotlib` : `errorbar()`, dont voici la syntaxe `errorbar(x, y, yer, fmt='o')` (cf. figure 2.3b).
5. Pour tenir compte de l'incertitude sur l'angle α que l'on prendra égale à 1° pour chaque angle, propagez les erreurs sur la fonction $\cos^2\alpha$ en utilisant la méthode de Monte Carlo utilisée précédemment.

C.2 Retour sur l'ajustement linéaire

On veut dans cette partie obtenir les paramètres d'une formule d'ajustement sur les données numériques (fit en anglais) ; ajustement linéaire dans un premier temps, puis non-linéaire dans une seconde étape.

Principe de la méthode des moindres carrés cette méthode consiste à déterminer la « meilleure droite » $y(x)$ passant par les points expérimentaux y_i en minimisant la distance de cette droite $Ax+B$ aux différents points mesurés y_i. Pour cela on construit la fonction

$$\chi^2(A,B) = \sum_{i=1}^{N} \frac{[y_i - (Ax_i + B)]^2}{\sigma_i^2}$$

où σ_i correspond à l'incertitude de mesure associée à la variable y_i. On posera par la suite $\sigma_i = \sigma$ constante.
On minimise cette distance en posant $d\chi^2 = 0$ qui conduit à la relation $\frac{\partial\chi^2}{\partial A} = \frac{\partial\chi^2}{\partial B} = 0$.
On obtient (faites le calcul !) le système linéaire en A et B

$$\begin{cases} A\Sigma x_i^2 + B\Sigma x_i &= \Sigma x_i y_i \\ A\Sigma x_i + BN &= \Sigma y_i \end{cases}$$

Ce qui donne

$$A = \frac{N\Sigma x_i y_i - \Sigma x_i \Sigma y_i}{\Delta} \quad \text{et} \quad B = \frac{\Sigma y_i \Sigma x_i^2 - \Sigma x_i \Sigma x_i y_i}{\Delta}$$

avec $\Delta = N\Sigma x_i^2 - (\Sigma x_i)^2$.

Exercice n° 7 *Loi de Malus : ajustement linéaire*

Programme « maison »

1. Traiter les données en appliquant les formules précédentes de la méthode des moindres carrés. Calculer A, B.
2. Re-écrire le code précédent sous la forme d'une fonction `reglin()` prenant comme argument les tableaux de données et retournant les paramètres A et B

Programme scipy

3. À l'aide de la fonction `curve_fit` de la bibliothèque `scipy.optimize` on peut traiter directement les données en définissant la fonction `fonc(x, a, b)` à ajuster. On récupère les paramètres A et B optimisés de la façon suivante :

```
def fonc(x, a, b):
    return a*x+b

popt, pcov = curve_fit(fonc, x, y)
[A, B] = popt
```

où `popt` est un vecteur qui renvoie les paramètres optimisés de la fonction et `pcov` est une matrice que nous n'utiliserons pas ici. Comparer avec les résultats de votre programme "maison" précédent.

C.3 Estimation des incertitudes sur les paramètres de l'ajustement

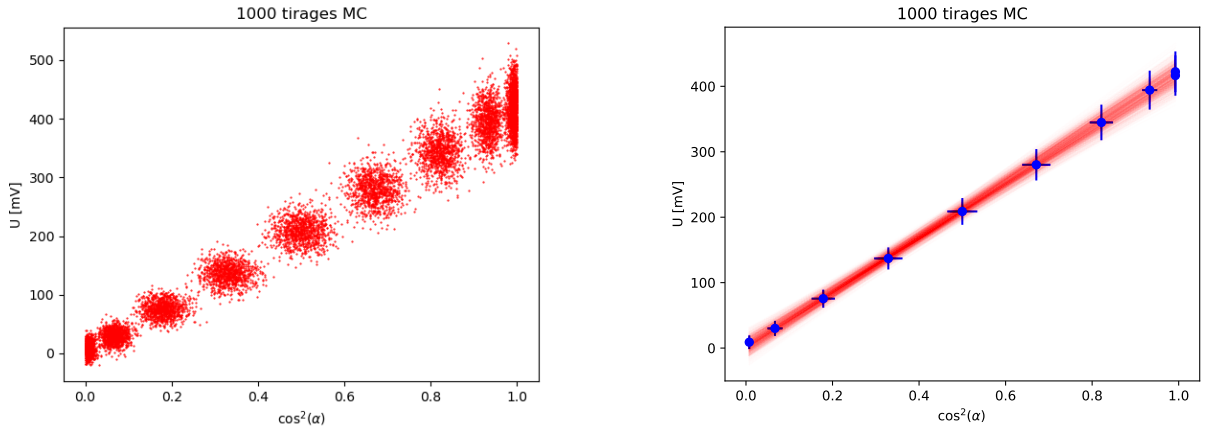
Afin d'effectuer l'ajustement du modèle nous avons utilisé les valeurs expérimentales **mais sans tenir compte des incertitudes** sur les mesures. Il est donc intéressant de se poser la question de l'influence de l'incertitude sur les mesures sur les paramètres de l'ajustement. Et donc d'**estimer l'incertitude sur les paramètres de l'ajustement**.

"Simuler" des données expérimentales Nous avons vu précédemment que l'on pouvait générer des jeux de "mesures" par méthode Monte Carlo en respectant la distribution expérimentale des valeurs. Pour estimer l'incertitude sur les paramètres de l'ajustement, la technique est la suivante :

- générer un jeu de données en tirant aléatoirement les points sur une loi normale centré sur la mesure et de largeur (écart-type) l'incertitude sur la mesure)
- ajuster ce "nouveau" jeu de données expérimentales et obtenir une nouvelle valeur de A et de B
- répéter l'opération un grand nombre de fois et stocker toutes les valeurs de A et de B obtenues dans un tableau
- Estimer la valeur moyenne de A et B et les écart-types sur A et B.

Exercice n° 8 *Loi de Malus : prise en compte des incertitudes*

1. Ecrire sous forme d'une fonction le code qui génère un nouveau lot de données
2. Effectuer 1000 appels de cette fonction et représenter les 1000 tirages sur le même graphe ($U = f\left(\cos^2\alpha\right)$)
3. Ecrire sous forme d'une fonction le code qui fait l'ajustement et retourne les paramètres A et B
4. Ecrire le code qui effectue 1000 tirages et stocke dans des tableaux les valeurs de A et de B obtenues
5. Tracer les histogrammes représentant les distributions de A et de B
6. Extraire les moyennes et écarts-type de A et B



(a) *Echantillonnage sur 1000 tirages*

(b) *Distribution des 1000 droites ajustées*

FIGURE 2.4 – *Ajustement linéaire et Monte Carlo*

C.4 Ajustement non linéaire - Pendule amorti

On considère un fichier de données relevées par un étudiant lors d'une étude du mouvement d'un pendule (fichier `Mesure_TP_pendule.csv`)

- La première colonne donne la date (en secondes) de la prise de mesure.
- La seconde colonne donne l'incertitude sur la mesure de cette date.
- La troisième colonne donne la mesure, en degrés, de l'angle du pendule avec la verticale.
- La quatrième colonne donne l'incertitude en degrés sur la mesure de cet angle.

Exercice n° 9 *Etude d'un pendule amorti*

Affichage

1. Ecrire un code qui lit les données avec pandas.
2. Extraire les quatre tableaux de valeurs (`t`, `dt`, `theta`, `dtheta`)
3. Afficher les points expérimentaux avec les barres d'erreur dans un graphe correctement légendé

Il apparaît clairement que la dépendance de θ en fonction de t n'est pas linéaire. L'écriture de l'équation différentielle du mouvement du pendule conduit à modéliser θ de la façon suivante :

$$\theta(t) = \theta_0 \cos(\omega t + \varphi) e^{-\alpha t}$$

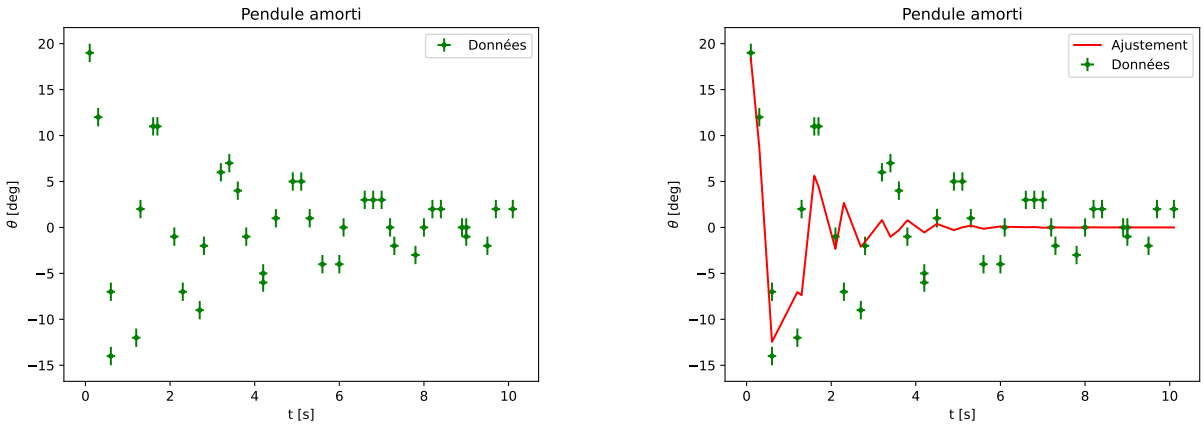
où

- θ_0 est l'angle initial en degrés (si θ est en degrés), si on suppose que le pendule est lâché sans vitesse initiale.
- ω est la pulsation (en rad.s^{-1}), avec $\omega = \frac{2\pi}{T}$ où T est la période d'oscillation du pendule.
- φ est la phase (en rad) à l'origine.
- α en s^{-1} exprime l'amortissement des oscillations avec le temps.

Premier essai de modélisation L'ajustement étant manifestement non linéaire, il va être fait appel à la fonction `curve_fit`.

4. Ecrire un code qui fait appel à `curve_fit` pour ajuster les données sur le modèle précédent
5. On tracera alors un graphe représentant les points expérimentaux avec leurs incertitudes et la courbe d'ajustement tracée avec `plot`

On constate ici deux problèmes, la courbe est moche et elle modélise très mal les données expérimentales.



(a) *Données brutes* (b) *Ajustement brut non optimisé*

FIGURE 2.5 – *Tracés bruts sinus amorti*

Obtenir une "jolie" courbe d'ajustement Pour obtenir une jolie courbe lissée, il faut utiliser plus de points.

6. Créer un tableau de temps contenant le nombre de points désirés pour obtenir une bonne représentation et tracer la courbe d'ajustement à partir de ce tableau de points.

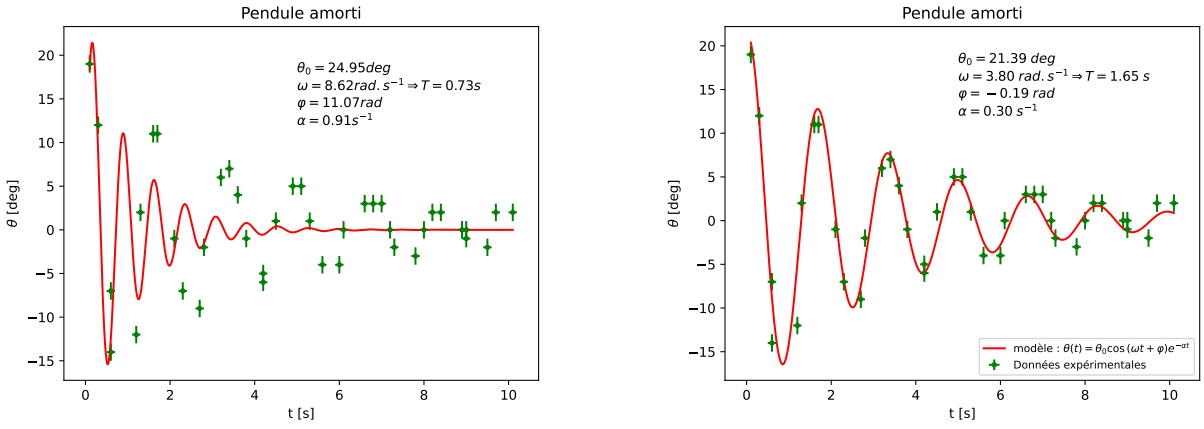
On a alors une "jolie" courbe, mais un **mauvais** ajustement comment en attestent les paramètres. Les valeurs attendues sont plutôt

$$\begin{cases} \theta_0 &= 20.00 \text{ deg} \\ \omega &= 3.76 \ rad.s^{-1} \\ \varphi &= 0 \text{ rad} \\ \alpha &= 0.32 \text{ s}^{-1} \end{cases} \Rightarrow \quad T \quad = \quad 1.67 \text{ s}$$

La raison est que l'on essaie de trouver le minimum de χ^2 pour une fonction à 4 paramètres. On est donc en train de chercher le minimum absolu d'une fonction dans un espace à 4 dimensions. Il est probable que le χ^2 présente de nombreux minimum locaux qui répondent eux aussi à la condition $d\chi^2 = 0$. Le problème est de trouver le minimum absolu. Pour cela, il vaut mieux commencer la minimisation avec un jeu de paramètres proches des paramètres optimum.

Second essai de modélisation

7. Estimer des paramètres initiaux
   - θ_{0ini}, on peut estimer de manière très grossière la valeur initiale du paramètre θ_0 comme la plus grande valeur de θ mesurée, donc $\theta_{0ini} = \max(\theta)$.
   - ω_{ini}, en observant la courbe, on peut grossièrement "estimer à l'oeil" la pseudo-période des oscillations T_{ini} et donc en déduire $\omega_{ini} = \frac{2\pi}{T_{ini}}$.
   - φ_{ini}, on observe sur la courbe que la valeur maximale de θ est obtenue pour t voisin de 0, estimer alors la phase à l'origine.
   - α, en repérant les maximas sur la courbe, on observe qu'après les trois premières périodes l'angle a été amorti d'environ un facteur 4. En déduire une estimation de α_{ini}
8. Reprendre le programme en ajoutant la liste des paramètres initiaux dans l'appel à `curve_fit` (paramètre `p0`).
   Tracer points expérimentaux et modèle sur le même graphe.



(a) *Sans donner les paramètres initiaux*

(b) *Avec de "bons" paramètres initiaux*

FIGURE 2.6 – *Ajustement sinus amorti*

Estimation des erreurs sur les paramètres d'ajustement par Monte Carlo

9. Ecrire le code qui tire aléatoirement des jeux de données et fait l'ajustement avec la fonction modèle. On stockera les valeurs obtenues des paramètres pour estimer pour chacun d'entre eux moyenne et écart-type.
10. En s'inspirant de ce qui a été fait pour la régression linéaire. Tracer le même graphe les points expérimentaux avec leurs incertitudes et le faisceau des courbes d'ajustement.

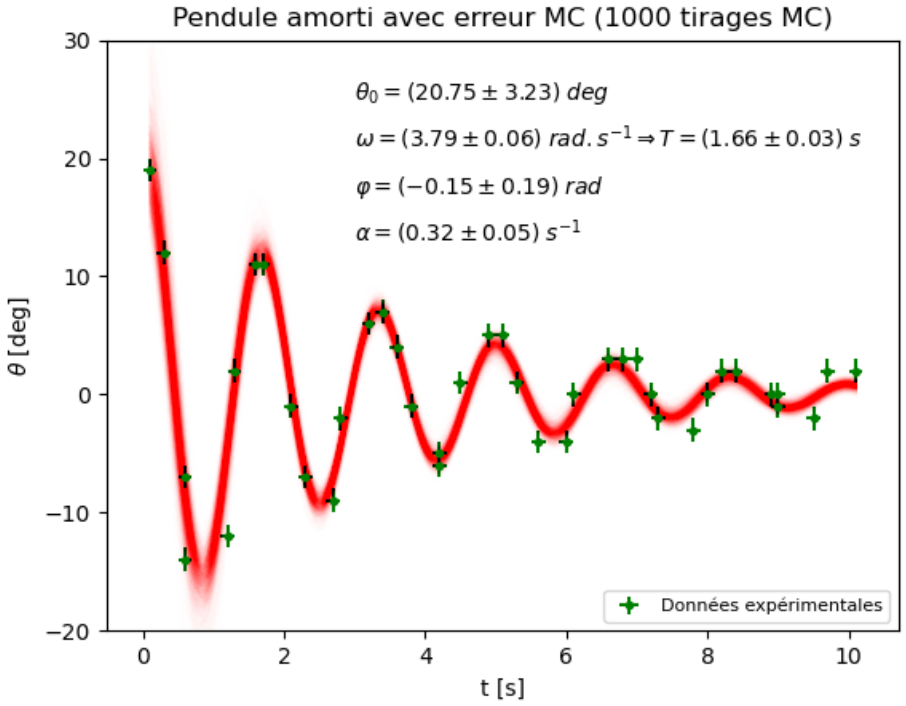


FIGURE 2.7 – *Ajustement non linéaire d'un sinus amorti avec estimation des incertitudes par Monte Carlo*