

---

## TP N° 2 : Tests d'adéquation, d'indépendance

---

Objectifs du TP : savoir réaliser des tests simples en Python (adéquation à une loi par exemple).

Quelques conseils de base : Pour chaque fiche de TP, nous utiliserons un script `jupyter notebook`. Il est conseillé de créer un répertoire HLMA408, puis un sous-répertoire de pour chaque TP, *e.g.*, TP1, TP2, TP3. Dans un tel répertoire TP1, vous stockerez :

- le sujet de TP au format PDF,
- les fichiers de données (qui seront téléchargés automatiquement),
- le fichier `TP2-squelette.ipynb` qui contient les commandes (pratiquement) pré-remplies correspondant au TP, et que vous pouvez télécharger sur le site du cours.

### 1 Test d'adéquation du $\chi^2$ à une loi donnée

La fonction `chisquare` du package `scipy.stats` permet de mettre en œuvre un test du  $\chi^2$  avec Python, voir par exemple <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html>

- 1) Importer la fonction `chisquare` mentionnée.


#### 1.1 Loi discrète

Un couple de cobayes à pelage gris et lisse a donné naissance à 64 descendants. Leurs pelages se répartissent ainsi :

- 33 gris et lisses,
- 13 blancs et lisses,
- 15 gris et rudes
- 3 blancs et rudes.

Le modèle de Mendel donne les probabilités données dans le tableau suivant :

type	effectif observé	probabilité théoriques	effectifs théoriques
gris et lisse	33	9/16	36
blanc et lisse	13	3/16	12
gris et rude	15	3/16	12
blanc et rude	3	1/16	4

- 2) Saisir les données dans un vecteur ( : `array`) ainsi

```
import numpy as np
cobaye = np.array([33, 13, 15, 3])
```

après avoir importé `numpy` sous le nom de `np`.

- 3) Saisir les probabilités théoriques :

```
mendel = np.array([9/16, 3/16, 3/16, 1/16])
```

- 4) Utiliser la commande `chisquare` pour conclure.

## 1.2 Loi continue regroupée en classe

Au préalable, nous devons d'abord comprendre deux fonctions de **pandas** : **cut** et **value\_counts**. La fonction **cut** permet de faire du regroupement par classes d'intervalles. Voyons un exemple sur un échantillon simulé de 50 observations provenant de 50 tirages aléatoires de lois gaussiennes centrées réduites. Les classes sont  $] -\infty; -2]$ ,  $] -2; -1]$ ,  $\dots$ ,  $]1; 2]$  et  $]2; +\infty[$  :

```
import pandas as pd # import de pandas
echantillon_gaussien = np.random.randn(50) # génération de données synthétiques
bins = [-np.inf, -2, -1, 0, 1, 2, np.inf]
echantillon_regroupe = pd.cut(echantillon_gaussien, bins=bins)
```

5) Affichez **cobaye\_bins** pour comprendre ce que contient l'objet créé.

La fonction **value\_counts** calcule les effectifs (le nombre d'occurrences) de chaque valeur possible dans un échantillon. Si on reprend l'échantillon regroupé calculé ci-dessus, et que l'on applique la fonction **value\_counts**, on obtient les effectifs de chacune des classes :

```
effectifs_classes = pd.value_counts(echantillon_regroupe)
```

ou de manière équivalente :

```
effectifs_classes = echantillon_regroupe.value_counts()
```

On souhaite maintenant mettre en place un test du  $\chi^2$  pour vérifier que ces données simulées proviennent bien d'une loi normale centrée réduite.

La fonction **norm.cdf** de **scipy.stats** permet de calculer les probabilités de la forme  $\mathbb{P}(Z \leq x)$  lorsque  $Z$  suit une loi normale centrée réduite et  $x$  est un nombre réel. Par exemple **norm.cdf(0)** donne 0.5 et **norm.cdf(1.96)** donne environ 0.975.

6) Que vaut la probabilité de chacune des classes ci-dessus sous la loi normale centrée réduite ?

7) Que fait la fonction **np.diff** ? Et la commande ci-dessous ?

```
np.diff(norm.cdf(bins))
```

8) Faire un test du  $\chi^2$  pour regarder si l'on peut considérer que les effectifs observés des différentes classes proviennent d'un regroupement par classe d'une loi normale centrée réduite.

9) Refaire ce qui précède avec un nouvel échantillon simulé de taille 100 et les classes suivantes :  $] -\infty; -1.5]$ ,  $] -1.5; -0.5]$ ,  $] -0.5; 0.5]$ ,  $]0.5; 1.5]$  et  $]1.5; +\infty[$ , le tout dans une unique cellule. Relancez la cellule plusieurs fois ? Observez vous des différences ? Si oui d'où viennent-elles ?

## 1.3 Un autre test de normalité (test de Shapiro-Wilk)

Le test de Shapiro-Wilk<sup>1</sup> permet de décider entre

$$\mathcal{H}_0 : "X \text{ suit une loi gaussienne}" \quad \text{vs.} \quad \mathcal{H}_1 : "X \text{ ne suit pas une loi gaussienne}"$$

Lire la page d'aide de la commande **shapiro**<sup>2</sup> pour comprendre comment celle-ci fonctionne

10) Exécuter ce test sur l'échantillon de taille 100 que l'on avait généré et conclure.

11) Exécuter ce test sur un échantillon de taille 100, d'observations tirées selon une loi exponentielle (d'espérance 1) et conclure.

1. pour les lecteurs curieux il est décrit ici : [https://fr.wikipedia.org/wiki/Test\\_de\\_Shapiro-Wilk](https://fr.wikipedia.org/wiki/Test_de_Shapiro-Wilk) ; on notera que la statistiques  $\approx 1$  sous  $\mathcal{H}_0$ .

2. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html>

## 2 Test d'adéquation du $\chi^2$ à une famille de lois

Python ne dispose pas de fonction toute faite pour répondre à ce problème. Pour cela, nous allons reprendre l'exemple de cours d'adéquation à la famille des lois de Poisson et mettre en place un test comparant

$$\mathcal{H}_0 : "X \text{ suit une loi de Poisson}" \quad \text{vs.} \quad \mathcal{H}_1 : "X \text{ ne suit pas une loi de Poisson}."$$

On rappelle que la variable  $X$  représente le nombre de palindromes dans une unité de longueur de 4000 paires de bases.

- 12) Importer et mettre en forme les données du fichier `hcmv.data` dans une dataframe nommé `df_mcv`.
- 13) Ce test suppose que l'on commence par estimer le paramètre  $\lambda$  de la loi de Poisson. Rappeler un estimateur naturel dans ce contexte. Calculer la valeur numérique sur le jeu de données et l'enregistrer dans `lambda_hat`; on pourra utiliser qu'il y a `n_basis = 229354` et compter le nombre de palindromes.
- 14) Les commandes suivantes permettent de compter le nombre de palindromes dans chacune des régions de longueur 4000 paires de bases.

```
new_bins = np.concatenate([[ -np.inf], np.arange(1, n_basis, step=4000), [ np.inf]])
hmcv_regroupe = pd.cut(df_hmcv['location'], bins=new_bins)
counts_palindrome = hmcv_regroupe.value_counts(sort=False)
```

- 15) Regrouper cette variable par classes en utilisant les bornes  $-\infty, 2, 3, 4, 5, 6, 7, 8, +\infty$ . On notera `count_regroupe` cette variable; on pourra utiliser la variable `counts_bin = [-np.inf, 2, 3, 4, 5, 6, 7, 8, np.inf]` et la fonction `cut` pour cela.
- 16) Que fait la commande ci-dessous? Comparer avec la loi normale centrée réduite vue précédemment.

```
from scipy.stats import poisson
bins_poisson = [-np.inf, 2, 3, 4, 5, 6, 7, 8, np.inf]
np.diff(poisson.cdf(bins_poisson, lambda_hat * 4000))
```

- 17) On veut utiliser la fonction `chisquare` ici avec

```
chisquare(count_regroupe, count_theoric, ddof=1)
```

où `count_theoric` est le vecteur des effectifs "théoriques" de chaque classe. Que signifie `ddof = 1` dans l'aide de la fonction `chisquare`?

- 18) Confirmer cette approche en calculant les quantiles d'un  $\chi^2$  "à la main" avec les fonctions `chi2.cdf`. Conclure.

## 3 Test d'indépendance du $\chi^2$

Il existe un test d'indépendance du  $\chi^2$  que nous n'avons pas vu en cours ni en TD. Son but est de discriminer entre les deux hypothèses suivantes :

$$\mathcal{H}_0 : "X \text{ et } Y \text{ sont indépendantes}" \quad \text{vs.} \quad \mathcal{H}_1 : "X \text{ et } Y \text{ sont liées}"$$

lorsque  $X$  et  $Y$  sont deux variables aléatoires discrètes. Rappelons que deux variables sont dites indépendantes si connaître la valeur de l'une ne donne aucune information sur la valeur de l'autre. Par exemple, si on lance deux dés simultanément, l'un rouge, l'autre blanc et que l'on modélise par  $X$  le résultat du dé blanc et  $Y$  le résultat du dé rouge, alors  $X$  et  $Y$  sont indépendants.

Pour cela, on utilise la fonction `chi2_contingency` (du module `scipy.stats`) sur un tableau de contingence croisé entre deux variables. Il s'agit d'un tableau dont chaque ligne correspond à une valeur possible de  $X$  et chaque colonne à une valeur possible de  $Y$ . Dans la case sur la ligne  $x$  et la colonne  $y$ , on compte le nombre d'observations où l'on voit simultanément  $X = x$  et  $Y = y$ . De tels tableaux de

contingence se calculent avec la fonction `pd.crosstab`. Par exemple, sur les données de la table `babies`<sup>3</sup>, chargées de la manière suivante :

```
df_babies = pd.read_csv("babies23.data", skiprows=38, sep='\s+')
```

il suffit d'exécuter :

```
pd.crosstab(df_babies['ed'], df_babies['smoke'])
```

pour obtenir un tableau de contingence croisant le niveau d'éducation des mères avec leur statut tabagique.

- 19) Calculer le tableau de contingence entre les variables `ed` et `smoke`. Que pensez-vous du résultat ? Enregistrer ce tableau dans un dataframe que l'on va appeler `cont_table`.
- 20) Lancer `chi2_contingency(cont_table)` et conclure.
- 21) En parcourant l'aide de la fonction `chi2_contingency` proposer le même test en utilisant la fonction `chisquare`.

---

3. <http://josephsalmon.eu/enseignement/datasets/babies23.data>