Fiche Python : Régression linéaire et méthode de Monte Carlo

I/ Régression linéaire

Polyfit est une fonction de la bibliothèque *numpy* qui permet, comme son nom l'indique, d'obtenir les paramètres de **modélisations polynomiales**.

Utilisation de la fonction

On commence par importer la bibliothèque *numpy*.

```
1 import numpy as np
```

On suppose qu'on dispose de deux listes :

- les données en abscisses : **X** ;
- les données en ordonnées : **Y**.

On utilise `polyfit` comme indiqué ci-dessous si on souhaite une modélisation de degré 2 du type $Y = a \cdot X^2 + b \cdot X + c$.

```
1 coefficients = np.polyfit(X, Y, 2)
```

coefficients est une liste telle que dans notre exemple :

- coefficients[**0**] est le paramètre **a** ;
- coefficients[**1**] est le paramètre **b** ;
- coefficients[**2**] est le paramètre **c** ;

<u>Remarque</u> : le nombre indiqué dans l'appel de polyfit, après les listes X et Y, correspond à l'ordre du polynôme modélisant la loi.

Dans le cas d'une loi affine, la modélisation polynomiale est d'ordre 1 :
np.**polyfit**(X, Y, 1) renvoie une liste avec les deux paramètres (pente et ordonnée à l'origine) pour le modèle affine entre X et Y.

Exemples d'utilisation de la fonction dans le cas d'une régression linéaire

- coefficient = np.polyfit(X,Y,1) : coefficient est une liste contenant deux éléments
  - ✓ Coefficient[0] correspond à la pente
  - ✓ Coefficient[1] correspond à l'ordonnée à l'origine

- a, b = np.polyfit(X,Y,1) : a correspond à la pente et b à l'ordonnée à l'origine

II/ Méthode de Monte Carlo

Les grandeurs mesurées servent souvent à calculer d'autres grandeurs.

Les grandeurs calculées à partir de grandeurs mesurées entachées d'incertitude sont également entachées d'incertitude : il y a **propagation des incertitudes.**

On note X_1 et X_2 deux grandeurs mesurées et Y une grandeur calculée à partir de X_1 et X_2 :

$Y = f(X_1,X_2)$

L'***objectif*** de la méthode de Monte Carlo est de déterminer l'incertitude-type de la grandeur calculée Y connaissant les incertitudes-type des grandeurs mesurées X_1 et X_2.

Notons $u(X_1)$ et $u(X_2)$ les incertitudes type associées aux grandeurs mesurées X_1 et X_2.

Le ***principe*** de la méthode de Monte Carlo est de calculer un grand nombre de valeurs de Y à partir de valeurs de X_1 et X_2 tirées aléatoirement dans l'intervalle $\left[X_i - \sqrt{3}\ u(X_i); X_i + \sqrt{3}\ u(X_i) \right]$ (i = 1 ou 2)

On a recours pour ce faire, à une simulation numérique à l'aide du langage Python. On définit un nombre N de tirages aléatoires. On écrit alors une boucle dans laquelle pour chaque tirage :

- On tire aléatoirement une valeur de X_1 dans l'intervalle $\left[X_1 - \sqrt{3}\ u(X_1); X_1 + \sqrt{3}\ u(X_1) \right]$
- On tire aléatoirement une valeur de X_2 dans l'intervalle $\left[X_2 - \sqrt{3}\ u(X_2); X_2 + \sqrt{3}\ u(X_2) \right]$
- On calcule la valeur Y_MC correspondant à ce tirage : Y_MC = $f(X_1,X_2)$
- On ajoute cette valeur Y_MC à un tableau qui contient les valeurs de Y_MC précédemment calculées par la même méthode.

En sortie de la boucle, on dispose donc d'un tableau, notée Y_MC, contenant N valeurs simulées de la grandeur Y.

Le résultat pour la valeur de Y sera la moyenne des valeurs du tableau Y_MC.

L'incertitude type pour Y sera l'écart type des valeurs du tableau Y_MC

Voici les outils dont on a besoin :

Modules :

- numpy as np : pour l'écriture des données sous forme de tableaux
- numpy.random : pour les tirages aléatoires

Fonctions du module numpy :

- np.**average**(A_MC) : renvoie la moyenne des valeurs du tableau A_MC
- np.**std**(A_MC, ddof = 1) : renvoie l'écart-type des valeurs du tableau A_MC

Fonction du module numpy.random :

- np.random.**uniform**(-1, 1, n) : renvoie un tableau de n valeurs tirées aléatoirement selon une loi uniforme entre -1 et 1.