Simulation Monte Carlo pour obtenir les incertitudes des paramètres d'une régression linéaire

Données (x,y)

Dans un premier temps, à titre d'exemple, nous synthétisons des données artificielles :

- `nbpts` abscisses x_i équiréparties, centrées sur `mx` et d'étendue `e` ;
- le même nombre d'ordonnées y_i, calculées à partir de $y_i = ax_i + b + \epsilon_i$, où ϵ_i est la réalisation d'une variable aléatoire normale centrée d'écart-type fixé `sig`.

Alternativement il reste possible de prendre n'importe quel jeu de données expérimentales.

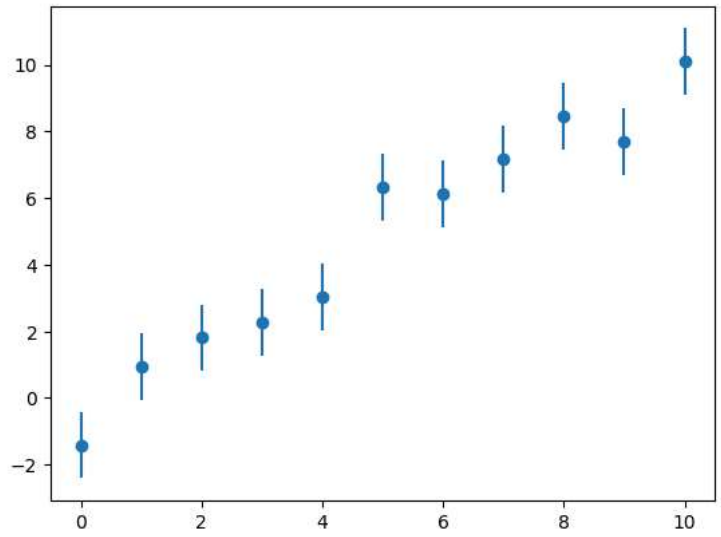
```
# Import des bibliothèques

import numpy as np
import numpy.random as rd              # Générateur de nombre aléatoires
import matplotlib.pyplot as plt        # Pour tracer divers types de graphiques

# Génération des xi et yi de synthèse (sinon prendre les xi et yi existants)

nbpts = 11 # nombre de points xi
e = 10 # étendue des xi
mx = 5 # moyenne des xi
#x = (np.arange(0,nbpts)/(nbpts-1) - 0.5)*e + mx # synthèse des xi (syntaxe python pure)
x = np.linspace(mx-e/2, mx+e/2, nbpts) # synthèse des xi (syntaxe avec numpy)
a = 1 # pente
b = 0 # ord. à l'origine
sig = 1 # incertitude-type sur chaque yi supposée indentique pour chaque yi, on peut également entrer une liste
y = a*x + b + rd.normal(0, sig, nbpts) # synthèse des yi

plt.errorbar(x, y, yerr=sig, fmt='o')
plt.show()
```



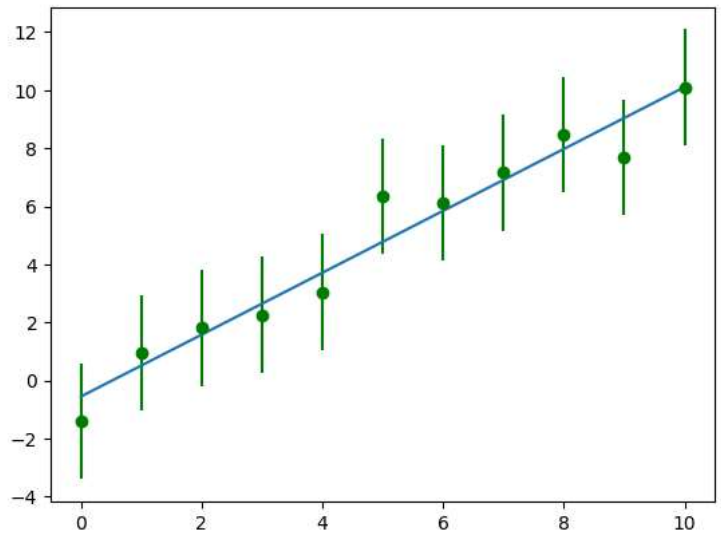
Régression linéaire ordinaire

La cellule suivante effectue les opérations suivantes :

- Tracé de la courbe $y(x)$, avec les barres d'incertitude.
- Calcul de la régression linéaire.
- Affichage des paramètres.
- Calcul des paramètres la droite de régression.
- Tracé de la droite de régression.

```
plt.errorbar(x, y, yerr=2*sig, fmt='og')
p = np.polyfit(x, y, 1)
print('pente =', p[0], '\nord. orig. =', p[1])
plt.plot(x, np.polyval(p, x)) # polyval(p, x) ou p[0]*x + p[1]
plt.show()
```

```
pente = 1.0666814753738223
ord. orig. = -0.5596211739220808
```



Simulation MC

Pour estimer les incertitudes associées aux paramètres de la régression, on simule d'autres points expérimentaux, en ajoutant des valeurs aléatoires aux ordonnées. Ces valeurs aléatoires sont choisies selon une distribution qu'on peut choisir. Ici comme on suppose connue l'incertitude-type des y_i, on utilise une loi normale centrée du même écart-type.

Il est difficile (mais pas impossible) d'utiliser les fonctions de régression linéaire sur de multiples de jeux de données de manière simultanée. Pour ce faire, à moins de maîtriser la bibliothèque scipy et les fonctions de division matricielle (par moindre carrés), il est préférable d'utiliser des boucles for appliquées à la fonction polyfit (disponible dans numpy) qui, rappelons,-le permet de trouver les paramètres d'une régression linéaire ordinaire.

Il y a deux méthodes pour réaliser la boucle :

1. soit en utilisant la méthode `.append` de Python™ ;
2. soit en utilisant les array Numpy.

Dans la première version, on crée une liste vide, on lui rajoute les éléments un par un dans la boucle, puis on convertit en array pour bénéficier de tous les avantages de ce format.

Dans la seconde version, on crée un vecteur (array unidimensionnel), rempli de zéros, à la bonne taille. Dans la boucle on substitue une par une les valeurs aux zéros. Le temps d'exécution est comparable.

Les deux cellules suivantes montrent comment faire dans ces deux cas.

```
# simulation MC
NMC = 100000
A = []
B = []
for n in range(0, NMC):
    # génération des yi
    y_sim = y + rd.normal(0, sig, nbpts)      #ici on ne génère pas de x_sim car on suppose l'incertitude type négligeable sui
    # régression linéaire ordinaire
    p = np.polyfit(x, y_sim, 1)
    A.append(p[0])
    B.append(p[1])
A = np.array(A)    # pour avoir des array
B = np.array(B)    # pour avoir des array
```

```
# simulation MC
NMC = 100000
A = np.zeros(NMC)
B = np.zeros(NMC)
for n in range(0, NMC):
    # génération des yi
    y_sim = y + rd.normal(0, sig, nbpts)
    # régression linéaire ordinaire
    p = np.polyfit(x, y_sim, 1)
    A[n] = p[0]
    B[n] = p[1]
```

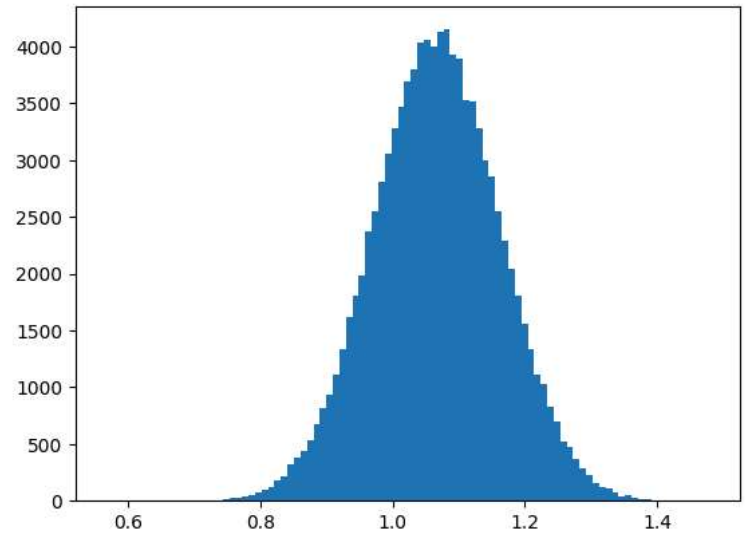
On peut représenter les résultats à l'aide d'un histogramme. On calcule la moyenne et l'écart-type des deux collections de paramètres obtenues. L'écart-type n'est autre que l'incertitude-type associée aux paramètres.

Il y a une petite différence entre la moyenne et l'estimation initiale des paramètres, sans grande importance. On pourra donc prendre indifféremment l'un ou l'autre.

Le résultat de la méthode de MC correspond bien aux prédictions des calculs analytiques.

```
plt.hist(A, bins='rice' )
print('MC : ', 'a=', np.mean(A), 'u(a)=', np.std(A, ddof=1) )
print('th : ', 'a=', a         , 'u(a)=', sig/np.sqrt(np.sum( (x-np.mean(x))**2 )) )
plt.show()
plt.hist(B, bins='rice' )
print('MC : ', 'b=', np.mean(B), 'u(b)=', np.std(B, ddof=1))
print('th : ', 'b=', b         , 'u(b)=', sig/np.sqrt(np.sum( (x-np.mean(x))**2 ))*np.sqrt(np.mean(x**2)) )
plt.show()
```

```
MC :  a= 1.0669819039731794 u(a)= 0.09533108140598522
th :  a= 1 u(a)= 0.09534625892455924
```




```
MC :  b= -0.5605347385462647 u(b)= 0.5620701113874952
th :  b= 0 u(b)= 0.5640760748177662
```