# Méthode Monte-Carlo

Lorsqu'une grandeur $X$ est **mesurée directement**, nous avons vu qu'il existe deux méthodes pour évaluer son incertitude-type : l'approche *statistique* (type A, évaluation de l'écart-type) si l'on peut répéter la mesure, ou l'approche *modélisée* si l'on ne peut pas (type B, prise en compte de la précision, des erreurs liées à la méthode, des données constructeur...).

Mais lorsqu'une grandeur $Y$ est **mesurée indirectement**, par exemple via la mesure de $X$ et une relation

$$Y = f(X)\,,$$

alors la variabilité de la mesure de $X$ engendre une variabilité de $Y$ qui dépend fortement de $f$. On parle de *propagation d'incertitude*.
À plus forte raison, lorsque la détermination de $Y$ requiert la mesure de plusieurs grandeurs $X_i$ via

$$Y = f(X_1, X_2, ..., X_p)\,,$$

l'incertitude dite *composée* engendrée par les mesures de tous les $X_i$ dépend fortement de $f$. Deux approches sont alors possibles pour évaluer $u(Y)$ connaissant $u(X)$ (ou $u(X_1)$, $u(X_2)\dots$) :

**Méthode différentielle :** utilisation de *relations de propagation* d'incertitude, établies par calcul différentiel, et dont certaines ne sont pertinentes que sous l'hypothèse de **distributions très piquées** (ie à faible incertitude relative) ;

**Méthode Monte-Carlo : simulation numérique** du processus aléatoire en supposant une distribution statistique connue pour $X$ (ou pour les $X_i$), aussi appelée méthode Monte-Carlo.

Ce cours aborde la seconde méthode, à l'aide du langage Python. Le cas échéant nous la comparerons à ce que donnerait la première. Et nous verrons comment l'appliquer à la régression linéaire, en remplcement de la méthode des droites extrêmes.
Nous aurons besoin de pouvoir :

- générer des nombres **aléatoirement** selon une distribution choisie (typiquement uniforme ou gaussienne) $\longrightarrow$ bibliothèque `random` ;
- représenter un **histogramme** d'une série de valeurs générées aléatoirement $\longrightarrow$ fonction `plt.hist()` (bibliothèque `matplotlib.pyplot`) ;
- calculer la **moyenne** et l'**écart-type** de cette série de valeurs $\longrightarrow$ fonctions `np.mean()` et `np.std()`.

## I. Cas monodimensionnel : $Y = f(X)$

Prenons l'exemple de la détermination d'une fréquence $Y = \nu$ via la mesure d'une période temporelle $X = T$ :

$$\nu = f(T) = \frac{1}{T}$$

avec une incertitude $u(T)$ supposée connue.
La méthode différentielle consisterait à donner à la fréquence la valeur $\nu = \frac{1}{T}$ (plus généralement pour une valeur mesurée $x$ la valeur donnée est $y = f(x)$), avec une incertitude relative évaluée par [1]

$$\frac{u(\nu)}{\nu} = \frac{u(T)}{T}\,.$$

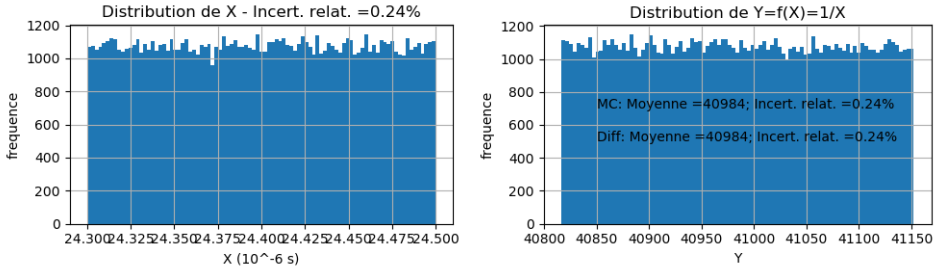
Ci-dessous on met en œuvre la méthode Monte-Carlo pour cet exemple, en supposant une distribution uniforme pour $T$ ($X$).

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

# MESURE D'UNE FRÉQUENCE VIA UNE PÉRIODE
# Période mesurée
xm = 24.4e-6 # s
# Précision de la mesure
DeltaX = 0.1e-6# s
# Incertitude (distribution uniforme supposée)
uX = DeltaX / np.sqrt(3)

def f(x):
    return 1/x # Fonction inverse

# Génération aléatoire d'une série de mesures
N = 100000 # nombre de simulations à exécuter.
vecX = np.zeros(N)
for i in range(0,N):
    vecX[i] = random.uniform(xm-DeltaX,xm+DeltaX) # spécifier l'intervalle
vecY = f(vecX) # calcul vectorisé

# Statistique sur Y
Ym = np.mean(vecY)
```

1. On obtient cette relation par différentiation logarithmique.

```
uY = np.std(vecY,ddof=1) # ddof=1 => non biaisé
print('Ym =',Ym,'; u(Y) =',uY)

# Histogrammes
plt.figure(1)
plt.hist(1e6*vecX,bins='rice')
plt.xlabel('X (10^-6 s)')
plt.ylabel('frequence')
plt.grid()
titre = "Distribution de X - Incert. relat. =" + \
        format(100*uX/xm,'.1f')+'%'
plt.title(titre)
plt.show()

plt.figure(2)
plt.hist(vecY,bins='rice')
plt.xlabel('Y')
plt.ylabel('frequence')
plt.grid()
titre = "Distribution de Y=f(X)=1/X"
plt.title(titre)
resultat_MC = 'MC: Moyenne ='+format(Ym,'.0f') + \
        '; Incert. relat. ='+format(100*uY/Ym,'.2f')+'%'
resultat_diff = 'Diff: Moyenne ='+format(f(xm),'.0f') + \
        '; Incert. relat. ='+format(100*uX/xm,'.2f')+'%'
plt.annotate(resultat_MC,(40850,700))
plt.annotate(resultat_diff,(40850,500))
plt.tight_layout()
plt.show()
```

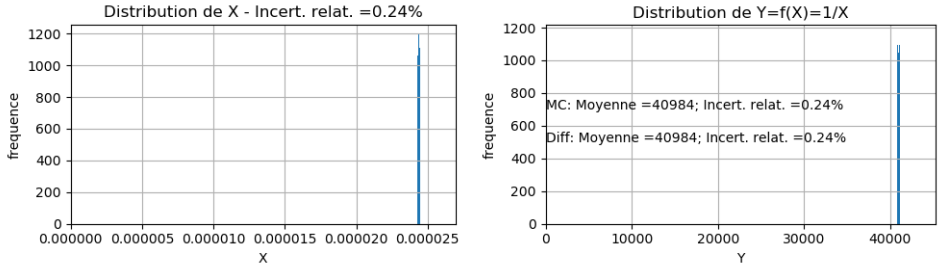
On obtient les graphes ci-dessous, sur lesquels on a inscrit les résultats.



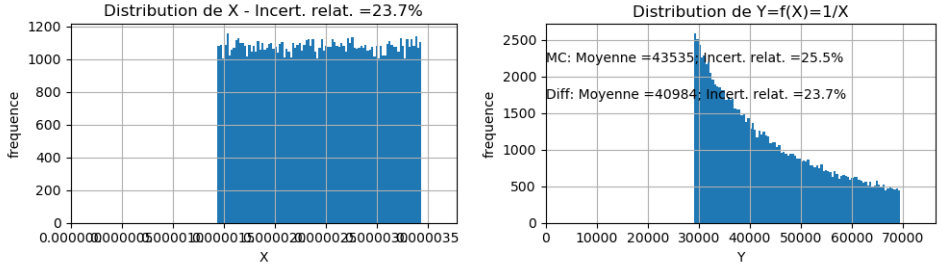

On observe que l'uniformité de la distribution de $X$ est préservée sur $Y$, et que les deux méthodes (différentielle et Monte Carlo) donnent ici la même chose, ce qui est normal car la distribution est très piquée. En effet si l'on représente l'histogramme relativement à l'origine avec

```
plt.xlim(0,1.1*max(vecX))
```

on obtient



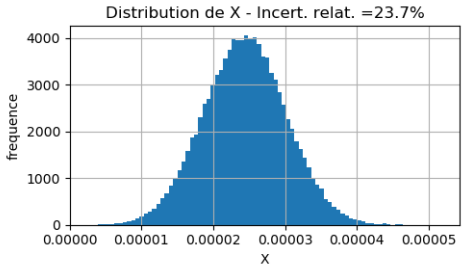

Maintenant il est intéressant de constater que si la distribution est beaucoup plus large (grande incertitude relative, on l'a multiplié par 100 ci-dessous), alors la distribution de $Y$ n'est plus du tout uniforme. La moyenne et l'écart-type s'écartent notablement des valeurs obtenues par la méthode différentielle, qui se trouve ici moins pertinente.

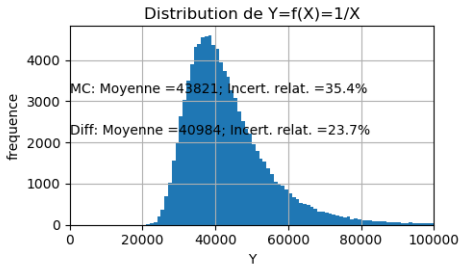



Enfin pour terminer, voyons ce qu'aurait donné une hypothèse de distribution gaussienne avec la même incertitude que précédemment, en modifiant les lignes suivantes dans le programme :

```
vecX[i] = random.gauss(xm,uX) # spécifier moyenne et écart-type
plt.hist(vecY,range=(0,100000),bins=100)
plt.xlim(0,100000)
```

On obtient de nouveau que la forme de la distribution de $X$ (caractère *centré*, c'est-à-dire symétrique) n'est pas conservée pour $Y$ en cas de grande incertitude relative. De nouveau la méthode différentielle se trouve en défaut.




## II. Cas multidimensionnel : $Y = f(X_1, X_2, ..., X_p)$

On souhaite par exemple déterminer la valeur de la capacité $C$ d'un condensateur en mesurant son temps caractéristique de décharge $\tau$ dans une résistance $R$ que l'on mesure également. On a $\tau = RC$ donc $C = \frac{\tau}{R}$, relation notée

$$Z = f(X, Y) = \frac{Y}{X}$$

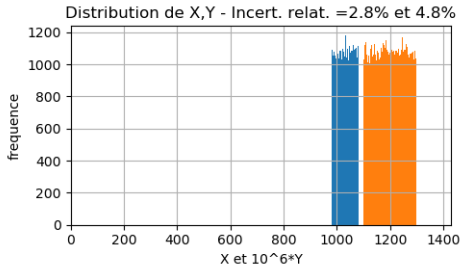
dans la suite, avec $X = R$ et $Y = \tau$.
La méthode différentielle utilise directement cette relation, et évalue

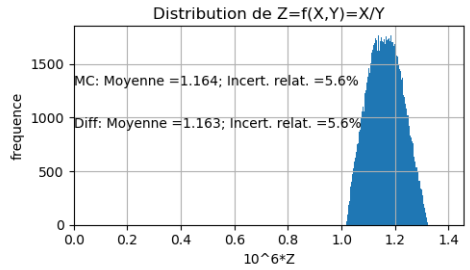
$$\frac{u(C)}{C} = \sqrt{\frac{u(\tau)^2}{\tau^2} + \frac{u(R)^2}{R^2}}.$$

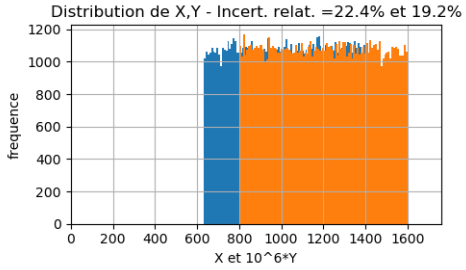
Ci-dessous on met en œuvre la méthode Monte-Carlo comme au I, toujours avec une distribution uniforme. Le programme commence comme ci-dessous, avec des choix de précision assez peu exigeants, pour illustrer.

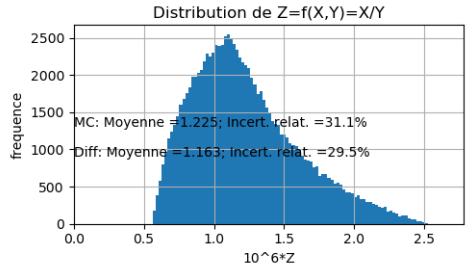
```
# Mesures (X=R ; Y=tau)
xm,ym = 1032, 0.0012 # Ohm, s
# Précision de la mesure
DeltaX,DeltaY = 50, 0.0001 # Ohm, s
# Incertitude (distribution uniforme supposée)
uX,uY = DeltaX / np.sqrt(3) , DeltaY / np.sqrt(3)

def f(x,y):
    return y/x # Fonction quotient

# Génération aléatoire d'une série de mesures
N = 100000 # nombre de simulations à exécuter.
for i in range(0,N):
    vecX[i] = random.uniform(xm-DeltaX,xm+DeltaX)
    vecY[i] = random.uniform(ym-DeltaY,ym+DeltaY)
vecZ = f(vecX,vecY) # calcul vectorisé

# Statistique sur Z
Zm = np.mean(vecZ)
uZ = np.std(vecZ,ddof=1) # ddof=1 => non biaisé
print('Zm = ',Zm,' ; u(Z) = ',uZ)

...
```

Puis on représente les distributions superposées de $X$ et $Y$ d'une part ($Y$ en $\mu s$ pour superposition), et de $Z$ d'autre part (en $\mu F$). De nouveau on observe que la méthode différentielle est équivalente pour des incertitudes relatives de l'ordre de quelques %, mais devient incorrecte aux alentours de 20%. Toutefois il est notable que l'écart reste faible entre les deux méthodes. On observe aussi que la distribution de $Z$ n'a plus rien à voir avec une distribution uniforme dans les deux cas, à condition que les distributions de $X$ et $Y$ soient de largeur relative comparable.



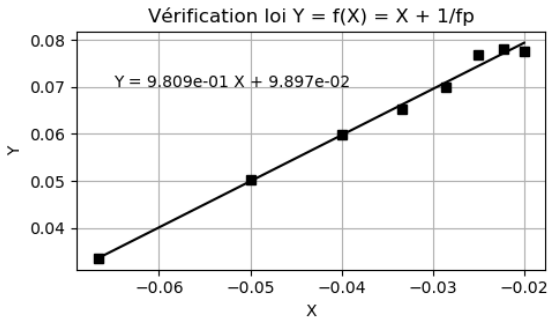
## III. Application à la régression linéaire

On souhaite vérifier la loi de Descartes de la conjugaison par une lentille mince

$$\frac{1}{\overline{OA'}} = \frac{1}{f'} + \frac{1}{\overline{OA}} \quad \Leftrightarrow \quad Y = \frac{1}{f'} + X \quad \text{avec} \quad X = \frac{1}{\overline{OA}} \quad \text{et} \quad Y = \frac{1}{\overline{OA'}}\,.$$

On saisi les mesures, puis on calcule et trace la droite de régression.

```
# Mesures
OA = -np.array([15.,20.,25.,30.,35.,40.,45.,50.])
OAp = np.array([29.9, 19.9, 16.7, 15.3, 14.3, 13.0 , 12.8, 12.9])
X = 1/OA
Y = 1/OAp

a,b = np.polyfit(X,Y,1)
# Valeurs calculées par le modèle
Y_reg = b + a*X
# Fabriquer l'équation de la courbe (type string)
equation = 'Y = '+format(a,'.3e')+' X + '+format(b,'.3e')

plt.figure(figsize=[5,3])
plt.plot(X,Y,'s',color='black')
plt.plot(X,Y_reg,'-k')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Verification loi Y = f(X) = X + 1/fp')
plt.annotate(equation,(-0.065,0.07))
plt.grid()
plt.tight_layout()
plt.savefig('reglin.png')
plt.show()
```




Chaque longueur est supposée déterminée par deux pointés de viseur de précision $\Delta x$ connue, ce qui permet de calculer les incertitudes $u(X)$ et $u(Y)$ par relation de propagation[2] pour chaque point. On construit ensuite $N$ séries de points de mesure modifiés aléatoirement via des distributions uniformes. Pour chacune on obtient des paramètres de régression $(a, b)$ différents que l'on stocke, afin de déterminer in fine leur moyenne et leur écart-type et de tracer leur histogramme.

```
# Incertitudes
Delta_x = 0.3 # précision pointé viseur
uOA = np.sqrt(2/3)*Delta_x
uOAp = uOA
uX = uOA/OA**2
uY = uOAp/OAp**2

# Génération aléatoire d'une série de séries de mesures
N = 100000 # nombre de simulations à exécuter.
veca,vecb = np.zeros(N),np.zeros(N)
Xmod,Ymod = np.copy(X),np.copy(Y)
for i in range(0,N):
    for k in range(len(X)):
        Xmod[k] = random.uniform(X[k]-uX[k],X[k]+uX[k])
        Ymod[k] = random.uniform(Y[k]-uY[k],Y[k]+uY[k])
    veca[i],vecb[i] = np.polyfit(Xmod,Ymod,1)

# Statistique sur Z
am = np.mean(veca)
bm = np.mean(vecb)
ua = np.std(veca,ddof=1) # ddof=1 => non biaisé
ub = np.std(vecb,ddof=1) # ddof=1 => non biaisé
print('am =',am,'; u(a) =',ua)
print('bm =',bm,'; u(b) =',ub)

plt.figure(1)
plt.hist(veca,bins='rice')
plt.xlabel('a')
plt.ylabel('frequence')
plt.grid()
titre = "Coeff dir a="+ format(am,'.4f') + \
        ' +- '+ format(ua,'.4f')
plt.title(titre)
plt.tight_layout()
plt.savefig('reglin_hist_a.png')
```

2. On peut aussi les évaluer directement par Monte-Carlo en reprenant la démarche des sections précédentes.

```
plt.show()

plt.figure(2)
plt.hist(vecb,bins='rice')
plt.xlabel('b')
plt.ylabel('frequence')
plt.grid()
titre = "Ordonnee a l'origine b="+ format(bm,'.4f') + \
        ' +- '+ format(ub,'.4f')
plt.title(titre)
plt.tight_layout()
plt.savefig('reglin_hist_b.png')
plt.show()
```

On obtient des distributions quasi gaussiennes pour $a$ et $b$.