Janvier 2025

TP Python « INCERTITUDES »
Méthode de MONTE CARLO

Nous avons vu lors du premier TP quelques éléments de calcul de propagation des incertitudes de mesure (formules quadratiques en particulier). ***Nous n'allons rien démontrer par les théories statistiques mais nous allons confirmer les résultats numériques par des tirages au sort dans des distributions définies****. On pourra décrire succinctement à cette occasion une distribution statistique fondamentale : la* ***distribution normale*** *ou* ***gaussienne*** *(interprétant alors l' « écart de mesure » comme le cumul de microévénements contraires équiprobables). (Le théorème central limite lui confère une importance considérable en statistiques)*

Ce TP devra remplir deux objectifs :

1- Confirmer sur un exemple les « formules » de propagation de l'incertitude (LPU) par la méthode statistique de Monte Carlo (MMC)

Cette **M**éthode de **M**onte **C**arlo nous permettra de simuler diverses distributions de variables (uniformes, triangulaires ou gaussiennes) ainsi que la distribution de la grandeur « fonction de ces variables » pour mesurer son écart-type, sa moyenne et comparer à la prédiction LPU. Nous l'appliquerons à une forme monomiale à plusieurs variables (cas ultra-fréquent)

Deux documents annexes sont joints (à consulter avant la séance) :

- un tableau comparatif du déroulement des deux méthodes d'évaluation LPU et MMC
- un article scientifique illustrant la MMC sur deux exemples types de physique simples (mesures de puissance thermique et de masse conventionnelle) *(Le second exemple montrant la supériorité de la MMC dans le cas de défauts de linéarité où les calculs de LPU doivent être menés à un degré supérieur : LPU2 contre LPU1)*

2- Utiliser des distributions de dispersion pour estimer l'incertitude sur la pente et l'ordonnée à l'origine d'une modélisation affine.

Nous partirons alors d'un exemple de TP (*TP B: Diffraction-Interférences de la série tournante 1*) pour lui appliquer la MMC : des distributions uniformes construites à partir de la variabilité de lectures de distances seront assignées aux variables « x » et « y » d'une loi devant s'avérer affine. Alors qu'une régression linéaire usuelle sur les mesures (par la méthode des « moindres carrés ») ne fournit qu'une proposition de droite (une pente et une ordonnée à l'origine), la MMC permettra d'obtenir une distribution de ces deux variables (pente et ordonnée) et donc d'estimer leur variabilité pour en déduire la variabilité de la grandeur que l'on cherchait à estimer par cette mesure.

I. Comparaison LPU et MMC

Les formules quadratiques de propagations de l'erreur nécessitent généralement des calculs de dérivées partielles (premières au moins) de la grandeur finale par rapport à chacune de ses variables d'influence (cf article annexe). On y incorpore également les incertitudes-type évaluées en « choisissant » des distributions diverses pour chaque variable (rectangulaire (ou uniforme), triangulaire, « dérivée d'arcsinus » , gaussienne etc.). La relation (connue) entre demi-largeur et incertitude-type dépend donc du profil de densité de probabilité de la distribution.

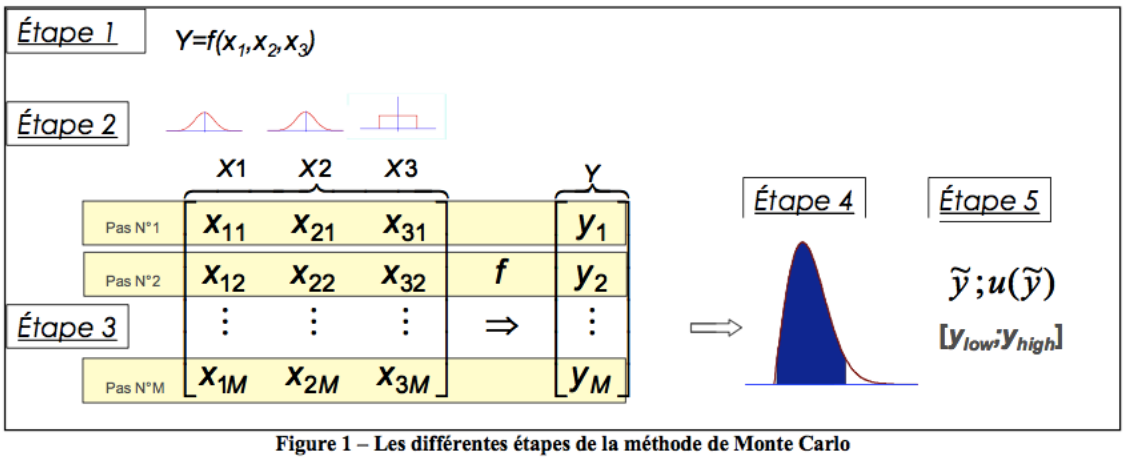
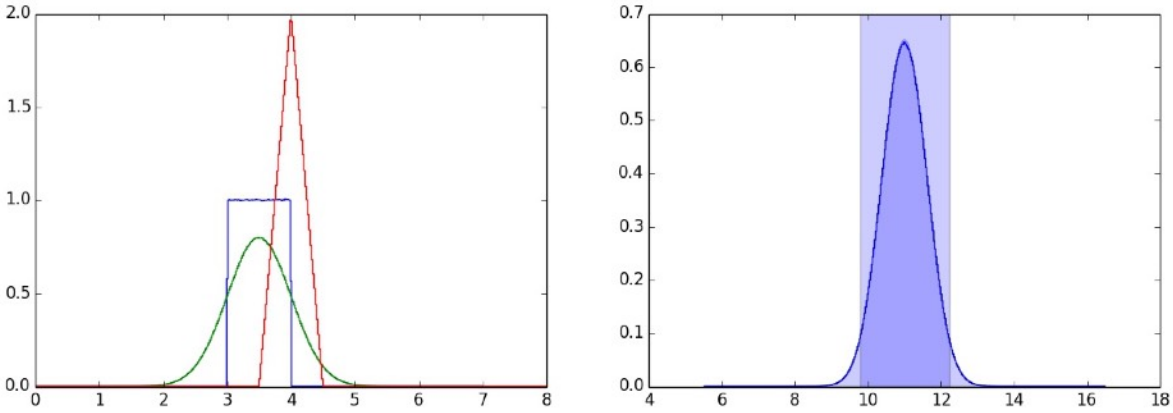


Figure 1 – Les différentes étapes de la méthode de Monte Carlo

Grâce au module « random » de fonctions aléatoires de **numpy**, nous allons pouvoir réaliser la méthode de Monte Carlo et ainsi valider les formules quadratiques utilisées en LPU.

Il s'agit donc de simulations de tirages pour obtenir la distribution empirique de la grandeur de sortie. On simule la PDF (fonction densité de probabilité) de chacune des variables. On génère 1 million de réalisations par tirage au sort dans la distribution de chaque variable. On obtient alors 1 million de valeurs pour la grandeur de sortie qui constituent la distribution empirique du mesurande. Il reste alors à calculer : moyenne, estimateur d'écart-type pour bâtir éventuellement un intervalle de confiance à 95% (HP) et comparer à celui qu'aurait donné la LPU.

On en profitera pour remarquer que des distributions différentes donneront souvent une distribution convergeant vers une gaussienne :



« La convolution d'un grand nombre de fonctions de densités de probabilités tend toujours vers une fonction gaussienne. »

La densité de probabilité de la **loi normale (loi de Gauss)** autour d'une moyenne nulle s'écrit :

$$\frac{dP}{dx} = \frac{1}{\sigma\sqrt{2\pi}} . e^{-\frac{x^2}{2\sigma^2}}$$

Les bibliothèques de fonctions nécessaires seront :
-**numpy** à renommer **np**
-**matplotlib.pyplot** à renommer **plt**
-**numpy.random** à renommer **npr**

A vous de vous renseigner sur les arguments de distributions :

- Rectangulaire (ou uniforme) : **npr.uniform**()
- Triangulaire : **npr.triangular**()
- Gaussienne (ou Normale) **: npr.normal**()

Nous allons utiliser le programme monômeMMC dont vous disposez d'une version à compléter et dont j'extrais ici les éléments à coder en python :

```
#Création d'un échantillon issu d'une population de densité de probabilité uniforme sur un intervalle (rectangulair
ars=3.12;brs=3.17;
# ..........................

#Calcul de l'incertitude-type de la distribution rectangulaire
urs=(brs-ars)/(2*np.sqrt(3));

#Calcul des moyenne et estimateur de l'écart-type de l'échantillon rectangulaire
moyrs=np.mean(rs);sigmans=np.std(rs);

#préparation d'un objet graphique : histogramme rectangulaire (normalisé) à présenter en premier dans une fenêtre g
plt.clf()
plt.subplot(1,2,1)
plt.hist(rs, bins=4000,range=(0,2*moyrs),density=True,histtype='step', linewidth=1)

#Création d'un échantillon issu d'une population de densité de probabilité gaussienne avec écart-type et moyenne fi
# ..........................

#Calcul des moyenne et estimateur de l'écart-type de l'échantillon gaussien
moyns=np.mean(ns);sigmans=np.std(ns);

#préparation d'un objet graphique  : histogramme gaussien (normalisé)
# ..........................

#Création d'un échantillon issu d'une population de densité de probabilité triangulaire
ats=3.6;bts=3.8;mts=(bts+ats)/2;
# ..........................

#Calcul de l'incertitude-type de la distribution triangulaire par la LPU
uts=(bts-ats)/(2*np.sqrt(6));

#Calcul de la moyenne et de l'estimateur de l'écart-type de l'échantillon triangulaire
moyts=np.mean(ts);sigmats=np.std(ts);

#préparation d'un objet graphique : histogramme triangulaire (normalisé)
# ..........................

#réalisation de l'échantillon fonction monômiale des trois variables
prodquot=ns*ts/(rs*rs);

#Calcul de la moyenne et de l'estimateur de l'écart-type de la distribution de la variable ""monôme des trois varia
moyprodquot=np.mean(prodquot);sigmaprodquot=np.std(prodquot);

#affichage console des moyennes et incertitudes-type pour LPU
print (moyrs,'±',urs)
print (moyns,'±',sigmans)
print (moyts,'±',uts)

#Calcul de l'incertitude-type sur la variable "monôme" par la LPU
# ..........................
```

Commenter l'allure de la distribution de la variable calculée. Augmenter les variabilités relatives (en particulier sur rs) et observer la « déformation » de la distribution finale et l'écart entre LPU et MMC.

II. MMC utilisée sur des régressions affines

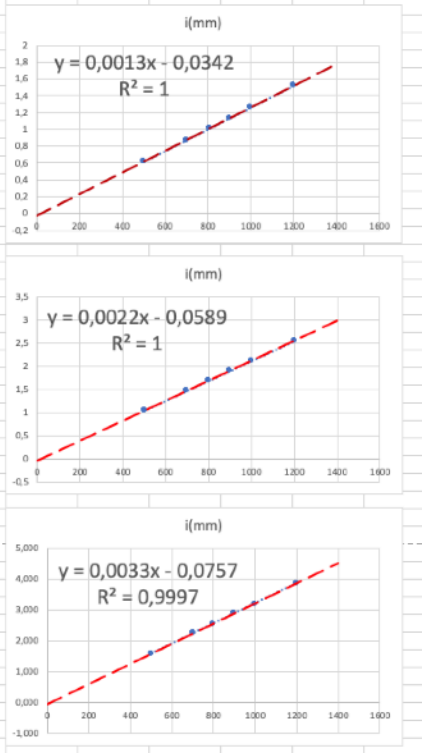
Lors du TP B (série 1) , vous avez dû utiliser des régressions linéaires pour déterminer l'écart entre les deux fentes à partir de mesures précises d'interfranges i (barrette CCD et logiciel CALIENS) et de distances d'éloignement « objet diffractant » - « barette » $b_{mesurée}$ évaluée sur banc d'optique.
Dans un premier temps, on vous a fait remarquer qu'il fallait éviter le modèle :

$$i = \frac{\lambda_0 . b}{a} \text{ pour lui préférer } i = \frac{\lambda_0 . b_{mes}}{a} \pm \frac{\lambda_0 . b_0}{a}$$

pour préserver la précision de mesure sur banc malgré le décalage b_0 inconnu mais invariable entre les repères de supports et les objets supportés (double-fentes et CCD)

Dans un deuxième temps, vous avez reporté les valeurs mesurées dans un tableur pour obtenir un graphe superposant aux mesures la régression linéaire proposée avec son équation (éventuellement le carré du coefficient de corrélation R^2 pour vous « rassurer »). Ci-dessous mes 3 séries de mesures :

	A	B	C	D	E	F	G	H
1	Double fentes							
2								
3	large							
4	500µ	bmesurée (cm)	50	70	81	90	100	120
5		10*i (mm)	6,14	8,7	10,12	11,27	12,61	15,18
6		bmesurée (mm)	500	700	810	900	1000	1200
7		i(mm)	0,614	0,87	1,012	1,127	1,261	1,518
8								
9								
10		pente précise	0,0012930					
11								
12		a	489,40					
13								
14								
15								
16								
17								
18								
19								
20	moyenne							
21	300µ	bmesurée (cm)	50	70	80	90	100	120
22		5*i (mm)	5,13	7,27	8,35	9,43	10,52	12,7
23		bmesurée (mm)	500	700	800	900	1000	1200
24		i(mm)	1,026	1,454	1,67	1,886	2,104	2,54
25								
26								
27		pente précise	0,0021634					
28								
29		a	292,50					
30								
31								
32								
33								
34								
35	étroite							
36	200µ	bmesurée (cm)	50	70	80	90	100	120
37		3*i (mm)	4,67	6,67	7,59	8,6	9,5	11,57
38		bmesurée (mm)	500	700	800	900	1000	1200
39		i(mm)	1,557	2,223	2,530	2,867	3,167	3,857
40								
41								
42		pente précise	0,0032655					
43								
44		a	193,78					



On estime ainsi une valeur de a par expérience mais nous n'avons pas vraiment d'information sur la variabilité associée.

Troisième et dernier temps :

On va alors utiliser la MCM et les régressions sous Python (**np.polyfit**) pour calculer une distribution considérable de pentes de droite compatibles avec notre proposition de variabilité sur les mesures de i et de b. On représentera l'histogramme de cette distribution (commentaire) et on estimera sa variabilité par l'estimateur de l'écart-type pour conclure sur la cohérence (ou non) de nos mesures avec la précision affichée par le constructeur sur ces double-fentes.

Les mesures de $b_{mesurée}$ sur banc seront estimées à une précision de lecture de ±2 mm sur chaque abscisse donc ±4 mm au total

Les mesures de i sont faites en réalité sur plusieurs interfranges dans la tache de diffraction centrale. Le positionnement des minima d'intensité lumineuse ne sont pas repérables au pixel près : on estimera cela à environ ±3 pixels (de 14µm chacun) [ils sont rectangulaires par ailleurs : 56µm de hauteur].

Ce qui justifie ces variabilités :

```
#-----------------------------------------------
# Variabilité estimée des lectures de distances
#-----------------------------------------------
# ±4 mm sur les mesures de positions de support sur banc
u_distances=4.0
# ±3 pixels pour les mesures de 10*i puis 5*i puis 3*i
u_interfranges500=(3*0.014)/10
u_interfranges300=(3*0.014)/5
u_interfranges200=0.014
```

Deux fonctions vous seront nécessaires pour compléter le programme et le tester ensuite sur un « tirage au sort » d'un échantillon (« population ») de 100000 valeurs.

-**Une fonction « regressionlinéairesimple »** dont les trois arguments seront :

-distancessupports (tableau numpy)
-interfranges (tableau numpy)
-decimales (entier pour la précision d'affichage des valeurs)

qui renvoie :

- le graphe des valeurs
- La pente de la régression affine (print dans la fonction)
- L'ordonnée de la régression affine (print dans la fonction)
- Une valeur de a (print dans la fonction)

```
#-------------------------------------
# Fonction régression linéaire simple
#-------------------------------------
def regressionlinéairesimple(distancessupports,interfranges,decimales):
    plt.figure()
    plt.plot(distancessupports,interfranges,'r+')
    #.....;
    print ("pente=",pente)
    print ("ordonnée=",ordonnée)
    print ("estimation de l'écart a entre les fentes (en µm) par régression linéaire",round(0.6328/pente,decimales))
```

-**Une fonction «Variabilitéa »** dont les cinq arguments seront :

-distancessupports (tableau numpy)
-interfranges (tableau numpy)
-u_distances (flottant représentant des mm)
-u_interfranges (flottant représentant des mm)
-population (entier choisi 100000 dans un premier temps)

qui renvoie :

- Le graphe de l'histogramme des valeurs de a
- La valeur de a moyenne
- L'écart-type de la distribution de a estimant sa variabilité

```
#-------------------------------------------------------------------------------------
# Fonction Variabilité utilisant la MCM pour estimer la variabilité sur l'écart du doublet
#-------------------------------------------------------------------------------------

def  Variabilitéa(distancessupports,interfranges,u_distances,u_interfranges,population)    :
    liste_tiragea=[]
    for i in range(0,population):
        l = len(distancessupports)
        # tiragedistances = .....
        # tirageinterfranges = .....
        tiragea=0.6328/np.polyfit(tiragedistances,tirageinterfranges,1)[0]
        liste_tiragea.append(tiragea)
    plt.figure()
    plt.hist(liste_tiragea,500,density=True,histtype='step')
    print('La valeur moyenne de a est de',round(np.mean(liste_tiragea),1),'µm')
    print('La variabilité de a est estimée à ±',round(np.std(liste_tiragea),1),'µm')
```