Université Lyon 1 | Année 2020-2021
Master Mathématiques Générales 1ère année
Logiciels Scientifiques

TP4 – Option A

Pour tout le TP, on pourra entrer les commandes

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
import scipy.integrate as integ
```

1 Aléatoire

Exercice 1 - Fonctions utiles

Tester les commandes suivantes .

```
M=np.array([[1, 3, 5], [-2, 6, 1]])
M>2
M[M>2]
indices=np.nonzero(M>2)

a=-2; b=2; h=0.5
x=np.arange(a, b+h/2, h)
y=np.linspace(a, b, round((b-a)/h+1))

l=np.logical_and(x>-1, x<1)
x[l]=3.14

np.cumsum(x)
np.sum(x)
np.mean(x)

poids=np.ones(x.size)
poids[0::2]=2
np.average(x)
np.average(x, weights=poids)

np.sort(x)
-np.sort(-x)
np.flip(np.sort(x),axis=0)
np.flipud(np.sort(x))

uni_x,num_x=np.unique(x, return_counts=True)
```

Exercice 2 - Génération d'aléa

Entrer les commandes suivantes et aller voir l'aide pour chacune d'entres elles.

```
npr.rand()
npr.rand(2, 4)
npr.randint(-5, 10, (2, 10))
a=-1; b=1;
npr.uniform(a, b, 1000)
lambda=2
npr.exponential(lambda,1000)
mu=1; sigma=2;
npr.normal(mu, sigma, 1000)
n=5; p=0.2;
npr.binomial(n, p, 1000)
```

La fonction `npr.rand` dépend d'une "graine", une suite (u_n) initialisée à l'ouverture de Python. À graine fixée, `npr.rand` fournira toujours la même série de valeurs. La commande `npr.seed(k)` permet de fixer la valeur de cette graine à k.

```
npr.seed(10);
npr.rand()
npr.rand()

npr.seed(10)
npr.rand()
```

Exercice 3 - Affichage

```
N=1000
tab_n=np.arange(1, N+1)
plt.plot(tab_n, np.cumsum(npr.rand(N))/tab_n)
plt.show()

n=10; p=0.2
u=npr.binomial(n, p, N)
x=np.sort(u)
y=tab_n/N
plt.plot(x, y)
plt.axis([-0.5, n, 0, 1])
plt.show()

U=npr.rand(2, N)
plt.plot(U[0, :], U[1, :], 'r.')
plt.show()
```

La commande `npr.hist` permet de tracer des histogrammes. (voir l'aide pour les options)

```
L=npr.normal(2, 1, 1000)
plt.hist(L, bins=100);
plt.show()
plt.hist(L, bins='auto')
plt.show()
plt.hist(L, bins='auto', density=True)
plt.show()
```

---

2 Exercices d'application

Exercice 1 - Lancé de dés

1. Écrire une fonction `dice()` simulant le résultat du lancer d'un dé non pipé à six faces.
2. Tester la validité de cette fonction en calculant les fréquences de sortie de chacun des chiffres au bout de $N = 100000$ tirages.

Exercice 2 - Paradoxe du chevalier de Méré

Le chevalier de Méré (1607-1684) qui était un grand joueur, avait remarqué que, lorsqu'on joue au dé, parier sur l'apparition d'un 6 en lançant 4 fois le dé était avantageux. Par un argument d'homothétie, il avait conclu que parier sur l'apparition d'un double-six quand on lance 24 fois deux dés l'était aussi.

1. En effectuant un grand nombre de simulations, vérifier que le premier jeu est avantageux.
2. En effectuant un grand nombre de simulations, constater que le second jeu ne l'est pas.

Exercice 3 - Problème des anniversaires

Soit une assemblée de p individus dont aucun n'est né un 29 février.

1. Écrire une fonction `anniv(p)` qui retourne 1 si deux individus sont nés le même jour, et 0 sinon.
2. Écrire une fonction `ProbAnniv(N,p)` qui retourne l'approximation de la probabilité d'avoir eu deux individus nés le même jour pour N simulations, et qui trace l'évolution au cours du calcul de l'approximation en fonction du nombre d'expériences déjà réalisées.
3. Pour $N = 1000$, effectuer les simulations avec $p = 24$, puis $p = 35$

Exercice 4 - Lecture de données

1. Télécharger le fichier de données `Nil.txt` à l'adresse `https://agreg.org/Textes/Nil.txt` qui contient des niveaux maximaux du Nil, relevés chaque année entre 722 et 1281.
2. Importer les données en utilisant la commande `np.loadtxt("Nil.txt", skiprows=1)`. Les données sont alors stockées dans un tableau A de type `array`.
3. Calculer la moyenne et la variance des données de niveaux en utilisant la commande `np.mean`.
4. Tracer l'histogramme des données de niveaux maximaux.

Exercice 5 - Mouvement brownien

Le mouvement brownien est une description mathématique du mouvement aléatoire d'une "grosse" particule immergée dans un fluide et qui n'est soumise à aucune autre interaction que des chocs avec les "petites" molécules du fluide environnant.
On simule ce mouvement en considèrant une particule se déplaçant suivant un axe (Ox). Aux instants $t_k = k\Delta t$, elle effectue un déplacement X_k. Chaque déplacement X_k est une variable aléatoire suivant une loi uniforme $U[-1, 1]$. On note $R_n = R(n\Delta t)$ la position de la particule après n intervalles Δt. On notera $T = N\Delta t$ le temps final de la simulation.

1. Écrire un programme `brownien(R0,n)` simulant la position R_n d'une particule avec $R_0 = 0$ et $n = 10000$.
2. Tracer une trajectoire correspondant à une réalisation du programme précédent.
3. On considère $p = 2000$ réalisations du programme précédent.
   (a) Tracer sur une même figure la moyenne arithmétique $\langle R_n \rangle$ et la moyenne quadratique $\sqrt{\langle R_n^2 \rangle}$ de la position des particules pour chaque pas de temps sur les p réalisations.
   (b) Vérifier que $\langle R_n^2 \rangle$ est proportionnel au temps.
   (c) Tracer sur une meme figure l'histogramme des R_N pour les p réalisations. Que constatez-vous ?
4. Modifier le programme `brownien` pour simuler la position R_n d'une particule dans $\mathbb{R}^2$ en prenant $R(0) = (0, 0)$ pour point de départ.
5. Tracer une trajectoire correspondant à une réalisation du programme $2D$ précédent.

---

3 Méthodes d'approximation

Exercice 1 - Méthode de Monte-Carlo

Soient deux solides S_i et S tels que $S_i \subset S \subset \mathbb{R}^d$. Le volume V de S est supposé connu, et on cherche une approximation du volume V_i de S_i. La méthode de Monte-Carlo consiste à tirer aléatoirement n points dans S avec une probabilité uniforme, et compter le nombre de points n_i qui sont situés dans S_i.

Une approximation de V_i est alors donnée par :

$$V_i \approx R(n) = V\,\frac{n_i}{n}$$

Le but est de calculer une approximation du volume de la boule unité $\mathcal{B}_d(0, 1)$ placée dans le cube $\mathcal{R}_d = [-1, 1]^d$ par la méthode de Monte-Carlo.

1. La suite permettant de calculer le volume d'une boule de dimension n quelconque est :

$$\begin{cases} V_1 = 2 \\ V_{n+1} = V_n \displaystyle\int_{-1}^{1} (1-x^2)^{n/2} dx, \quad n \geq 1 \end{cases}$$

   Écrire une fonction `VolHyperboule(d)` qui calcule numériquement le volume de $\mathcal{B}_d(0,1)$.
   $\hookrightarrow$ *Utiliser la commande* `integ.quad`.
2. Écrire une fonction `boule(n,d)` qui calcule une approximation du volume de $\mathcal{B}_d(0,1)$ située dans $\mathcal{R}_d$ par la méthode de Monte-Carlo avec n points et pour d quelconque.
3. On considère $d = 2$.
   (a) Tracer $R(n)$ pour $n \leq 5000$.
   (b) Comparer la valeur moyenne obtenue pour $n \in [2500, 5000]$ avec la valeur théorique.
4. Mêmes questions pour $d = 3$.
5. Pour $d = 20$, calculer la valeur moyenne de $R(n)$ pour $n \in [2500, 5000]$ et comparer avec la valeur numérique obtenue avec la suite (V_n). Qu'en pensez-vous ? Recommencer pour $n \in [22500, 25000]$

Exercice 2 - Densité à support compact

On souhaite simuler une variable aléatoire Z sachant un événement A. Pour cela, on peut simuler de manière répétée et indépendante (Z, A), et rejeter le résultat quand A n'est pas réalisé.
On considère un trifolium de Descartes dont les équations paramétrique et cartésiennes sont

$$\theta \mapsto \begin{cases} x = \cos(3\theta)\cos(\theta) \\ y = \cos(3\theta)\sin(\theta) \end{cases} \quad \text{pour } \theta \in [0, \pi] \qquad (1)$$

$$(x^2+y^2)^2 - x(x^2-3y^2) = 0 \qquad (2)$$

Le graphe est alors inclu dans le rectangle $[-1, 1]^2$.

1. Tracer un trifolium en utilisant l'équation (1).
2. Écrire une fonction `RejetTrefle(n)` génèrant une échantillon de loi uniforme de taille n à l'intérieur du trifolium par la méthode du rejet en utilisant l'équation (2).
3. Vérifier que l'échantillon obtenu suit une loi uniforme en traçant les points obtenus.

Exercice 3 - Densité à support non compact

On considère une variable aléatoire X de densité f connue mais dont la simulation directe n'est pas facile, et une variable aléatoire Y de densité g vérifiant

$$\exists c \in \mathbb{R}, \forall x \in R, \ f(x) \leq cg(x) \qquad (3)$$

Soit U une loi uniforme sur $[0, 1]$ indépendante de Y, la loi de X est alors la loi de Y sachant l'événement $E = cg(Y)U < f(Y)$.
On souhaite construire une échantillon suivant la loi normale réduite centrée de densité f en utilisant la loi de Laplace standard de densité g. On peut montrer que dans ce cas $c = \sqrt{2e/\pi}$.

1. Vérifier graphiquement l'inégalité (3) en traçant f et cg sur $[-6, 6]$.
2. Contruire un échantillon X de taille $N = 10000$ suivant la loi normale centrée réduite par la méthode du rejet à partir de la loi de Laplace. On utilisera pour cela le fait que si u suit la loi uniforme sur $[0, 1]$, alors $y = \ln(2u)\mathbb{1}_{u<1/2} - \ln(2(1-u))\mathbb{1}_{u\geq 1/2}$ suit la loi de Laplace.
3. Tracer l'histogramme correspondant à l'échantillon X et la densité f.

Exercice 4 - Markov Chain Monte Carlo

Dans le cas d'une densité en dimension supérieur à 1, la méthode de rejet de l'exercice précédent est moins adaptée. On lui préfère des méthodes du type Markov Chain Monte Carlo (MCMC) qui consiste à simuler une chaîne de Markov passant plus souvent dans les zones de forte probabilité et moins dans les zones de faible probabilité.

1− On construit un échantillon 2D suivant la loi dont la densité s'écrit :

$$f(x, y) = \frac{1}{I_f} g(x)g(y)$$

où I_f est un coefficient de normalisation de sorte que $\displaystyle\int_{\mathbb{R}^2} f = 1$ et avec

$$g(x) = e^{-x^2}(2 + \sin(5x)\sin(2x)).$$

1. Écrire une fonction `g(x)` qui prend un tableau x en argument et retourne un tableau contenant les valeurs de g en chaque élément de x.
2. Calculer mathématiquement l'intégrale de g sur $\mathbb{R}$. En déduire I_f.
3. Écrire une fonction `f(x, y)` qui retourne les évaluations de f au point (x, y).
4. À l'aide des commandes `np.meshgrid` et `plt.contourf`, tracer la fonction f sur $[-3, 3]^2$.

2− On utilise ensuite l'algorithme de Metropolis-Hastings pour simuler la chaîne de Markov.

1. Écrire une fonction `MH(f, X0, sigma, N)` qui simule l'algorithme de Metropolis-Hastings suivant :

— On pose $(x_0, y_0) = X0$.

- Pour $0 \leq n \leq N-1$:
  - Étant donné un point (x_n, y_n), tirer un nouveau point (x_*, y_*) suivant une loi normale centrée en (x_n, y_n) d'écart type σ afin de tirer des points proches du point précédent.
  - Calculer le taux d'acceptation $\alpha = f(x_*, y_*)/f(x_n, y_n)$.
  - Tirer un nombre u dans $[0, 1]$ suivant une loi uniforme.
  - Si $u \leq \alpha$, on pose $(x_{n+1}, y_{n+1}) = (x_*, y_*)$. Sinon, on pose $(x_{n+1}, y_{n+1}) = (x_n, y_n)$.

2. Générer une suite $X_n = (x_n, y_n)$ en appelant la fonction `metropolis_hastings` avec f, $N = 100000$, $\sigma = 0.1$ et un point X_0 tiré aléatoirement de façon uniforme dans $[-3, 3] \times [-3, 3]$.
3. Les premiers points de X_n dépendent du point initial. Dans un subplot, tracer la fonction f avec la commande `plt.contourf` et un histogramme de la suite X_n en enlevant les 10000 premiers termes avec la commande `plt.hist2d`.
