L2 M-MA
2024-2025

Programmation sous Python

Feuille 4 : Simulation en `Python`

Exercice 1. (Pseudo-aléatoire) Dans tout logiciel de calcul numérique, on trouve un générateur de loi uniforme sur $[0,1]$. Plus précisément, avant de simuler des variables aléatoires de lois diverses, il s'agit de savoir générer un hasard 'de base' qui correspond à une suite de variables aléatoires indépendantes de loi uniforme sur $[0,1]$. Celle-ci se fait à partir d'une initialisation réellement aléatoire (basée par exemple sur l'heure de démarrage du programme, la température, etc.) puis par l'intermédiaire d'une suite de nombres 'pseudo-aléatoires', c'est-à-dire une suite déterministe mais ayant un comportement chaotique. Cette simulation ne peut être parfaite : l'ensemble est non-dénombrable mais l'erreur commise est considérée comme négligeable. Sur Python, la génération de la loi uniforme peut se faire à l'aide de la commande `rand()` du module `numpy.random` qu'on nomme généralement `npr`.

1. Construire un échantillon de taille 500 de la loi uniforme sur $[0,1]$ et tracer l'histogramme des fréquences qui lui est associé avec `plt.hist`.
2. On se propose de recréer une suite de nombres pseudo-aléatoires. Écrire une fonction `unif(n, y0)` permettant de générer une suite de n nombres pseudo-aléatoires selon la formule

$$x_n = \frac{y_n}{N} \quad \text{avec} \quad y_{n+1} = (ay_n + b) \bmod N$$

   pour $a = 16807$, $b = 0$, $N = 2^{31} - 1$ et $y_0 > 0$ passé en paramètre. Générer quelques échantillons et tracer les histogrammes associés.
3. Initialisez y_0 comme votre voisin ou voisine et comparez vos simulations. Est-ce vraiment aléatoire ?

Exercice 2. (Paradoxe du duc de Toscane) On regarde le lancer de trois dés. Le duc de Toscane a observé que l'on obtient plus souvent un total de 10 qu'un total de 9. L'aspect paradoxal de l'observation vient du fait que les nombres 9 et 10 se décomposent tous les deux exactement de 6 façons comme somme de 3 entiers entre 1 et 6.

1. Faire des simulations pour mettre en évidence ce phénomène. On cherchera comment générer des variables uniformes à valeurs entières.
2. Écrire une fonction `decompose(n)` qui renvoie la liste de toutes les décompositions d'un entier n comme somme de trois entiers entre 1 et 6. Ainsi pour 9 et 10, la fonction devra rendre 6 possibilités.
3. Quelle réponse apporter au paradoxe ?

Exercice 3. (Pile ou face) On considère le jeu de pile ou face entre deux joueurs A et B. Si A fait 'pile' alors B lui donne 1 euro et si A fait 'face' c'est lui qui donne 1 euro à B. Chacun dispose d'une mise initiale, n_A pour A et n_B pour B, et le jeu s'arrête lorsqu'un joueur est ruiné.

1. Écrire une fonction `ruine(nA, nB)` renvoyant les évolutions $(a_0, a_1, \ldots, a_n)$ et $(b_0, b_1, \ldots, b_n)$ des sommes à disposition de A et de B au cours du jeu, pour lequel on a appelé n le dernier lancer. On cherchera comment simuler des variables de loi de Bernoulli.
2. Écrire une fonction `trajectoire(nA, nB)` permettant de tracer l'évolution de ces deux suites pour une partie donnée. On veillera à générer le graphe sous la forme d'une fonction en escalier.

3. Reprendre les questions précédentes en faisant en sorte que la pièce n'est plus équilibrée mais qu'elle donne 'pile' avec probabilité p et 'face' avec probabilité $1-p$.
4. Simuler un grand nombre de parties et calculer la fréquence avec laquelle A gagne. Comparer avec p.

Exercice 4. (Intégrales) On veut approximer la valeur de

$$I = \int_0^1 f(x)\,\mathrm{d}x \qquad \text{avec} \qquad f(x) = x\,(1-x)\sin^2(200\,x\,(1-x))$$

par une méthode de Monte-Carlo (c'est-à-dire à l'aide de simulations de variables aléatoires).

1. Représenter la fonction f sur $[0,1]$.
2. Expliquer le principe de l'algorithme ci-dessous.

```
import numpy.random as npr
def intMonteCarlo(f, xmin, xmax, n):
    Y = [f(npr.rand()*(xmax - xmin) + xmin) for k in range(n)]
    S = sum(Y)
    I = ((xmax - xmin)/n)*S
    return I
```

3. On a en fait $I \approx 0.080498$ (on pourra tenter de retrouver cette valeur avec la commande `quad` du module `scipy.integrate`). Évaluer la qualité de la méthode sur quelques simulations.
4. Effectuer des tentatives avec d'autres fonctions.

Exercice 5. (Koh-Lanta) Dans cet exercice comme dans le suivant, on déduit par simulation une approximation (plus amusante qu'efficace) de π. On simule ici un tir à l'arc.

1. Commencer par écrire une fonction `unifrectangle(a, b, c, d)` qui renvoie un couple (u_1, u_2) simulé de manière uniforme et indépendante dans $[a,b]\times[c,d]$.
2. Par la suite, on choisit $a = c = -1$ et $b = d = 1$. Écrire une fonction `tracercible()` qui représente graphiquement le carré $[-1,1]\times[-1,1]$ ainsi que le cercle inscrit dans ce carré. Le disque délimité par ce cercle jouera le rôle de la cible. Pour ajuster l'échelle sur les deux axes, on pourra utiliser `plt.axis("scaled")`.
3. On suppose que le tireur à l'arc, qui n'est pas un expert, répartit ses flèches uniformément dans le carré $[-1,1]\times[-1,1]$. Proposer une fonction `simulerfleches(n)` qui simule n tirs et qui affiche sur le graphique uniquement les flèches ayant atteint la cible.
4. En vertu d'un célèbre résultat, la *loi des grands nombres*, on peut affirmer que la proportion de flèches qui atteignent la cible est une bonne approximation de la probabilité qu'à chaque flèche d'atteindre la cible (à condition que le nombre de flèches soit suffisamment grand). Vérifier expérimentalement ce résultat.
5. Pour aller plus loin, écrire une fonction `calculerscore(n)` qui, à l'issue de n flèches, détermine le score obtenu par le tireur (avec une convention à choisir : par exemple 0 à l'extérieur de la cible et ensuite de 1 à 5 selon des anneaux concentriques de même épaisseur, à représenter sur le graphique).

Exercice 6. (Aiguille de Buffon) L'aiguille de Buffon est une expérience statistique proposée dès le XVIIIème siècle par le mathématicien français Buffon. Le concept est assez simple : en répétant un grand nombre de fois une situation aléatoire bien particulière que l'on va décrire, on peut aboutir à une approximation de π. On considère un parquet composé de planches parallèles sur lequel une aiguille est lancée un grand nombre de fois. On compte le nombre de fois où l'aiguille touche au moins deux planches du parquet. Les lames du parquet sont de même largeur $\ell > 0$ et l'aiguille est modélisée par un segment de longueur $0 < a \leqslant \ell$. À chaque lancer, on note $0 \leqslant d \leqslant \frac{\ell}{2}$ la distance du centre de l'aiguille à la rainure la plus proche et $\theta \in [0, \frac{\pi}{2}]$ l'angle formé par l'aiguille avec les rainures, et ces deux quantités sont considérées comme générées uniformément et indépendamment.

1. Faire un schéma pour bien saisir le problème.
2. Montrer que la condition pour que l'aiguille coupe une rainure est $d \leqslant \frac{a}{2}\sin\theta$. Sans entrer dans les détails techniques, on peut montrer que la probabilité qu'une aiguille coupe une rainure vaut $\frac{2a}{\pi\ell}$.
3. Écrire une fonction `tracerparquet(N, l)` qui représente graphiquement un parquet composé de N lames parallèles de largeur ℓ.
4. Écrire une fonction `lanceraiguilles(n, a, N, l)` qui simule le lancer de n aiguilles de longueur a et qui les représente graphiquement sur un parquet de N lames de largeur ℓ. On pourra réutiliser la fonction `unifrectangle` de l'exercice précédent.
5. Étudier l'évolution de la suite $(\pi_n = \frac{2a}{\ell F_n})_{n\geq 1}$ où F_n est la fréquence avec laquelle les n aiguilles ont coupé une rainure du parquet.

Exercice 7. (Régression) Considérons un ensemble de points aléatoires $(x_i, y_i)_{1\leq i\leq n}$ générés selon le schéma

$$\forall\, 1 \leq i \leq n \quad x_i \sim \mathcal{N}(2,2), \quad \varepsilon_i \sim \mathcal{N}(0,1) \quad \text{et} \quad y_i = a + b\,x_i + \varepsilon_i$$

où a et b sont deux paramètres réels. Plus précisément, cela veut dire que [1]

$$\mathbb{P}(x_i \in [\alpha,\beta]) = \int_\alpha^\beta \frac{1}{2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-2}{2}\right)^2}\,\mathrm{d}x \text{ et } \mathbb{P}(\varepsilon_i \in [\alpha,\beta]) = \int_\alpha^\beta \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}\,\mathrm{d}x$$

1. Écrire une fonction `simulernuage(a, b, n)` renvoyant les vecteurs $(x_1,\dots,x_n)$ et $(y_1,\dots,y_n)$. On cherchera comment simuler des variables de loi normale.
2. (plus difficile) Montrer que les valeurs des paramètres qui minimisent la somme des erreurs quadratiques $\sum_{i=1}^n (y_i - a - b\,x_i)^2 = \sum_{i=1}^n \varepsilon_i^2$ (i.e. qui minimisent la fonction $(a,b)\in\mathbb{R}^2 \mapsto \sum_{i=1}^n (y_i - a - b\,x_i)^2 \in \mathbb{R}$) sont données par
$$b^* = \frac{\gamma_{xy}}{v_x^2} \quad \text{et} \quad a^* = m_y - b^*\,m_x$$
où
$$m_x = \frac{1}{n}\sum_{i=1}^n x_i, \quad m_y = \frac{1}{n}\sum_{i=1}^n y_i, \quad v_x^2 = \frac{1}{n}\sum_{i=1}^n (x_i - m_x)^2 \quad \text{et} \quad \gamma_{xy} = \frac{1}{n}\sum_{i=1}^n (x_i - m_x)(y_i - m_y).$$
3. La droite $y = a^* + b^* x$ est appelée *droite de régression*. Écrire une fonction `nuage(a, b, n)` simulant un nuage de n points, calculant l'équation de la droite de régression et effectuant les représentations graphiques associées.

Exercice 8. (Plaque chauffée) On suppose qu'une plaque métallique est initialement à température ambiante (0°C par convention). On choisit aléatoirement n points sources et on les chauffe ou on les refroidit. De manière très simpliste, on suppose que, à l'exception des sources, tout point de la plaque P_{ij} de coordonnées (i,j) subit l'influence des sources P_k de la façon suivante,

$$T_{ij} = \sum_{k=1}^n \frac{T_k}{d(P_k, P_{ij})}$$

où T_k est la température de la k-ème source et $d(P_k, P_{ij})$ la distance entre la source P_k et le point P_{ij}.

1. Écrire une fonction `plaque(l, h, n)` calculant la température à la surface d'une plaque de longueur l et de largeur h, sur laquelle on a choisi aléatoirement n points sources. Chaque point source est chauffé à 20°C ou refroidi à -20°C, là encore aléatoirement. Cette plaque sera vue comme un tableau `numpy` à deux dimensions.
2. Effectuer quelques tests avec les représentations graphiques adéquates (on pourra utiliser `plt.imshow`). On cherchera un colormap adapté aux nuances chaud/froid.

1. Ces intégrales sont difficiles à calculer : on ne peut pas trouver une primitive de $x \mapsto e^{-x^2}$ avec des fonctions « élémentaires »