Université de Bordeaux
L2 Informatique/Maths-Informatique

Probabilités, Statistiques, Combinatoire : TM3

Simulations (3) : Cas générique

Lors des TM précédents, vous avez été amenés à écrire des fonctions de simulation pour des expériences aléatoires plus ou moins complexes ; à chaque fois, l'expérience était décrite en des termes qu'il fallait interpréter et "comprendre" pour écrire la simulation.

Cette semaine, l'objectif est de simuler des expériences très génériques ; la seule hypothèse sera que l'expérience n'a qu'un nombre fini (et, en pratique, pas trop élevé) de résultats possibles. La nature des résultats importera peu : ce seront les clés d'un dictionnaire Python ; et la loi de probabilité sera décrite par les valeurs de ce même dictionnaire.

3.1 Simulation générique d'une loi à support fini

1. Vous avez vu en TD (ex. 4.5) une méthode générale pour simuler une loi de probabilités quelconque sur l'ensemble d'entiers $\{1, 2, \ldots, n\}$, en tirant un nombre aléatoire uniforme dans l'intervalle $0, 1]$ (ce qui, en Python, est fourni par la fonction `random.random()`). Écrivez une fonction `simu_generique(L)`, qui prend en entrée une liste L de nombres (tous positifs ou nuls, et de somme 1) et qui une fonction de simulation (similaire à celles écrites lors de la feuille TM2) qui retourne un entier compris entre 0 et $n-1$ (où n est la longueur de L), chaque entier i étant retourné avec probabilité `L[i]`.
   Conseil : à cause des erreurs d'arrondi inévitables lors des calculs sur nombres en virgule flottante, il est conseillé d'adapter légèrement la méthode : il est possible que la somme (calculée numériquement) des éléments de la liste soit très légèrement inférieure à 1, et que, donc, il existe une toute petite probabilité que le résultat de `random.random()` soit supérieur à cette somme).
2. Testez les fonctions produites par votre fonction sur des lois simples : par exemple avec la liste $[0.25, 0.25, 0.25, 0.25]$, on devrait obtenir des tirages uniformes sur $\{0, 1, 2, 3\}$.
3. Adaptez votre fonction `simu_generique` pour qu'au lieu d'une liste de probabilités, elle prenne un *dictionnaire* de couples (valeur,proba) ; la somme des probabilités devra toujours valoir 1, et la fonction retournée devra, lorsqu'elle est appelée, retourner l'une des clés du dictionnaire, chacune avec probabilité égale à sa valeur associée.
4. La *loi de Poisson* de paramètre x (où x est un réel strictement positif quelconque) est une loi sur les entiers naturels, définie par les probabilités suivantes : (pour tout entier $k \geq 0$) $p_k = e^{-x}x^k/k!$, où $k! = 1 \times 2 \times \cdots \times k$ est la fonction factorielle (avec $0! = 1$ par convention) ; on a donc, pour tout k, $p_{k+1} = xp_k/(k+1)$.
   La loi de Poisson prend une infinité de valeurs, et n'est donc pas simulable directement par la méthode mise en oeuvre dans la fonction `simu_generique` ; on va donc la *tronquer à* n, en posant, pour tout entier $0 \leq k \leq n$, $p'_k = p_k$, et $p'_n = 1-(p'_0+p'_1+\cdots+p'_{n-1})$.
   Utiliser votre fonction `simu_generique` pour simuler cette loi de Poisson tronquée ; vous prendrez des valeurs de x pas trop petites (entre 5 et 10 ; x n'a pas besoin d'être entier), et tronquerez avec une valeur de n d'au moins 20. (Le cas $x = 1$ correspond à la loi "mystère" de la feuille TM1)

5. La complexité (en moyenne) de la fonction de simulation obtenue, dépend de l'ordre dans lequel les valeurs sont traitées par l'algorithme (et donc, si on n'y prend pas garde, de l'ordre dans lequel les clés du dictionnaire sont traitées). Réfléchissez à l'ordre qui devrait être le plus efficace, et modifiez votre fonction `simu_generique` pour en tenir compte. Comparer les résultats par des appels à votre fonction `tirages_repetes` de la feuille TM2.

3.2 Méthode de l'alias de Walker

Une méthode pour simuler des lois discrètes, qui évite le parcours d'une liste, est celle dûe à Walker. Elle nécessite un précalcul, qu'il vous appartiendra de réaliser.

L'idée est la suivante. On souhaite simuler une loi de probabilités sur n valeurs, décrite par une liste $[(v_0, p_0), (v_1, p_1), \ldots, (v_{n-1}, p_{n-1})]$ (l'ordre, ici, importe peu).

Le précalcul consiste à calculer deux tableaux (ou, en Python, deux listes) : un tableau I, contenant des entiers de 0 à $n-1$; et un tableau P, contenant des nombres réels compris entre 0 et 1. Une fois calculés ces deux tableaux, l'algorithme de simulation est le suivant :

— Tirer un entier uniforme J entre 0 et $n-1$.

— Tirer un réel uniforme $U \in [0, 1]$.

— Comparer U à $P[J]$: si $U \leq P[J]$, retourner v_J ; sinon, retourner $v_{I[J]}$.

En d'autres termes, la méthode de Walker remplace le parcours d'une liste par un accès direct dans un tableau.

La difficulté est de comprendre comment se fait le précalcul : quelles valeurs mettre dans les deux tableaux, pour que la loi simulée soit bien celle qui est décrite ? Essayez en inversant le problème : si on se donne les tableaux I et U (ainsi que la liste des valeurs $v_0, \ldots, v_{n-1}$), quelle est la loi du résultat ? Le paragraphe suivant est volontairement elliptique ; il contient seulement des indications pour cet exercice plus difficile que la moyenne.

Pour le calcul (itératif) des tableaux, il est typiquement utile de trier les couples (valeur, proba) par probabilités croissantes. Nécessairement la plus petite probabilité, p_k, est inférieure ou égale à $1/n$, et la plus grande, p_ℓ, est supérieure ou égale à $1/n$; on peut alors fixer $I[k] = \ell$, choisir "comme il faut" $P[k]$, puis continuer en retirant le couple (v_k, p_k) et en diminuant convenablement la probabilité p_ℓ.