TP Informatique 4 : Collection de cartes

1 Introduction

1.1 Avant toute chose...

Dans cette séance de TP, nous nous entraînerons à écrire quelques fonctions pour essayer de répondre à un problème. Lorsque vous écrivez une fonction, le fait que Python ne proteste pas lorsque vous la validez ne doit pas vous satisfaire. **Il est indispensable d'effectuer quelques tests (avec des paramètres vous permettant de contrôler le résultat obtenu) pour vérifier qu'elle fonctionne comme vous l'espérez**.

1.2 Présentation du problème

Bon nombre d'entre vous ont sans doute déjà essayé de compléter une collection de cartes, de vignettes ou de figurines vendues dans des paquets aveugles. Cela donne toujours l'impression que les dernières nous échappent. Sont-elles pour autant vraiment rares, ou bien est-ce simplement un effet du hasard ? C'est ce que nous allons chercher à savoir aujourd'hui.

Nous supposerons que l'on cherche à compléter une collection comprenant au total $N = 100$ cartes, vendues par paquets aveugles de $p = 3$ cartes. On suppose qu'il n'y a pas de cartes plus rares que d'autres, et que l'assortiment est totalement aléatoire.

1. Définir deux noms `p` et `N`, que l'on associera respectivement à 3 et 100, afin de pouvoir s'en servir dans la suite du problème.

`p` et `N` seront utilisés comme noms *globaux*. **Ce n'est en principe pas une bonne pratique**, mais cela évitera que l'on ait à passer n et `P` comme arguments d'un grand nombre de fonctions dans la suite, ce qui vous simplifiera quelque peu la tâche.

Dans la suite, on supposera que chacune des cartes de la collection est représentée par un entier entre `0` et `N-1`, soit entre 0 et 99.

1.3 Mise en place

La fonction `randint(a, b)` du module `random` retourne un entier aléatoire entre `a` (inclus) et `b` (inclus), `a` et `b` devant être eux-même des entiers. Chacun des entiers de l'intervale a une probabilité identique d'apparaître (soit $\mathscr{P} = \frac{1}{b+1-a}$).

2.a Ecrire une fonction `Paquet()` ne prenant aucun argument et retournant une liste de `p` entiers tirés aléatoirement entre 0 et $N-1$.

En fait, l'éditeur garantit à ses acheteurs que dans le paquet ne se trouveront jamais deux cartes identiques.

2.b Ecrire une fonction `PaquetSansDoublon()` qui renvoie elle aussi une liste de p entiers tirés aléatoirement entre 0 et $N-1$, mais ne contenant aucun doublon (cela correspond à un tirage *sans remise*). On supposera évidemment $p \leq N$.

Note : la fonction `sample` du module `random` est particulièrement adaptée pour cet usage, et vous pouvez essayer de l'utiliser grâce au cours ou à l'aide en ligne de la fonction, mais essayez également d'écrire la fonction `PaquetSansDoublon` *en vous abstenant* utiliser la fonction `sample`, c'est un bon exercice.

2 Simulation d'un remplissage d'album

2.1 Principe

On souhaite simuler le remplissage d'un album par quelqu'un achetant des paquets de p cartes un à un. Pour représenter les cartes que possède notre collectionneur, on construira une liste `L` contenant `N` entiers, chacun d'eux indiquant le nombre d'exemplaires possédé pour chacune des cartes. Ainsi, `L[i]` correspond au nombre d'exemplaires de la carte portant le numéro i.

1. Avant l'achat du tout premier paquet, que doit contenir `L` ? Créer une telle liste `L`.

2.2 Simulation du remplissage

2. Ecrire une fonction `AjoutePaquet(L, listeCartes)` qui prend en argument une liste `L` correspondant à une collection, et une liste `listeCartes` correspondant à un ensemble de cartes (telle une liste produite par `Paquet` ou `PaquetSansDoublon`), et met à jour la liste `L`. La fonction n'a pas besoin de renvoyer quoi que ce soit, elle effectuera directement les modifications dans la liste désignée par `L`.

3. Ecrire une fonction `Compte(L)` qui retourne le nombre de cartes *différentes* possédées.

4.a Écrire une fonction `Simule(nbPaquets, foncGen)` qui prend en argument un nombre `nbPaquets` de paquets achetés et une fonction `foncGen` retournant des paquets générés aléatoirement, et qui retourne une liste contenant `nbPaquets+1` éléments, telle que le `i`-ème élément de cette liste indique le nombre de cartes possédées après l'achat du i-ème paquet. `Simule` doit initialiser puis utiliser sa *propre* liste `L` (locale à la fonction) pour que l'utilisateur n'ait pas à initialiser lui-même une liste `L` avant d'appeler `Simule`.

4.b Utiliser la fonction précédente pour construire une liste `P` pour `nbPaquets=200` paquets achetés, en utilisant la fonction de génération de paquets sans doublons `PaquetsSansDoublons`.

4.c À l'aide de la fonction `plot` du module `matplotlib.pyplot`, tracer la courbe correspondante illustrant l'évolution de la collection.

5.a Écrire une fonction `Necessaire(P, nbCartes)` qui indique après combien d'ouvertures de paquets on a possédé plus de `nbCartes` dans la simulation dont le résultat est représenté par la liste `P`.

5.b Après combien de paquets achetés a-t-on 50% des cartes ? 90% des cartes ? 95% des cartes ? Rappelons que l'achat de 200 paquets représente 600 cartes !

5.c Recommencer plusieurs fois la simulation afin de voir si ces valeurs changent beaucoup ou non.

2.3 Effet des probabilités d'assortiment

6.a Créer une liste `P2` correspondant à un achat de `nbPaquets=200` paquets, mais en supposant qu'il peut y avoir des doublons dans le paquet.

6.b Tracer les deux courbes sur le même graphique. Y a-t-il une différence notable sur la constitution de la collection ?

En fait, dans la collection à laquelle s'intéresse notre collectionneur, les cartes dans le paquet ne sont pas équiprobables. Sur les trois cartes, une carte fait partie des cartes numérotées de `0` à `N//5-1` (soit de 0 à 19), les deux autres sont deux cartes différentes parmi celles numérotées `N//5` à `N-1` (soit de 20 à 99). À l'intérieur de chacun des deux groupes, la répartition des cartes est en revanche équiprobable.

7.a Créer une fonction `PaquetSpecial()` qui génère un paquet aléatoire de trois cartes avec la règle précédente.

7.b Quelle influence cela a-t-il sur la constitution de la collection ?

3 Collaboration pour le remplissage

3.1 Principe

On suppose à présent que k personnes collaborent pour tenter de compléter k exemplaires de la collection (chacun voudra la sienne !). Pour ce faire, à chaque étape, *chacun* des participants achète un paquet de cartes. Toutes les cartes sont mises en commun.

Une liste `L` contiendra le nombre de cartes possédées de chaque exemplaire, comme précédemment, mais pour l'ensemble du groupes de k personnes.

3.2 Simulation

1. Ecrire une fonction `Compte_bis(L, k)` qui détermine le nombre *moyen* de cartes différentes possédées par chacun des k participants. Attention, il peut toujours y avoir des cartes dont personne n'a besoin (lorsque le groupe possède plus de k exemplaires d'une même carte), mais il faut à présent obtenir plusieurs exemplaires de chacune des cartes.

2.a Écrire une fonction `SimuleGroupe(nbAchats, k, foncGen)` qui prend en argument le nombre `nbAchats` de fois où k paquets seront achetés, le nombre k de participants, et une fonction `foncGen` générant les paquets, et qui retourne une liste contenant `nbAchats+1` éléments, telle que le i-ème élément de la liste corresponde au nombre moyen de cartes possédées par les k participants après le i-ème achat de k paquets.

2.b Utiliser cette fonction pour construire une liste `P5` indiquant, sur le même principe que `P`, le nombre moyen `P5[i]` *moyen* de cartes différentes possédées par chacun des participants après l'achat de $i \times k$ paquets, dans le cas où $k = 5$. On utilisera la fonction `PaquetSansDoublon` pour générer les différents paquets.

2.c Comparer graphiquement l'évolution de `P[i]` et `P5[i]`. Est-il avantageux de « travailler » en groupe ?

4 Méthode de Monte-Carlo

Dans la suite, on revient au cas d'une personne seule cherchant à compléter sa collection sans procéder à des échanges. Les simulations que l'on a fait dans ce cadre ne fournissent qu'une possibilité parmi de nombreuses autres quant à la progression de la collection. On souhaiterait obtenir une progression « moyenne » qui ne soit pas influencée par une succession d'événement favorables ou défavorables dans la simulation. Bref, obtenir le nombre *moyen* de cartes possédées après l'ouverture de i paquets.

Il est possible d'en faire une étude théorique, ce que nous ferons dans la section suivante, mais dans un premier temps, nous allons utiliser une autre approche, du type méthode de *Monte-Carlo*. Ces méthodes consistent à effectuer un grand nombre d'expériences aléatoires (le nom même de ces méthodes vient des casinos du rocher monégasque), et de moyenner les résultats ainsi obtenus. On peut montrer que, sous certaines conditions, la moyenne converge bien vers ce que l'on souhaite.

1.a Proposer une fonction `MonteCarlo(nbexp, nbpaquets, funcGen)` qui effectue `nbexp` simulations (en utilisant la fonction `Simule`) et retourne une liste de `nbpaquets+1` éléments où chacun des éléments représente la moyenne des éléments à la même position dans chacune des listes retournées par les différents appels à `Simule`.

Par exemple, si `nbexp=4` et `nbpaquets=4`, les quatre appels successifs à `Simule` ont pu retourner successivement les listes `[0, 3, 5, 8, 11]`, `[0, 3, 6, 9, 12]`, `[0, 3, 6, 7, 10]` et `[0, 3, 5, 8, 10]`. Le résultat que l'on cherche sera alors `[0.0, 3.0, 5.5, 8.0, 10.75]`.

1.b Tracer, sur un même graphique, le résultat d'une simulation normale effectuée avec `Simule` et le résultat de la méthode de Monte-Carlo effectuée avec 100 expériences.

1.c À partir de combien d'expériences dans la méthode de Monte-Carlo la courbe obtenu vous semble suffisamment « lisse » ?

5 Calcul exact

En fait, il est parfaitement possible d'obtenir le nombre de cartes possédées en moyenne après l'ouverture de i paquets en travaillant avec des probabilités. Dans la suite, on se place toujours dans le cadre d'un unique collectionneur tentant de compléter sa collection sans échanges, et on supposera que les paquets contiennent $p = 3$ cartes *différentes*, chaque carte apparaissant avec la même probabilité.

On utilisera une liste `Prob` contenant N + 1 = 101 éléments, où chaque élément de la liste, `Prob[i]`, désigne la probabilité, à une étape donnée, de posséder i cartes dans sa collection. Chacun des termes sera donc compris entre 0 et 1, et la somme de tous les termes de la liste devrait être égal à 1 à tout moment.

1. Que doit contenir la liste `Prob` avant l'ouverture du premier paquet ? La construire.

Si toutes les cartes sont équiprobables et que les paquets ne contiennent pas de doublons, on peut montrer que la probabilité de découvrir k cartes nouvelles, lorsque l'on en possède déjà q et que l'on ouvre un paquet de p cartes (le nombre total de cartes différentes étant N) peut s'écrire

$$\mathscr{P}(k,q,p,\mathrm{N}) = \frac{\binom{n-q}{k} \times \binom{q}{p-k}}{\binom{n}{p}} \quad \text{où} \quad \binom{b}{a} = \frac{b!}{a!(b-a)!}$$

La fonction `factorial(n)` du module `math` permet de calculer la factorielle de l'entier n. La formule précédente n'est pas la méthode la plus efficace de calculer $\binom{b}{a}$ (que l'on appelle *combinaison de a parmi b*), mais on s'en contentera aujourd'hui.

2.a Écrire une fonction `Probabilite(k, q)` qui donne la probabilité d'obtenir k nouvelles cartes en ouvrant un paquet lorsque l'on en possède déjà q différentes avant son ouverture (on a toujours $p = 3$ et N = 100).

2.b Quelle est la probabilité d'obtenir 3 nouvelles cartes lorsque l'on n'en possède aucune ? Pourquoi ?

2.c Qu'en est-il de la probabilité d'obtenir 1 nouvelle carte lorsque l'on en possède 99 ?

3.a Écrire une fonction `AjoutePaquetProb(Prob)` qui prend en argument la liste des probabilités à une étape, et met à jour cette liste de sorte que les probabilités correspondent à l'étape suivante, après ouverture d'un paquet supplémentaire (il n'est pas besoin de renvoyer de résultat, puisque l'on effectue des mutations de la liste `Prob`).

3.b Vérifier le bon fonctionnement de cette fonction en l'utilisant pour construire les listes de probabilité des trois premières étapes (on vérifiera tout particulièrement qu'après i étapes la probabilité de posséder plus de $p \times i = 3 \times i$ cartes est nulle, et que la somme des probabilités à une étape donnée est toujours égale à 1, aux erreurs d'arrondi près).

Pour obtenir le nombre moyen de cartes, il faut calculer l'*espérance* du nombre de cartes associé à un ensemble de probabilité. Ici, cette espérance se calcule très facilement. si $\mathscr{P}(q)$ est la probabilité de posséder q cartes, alors l'espérance $\mathscr{E}$ (soit le nombre moyen de cartes possédés à cette étape) se calcule par la relation :

$$\mathscr{E} = \sum_{q=0}^{\mathrm{N}} q \times \mathscr{P}(q)$$

4. Écrire une fonction `CompteProb(Prob)` qui prend en argument une liste de probabilités `Prob` et retourne le nombre moyen de cartes possédées.

5.a Écrire une fonction `SimuleProb(nb)` qui effectue un calcul de l'évolution de l'espérance du nombre de cartes possédées lorsque l'on ouvre `np` paquets un à un, et retourne une liste de `nb+1` éléments correspondant à cette évolution.

5.b Comparer ce résultat, en le traçant, à celui obtenu par la méthode de Monte-Carlo.