Université de Bordeaux
L2 Informatique/Maths-Informatique

Probabilités, Statistiques, Combinatoire : TM2

Simulations(2)

Lors du premier TM, vous avez écrit deux types de fonctions : certaines pour réaliser une expérience aléatoire et retourner son résultat (jusqu'ici, toujours sous la forme d'un nombre entier) ; et d'autres, pour simuler de manière répétée une expérience et retourner un dictionnaire qui recense les fréquences observées des différents résultats de cette expérience.

Il n'a pas du vous échapper que les différentes fonctions de tirage répété étaient extrêmement similaires : idéalement, seule la ligne de code qui réalise l'expérience change d'une fois sur l'autre. Il est donc pertinent de chercher à factoriser ce code : c'est l'un des objectifs de cette séance.

1. Écrire une fonction `tiragesRepetes(modele,n)`, générique, qui retourne un dictionnaire des fréquences de résultats observés pour n tirages. Le premier paramètre, `modele`, est censé être une fonction (sans paramètres !) qui effectue la simulation de l'expérience et en retourne le résultat ; pour la feuille **TM1**, les fonctions `premierSix()` et `sommeDeuxDes()` que vous avez déjà écrites devraient être utilisables directement comme `modele`.
2. Pour le tirage d'un dé classique (cubique, à 6 faces, équilibré), on a utilisé `random.randint(1,6)` ; pour le tirage d'un hypothétique dé à d faces équiprobables (y compris les cas géométriquement douteux, comme $d = 7$), on peut naturellement utiliser `random.randint(1,d)`.
   Écrire une fonction `genere_de(d)`, qui retourne **une fonction** (ne prenant aucun paramètre) qui simule le tirage d'un dé équilibré à d faces. Les commandes suivantes devraient alors fonctionner pour produire un dictionnaire de tirages d'un dé à 12 faces :

   ```
   d12 = genere_de(12)
   dict = tiragesRepetes(d12,1000)
   ```

   (et, bien sûr, `d12()` devrait également pouvoir servir à simuler le lancer d'un dé à 12 faces)
   Deux mécanismes sont utilisables pour faire retourner une fonction par une fonction : définir une fonction (par `def toto():`) dans le corps de `genere_de`, et retourner cette fonction (`return toto`) ; ou l'utilisation d'une fonction anonyme, par le mot-clé Python `lambda` (attention, le corps d'une fonction `lambda` doit être réduit à une seule expression : en particulier, il n'est pas possible d'y exécuter un `while`). Vous utiliserez ce qui vous semble le plus pertinent.
3. À partir de cette construction de "dés à d faces", construire également des simulateurs pour les expériences "composites" suivantes :
   — tirage de k dés à d faces, retournant la somme des valeurs des dés ;
   — tirage répété d'un dé à d faces, jusqu'à obtenir une valeur donnée v ($1 \leq v \leq d$) ; retournant le nombre de tirages effectués ;
   — tirage de k dés à d faces, retournant le k-uplet de valeurs. **Attention :** il est important que la fonction retourne un tuple et non une liste, pour que la valeur en question puisse être utilisée comme clé dans un dictionnaire par la fonction `tiragesRepetes` ; utiliser `tuple(L)` pour obtenir un tuple à partir d'une liste.

À chaque fois, l'objectif est d'écrire une fonction (fonctionnelle), qui prend en entrée les paramètres (d, k, $v\ldots$) et retourne la fonction de simulation qui, elle sera utilisable comme paramètre de `tiragesRepetes`.

4. **Cas des variables aléatoires rélles :** lorsque les valeurs prises sont des nombres réels, on peut les ordonner et décrire, à partir d'un échantillon de n tirages (synthétisé par nos dictionnaires de fréquences), une *fonction empirique de répartition*, qui est mathématiquement définie ainsi : la valeur de cette fonction en x, est *la proportion des tirages qui ont donné une valeur inférieure ou égale à x*. On se propose de représenter de telles fonctions empiriques, de nouveau, par des dictionnaires. Plus précisément, à partir d'un dictionnaire `dico` de fréquences (tel que produit par votre fonction `tiragesRepetes`), un algorithme possible est le suivant pour produire un nouveau dictionnaire :
   — Former la liste des clés du dictionnaire (`dico.keys()`) ;
   — trier cette liste ;
   — calculer, pour chaque clé, la *somme* des fréquences (valeurs associées) des clés qui lui sont inférieures (clé elle-même comprise) ;
   — dans le nouveau dictionnaire, associer à chaque clé, cette somme.

   Par exemple, si le dictionnaire de départ est {`1:0.2, 3:0.1, 5:0.3, 4:0.1, 6:0.3`}, le résultat devrait être {`1:0.2, 3:0.3, 4:0.4, 5:0.7, 6:1.`}.

   Écrire une fonction `frequencesVersDistribution(dict)` qui réalise cette opération. Les clés du nouveau dictionnaire devraient être exacttement les mêmes que celles de l'ancien, c'est-à-dire les valeurs obtenues lors des tirages (cela peut être un sous-ensemble strict des valeurs possibles de la variable aléatoire simulée).

5. **Visualisation :** dans le fichier `annexe_tm1.py` fourni, vous trouverez une fonction `dessiner_fonctions_de_repartition`, qui permet de visualiser graphiquement [1] les fonctions de répartition (ou même les dictionnaires de fréquences) obtenues. **Attention :** le premier paramètre de cette fonction est censé être une **liste** de dictionnaires ; même pour visualiser une seule fonction de répartition, il faut lui passer une liste.

   Utiliser cette fonction pour visualiser, sur le même graphique, plusieurs fonctions de répartition empiriques, pour des échantillons de tailles différentes (par exemple : 10, 100, 1000, 100000) de la même expérience.

   Pour les expériences dont vous savez le faire, trouver un moyen d'avoir aussi, sur le même graphique, la vraie fonction de distribution (non empirique) de votre expérience.

1. À condition que le module `tkinter` soit installé ; c'est le cas au CREMI, c'est moins automatique si vous travaillez sur votre propre machine