Université de Bordeaux
L2 Informatique/Maths-Informatique

Probabilités, Statistiques, Combinatoire : TM3

Simulations génériques

Lors de la séance précédente, vous avez écrit deux types de fonctions pour simuler des expériences aléatoires : des fonctions de *simulation*, et des fonctions de "test" qui avaient pour objectif d'observer le résultat d'un grand nombre d'appels d'une fonction de simulation.

Il n'a pas dû vous échapper que les fonctions de test se ressemblaient beaucoup : typiquement, seul le code réalisant l'appel à une fonction de simulation changeait de l'une à l'autre. Vous allez cette semaine factoriser votre code, et écrire une fonction de test unique, prenant en entrée la fonction de simulation à utiliser.

4.1 Simulation de lois de probabilités

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 **TM3**, les fonctions `premierSix` et `sommeDeuxDes` que vous avez déjà écrites devraient être utilisables directement comme `modele`.
   Attention : le premier paramètre de `tiragesRepetes` est une *fonction*, pas le résultat d'un appel au simulateur ! on pourra donc utiliser

   ```
   dic = tiragesRepetes(premierSix,1000)
   ```

   mais un appel de la forme

   ```
   dic = tiragesRepetes(premierSix(),1000)
   ```

   devrait provoquer des erreurs bien méritées à l'exécution.
2. Jusqu'à présent, vous avez écrit vos fonctions de simulation une par une : une fonction pour simuler le lancer d'un dé, une fonction pour simuler la somme des lancers de deux dés... et si d'un coup on souhaite changer le modèle de dés simulé (dé à 8 faces, dé à 20 faces), il faudra réécrire tous les simulateurs.
   Il est, ici aussi, souhaitable d'éviter la duplication de code, en écrivant non pas directement les simulateurs, mais en écrivant des fonctions qui seront des "générateurs de simulateurs" : ces fonctions retourneront des *fonctions* (sans paramètres), qui pourront, elles, directement être utilisées comme modèles par `tiragesRepetes`.
   Ainsi, au lieu d'écrire :

   ```
   def simuleDe12():
     ...

   def simuleSommeDeuxDes12():
     return simuleDe12()+simuleDe12()

   dic = tiragesRepetes(simuleSommeDeuxDes12,1000)
   ```

on pourra écrire :

```
monDe12 = genereDe(12)
sommeDeuxDes12 = sommeTiragesIndependants(monDe12,2)
dic = tiragesRepetes(sommeDeuxDes12,1000)
```

Écrire une fonction `genereDe(d)`, qui retourne **un simulateur** de dé à d faces. **Indication :** pour faire retourner une fonction par une fonction PYTHON, vous pouvez définir une fonction (`def f():`) à l'intérieur d'une autre fonction, et retourer cette fonction (`return f`).

3. Écrire également des générateurs composites permettant de simuler simplement des expériences de plus en plus complexes :
   (a) `genereBernoulli(p)` : retourne un simulateur de variable de Bernoulli de paramètre p (**Indication :** la fonction `random.random()` retourne un réel compris entre 0 et 1 ; il a probabilité p d'être inférieur à p) ;
   (b) `genereGeometrique(p)` : retourne un simulateur de variable géométrique de paramètre p ;
   (c) `sommeTiragesIndependants(modele,n)` : retourne un simulateur de la somme de n tirages indépendants selon le modèle passé en paramètre ;
   (d) `genereBinomiale(n,p)` : retourne un simulateur de la loi binomiale de paramètres (n, p) ;
   (e) `genereSomme(modele1,modele2)` : retourne un simulateur pour la somme de deux variables aléatoires indépendantes, dont les simulateurs sont passés en paramètres ;

   Avec vos fonctions, il devrait être élémentaire (en une ligne) d'obtenir un dictionnaire des fréquences de 10000 tirages de la somme de trois dés à 6 faces, ou un dictionnaire de fréquences pour des tirages de la somme de deux dés à 6 faces et d'un dé à 10 faces.
4. Les constructions précédentes ne permettent pas d'obtenir simplement des simulateurs pour des variables aléatoires qui n'ont pas de représentation simple en termes d'expériences de variables aléatoires indépendantes "plus simples", comme la loi de Poisson (une telle représentation existe, mais c'est un peu compliqué à voir).

   Vous allez donc écrire une fonction qui retourne un *simulateur* à partir d'un dictionnaire, `genereSimu(dict)`. Comme pour les dictionnaires de fréquences que vous avez produits jusqu'ici, les clés du dictionnaire seront les valeurs prises par la variable aléatoire simulée, et les valeurs associées seront les probabilités respectives des valeurs.

   L'algorithme de simulation est le suivant : on fait un (et un seul !) tirage d'un réel x entre 0 et 1, avec la fonction `random.random()` ; puis, on passe en revue (dans un ordre quelconque – on pourrait chercher à optimiser l'ordre, mais c'est une autre question) les clés :
   — si x est inférieur à la probabilité de la première clé, on retourne la première clé ;
   — si x est compris entre la probabilité de la première clé, et la somme des probabilités des deux premières clés, on retourne la deuxième clé ;
   — si x est compris entre la somme des probabilités des deux premières clés, et la somme des probabilités des trois premières, on retourne la troisième clé ;
   — plus généralement, si x est compris entre la somme des probabilités des k premières clés, et la somme des probabilités des $k + 1$ premières, on retournera la $k + 1$-ème clé.

5. Pour simuler les lois de Poisson, il reste une difficulté : ces lois prennent n'importe quelle valeur entière, il faudrait donc préparer un dictionnaire avec une infinité de clés... Vous allez contourner le problème en préparant un dictionnaire approché, en utilisant la propriété suivante (vue en TD7) : pour une variable aléatoire X suivant une loi de Poisson de paramètre x, la suite des probabilités $(\mathbb{P}(X = k))_{k\geq 0}$ est croissante jusqu'à $k = \lfloor x \rfloor$, et décroissante au-delà de $k = \lceil x \rceil$. Vous calculerez donc un dictionnaire pour la "loi de Poisson tronquée" : toutes les valeurs au-delà d'un certain k_{max} seront remplacées par k_{max}, avec la probabilité correspondante. L'indice k_{max} pourra être pris comme le plus petit entier $k > x$ tel que la probabilité $\mathbb{P}(X \geq k)$ soit inférieure à une limite suffisamment petite, par exemple 10^{-8}.

   Vous écrirez donc une fonction `genere Poisson(x)`, qui retourne un simulateur (approché) pour la loi de Poisson de paramètre x. **Indication :** la fonction exponentielle est `math.exp` ; vous aurez besoin d'un `import math` au début de votre fichier. Pour calculer les probabilités, mieux vaut procéder de proche en proche : si on note p_k la probabilité que la variable de Poisson prenne la valeur k, on a $p_0 = e^{-x}$, et une relation simple (à trouver !) permet de calculer p_k à partir de k et de p_{k-1}, sans autre appel à la fonction exponentielle ni calcul direct de factorielle.

4.2 Expériences autour de la loi de Poisson

Une des raisons pour lesquelles la loi de Poisson est très utilisée en modélisation, est la suivante (qu'il n'est pas question de démontrer dans ce cours) : Si on a un grand nombre de variables aléatoires de Bernoulli indépendantes, toutes de petit paramètres (mais la somme des paramètres, elle, peut ne pas être petite), alors leur somme "ressemble beaucoup" à une variable de Poisson dont le paramètre serait la somme des paramètres des Bernoulli.

Vous pouvez en faire l'expérience avec des variables binomiales (équivalentes à des sommes de Bernoulli) : faites simuler, par exemple, des binomiales de paramètres $(1000, 0.01)$ et des Poisson de paramètre 10, et comparez les dictionnaires ; ou des binomiales de paramètres $(1000, 0, 005)$ et des Poisson de paramètre 5...

Une autre expérience facile à réaliser est la suivante : une propriété des lois de Poisson (TD 7) est que la somme de deux variables de Poisson indépendantes suit également une loi de Poisson (dont le paramètre est la somme des paramètres). Vous pouvez facilement réaliser des dictionnaires de fréquences pour la somme de deux Poisson et les comparer à une Poisson du paramètre correspondant.