Université de Bordeaux
L2 Informatique/Maths-Informatique

Probabilités, Statistiques, Combinatoire : TM1

Simulations

Les TP machine s'appuient sur le langage `python` (a priori `python3`). L'objectif de cette première séance est de mettre en place des méthodes de simulation.

Pour la génération aléatoire, on utilise le module `random` chargé avec la commande `import random`.

La première primitive utile est `random.randint(a,b)`, un générateur aléatoire qui renvoie un nombre entier choisi au hasard dans $\{a, a+1, \dots b\}$.

Pour quelques fonctions mathématique usuelles, on utilisera le module `math`.

Pour manipuler les fréquences empiriques d'apparition d'événements, nous allons utiliser une structure de données de *dictionnaire*, fournie par le langage `python`.

Le dictionnaire est un conteneur – il n'y a pas d'ordre sur les données. Celles-ci sont organisées en couples de forme `clé : valeur`, par exemple {`('id' : 'Dupont', 'mdp' : 'badPassword')`}.

Les crochets délimitent les listes, les parenthèses délimitent les tuples, pour délimiter les dictionnaires on utilise les accolades {}.

Pour initialiser un dictionnaire vide, il suffit de dire `monDict = {}`.

Pour ajouter une entrée à un dictionnaire, on utilise la clé comme indice :

```
monDict['id'] = 'Dupont'
```

Si la clé n'existe pas encore dans le dictionnaire, elle est ajoutée au dictionnaire avec la valeur spécifié après le signe `=`. Sinon, l'ancienne valeur est remplacée par la nouvelle.

Pour accéder à une valeur précise, c'est très simple :

```
id = monDict['id']
```

affectera `'Dupont'` à la variable `id`. Si la clé n'existe pas dans le dictionnaire, une exception de type `KeyError` sera levée.

Pour tester si une clé appartient au dictionnaire, et aussi pour parcourir toutes les entrées du dictionnaire, on utilise le mot clé `in` comme pour les listes :

```
'clef' in monDict
```

vaut `False` ;

```
for clef in monDict:
   print(monDict[clef])
```

affiche `'Dupont'`.

Simulation de lois de probabilités

Pour simuler les lancers de dé, on va utiliser `random.randint(1,6)`.

1. Écrire une fonction `lancers(n)` qui retourne un nouveau dictionnaire contenant les *fréquences* d'apparition des valeurs de 1 à 6 après `n` lancers d'un dé, calculées avec l'algorithme suivant :
   — Initialiser un dictionnaire vide.

— Pour chaque lancer, si le résultat est déjà dans le dictionnaire, incrémenter la valeur associée ; sinon, l'initialiser à 1.
— Avant de retourner, normaliser, c-à-d diviser tous les nombres d'occurrences par `n` pour obtenir des valeurs dont la somme soit 1.

La valeur retournée de `lancers(5)` pourrait donc ressembler à { `1 : 0.2, 2 : 0.4, 3 : 0.2, 4 : 0.2, 5 : 0, 6 : 0.2`} (ici les clés sont les résultats possibles de l'expérience, et les valeurs, les fréquences observées, sous formes de flottants)

2. Tester votre fonction `lancer(n)` avec des valeurs de `n` $= 10, 100, 1000, 1000000$, chacune à plusieurs reprises. Commenter les résultats obtenus, notamment en ayant en tête la vision "fréquentiste" des probabilités.
3. Écrire une fonction `premierSix()` qui simule le lancer répété d'un dé et retourne le nombre total de lancers nécessaires pour obtenir le résultat 6 pour la première fois.
4. Écrire une fonction `tirageRepetePremierSix(n)` comparable à la fonction `lancers()`, mais pour l'expérience consitant à attendre le premier six ; comme pour la fonction `lancers()`, expérimenter avec des valeurs de `n` de 10, 100, 1000 et 1000000 et commenter les résultats obtenus.
5. Faire la même chose avec l'expérience correspondant à lancer deux dés, et à prendre la somme des deux dés : écrire une fonction `sommeDeuxDes()` qui simule cette opération, et une fonction `tirageRepeteSommeDeuxDes(n)` qui retourne un dictionnaire contenant les fréquences observées des différents résultats possibles pour n tirages.
6. On va maintenant utiliser une autre primitive, `random.random()`, qui retourne un résultat flottant entre 0 et 1 ; de manière approchée, pour $0 \leq a < b \leq 1$, la probabilité que `random.random()` retourne un résultat compris entre a et b est $b - a$ (oublions que l'intervalle $[0, 1]$ est un ensemble non dénombrable).

   On se propose de tirer un nombre entier aléatoire selon la méthode suivante : pour un nombre flottant p, compris entre 0 et 1, on tire de manière répétée des nombres avec `random.random()`, et on calcule leur *produit* ; et on retourne le nombre de tirages nécessaires pour atteindre un produit inférieur à p.

   Écrire une fonction `tirageMystere(p)` qui simule cette expérience, et une fonction `tirageRepeteMystere(n,p)` qui réalise n fois cette expérience et retourne un dictionnaire des fréquences observées. Expérimenter avec votre fonction `tirageRepeteMystere()`, et proposer, pour $p = e^{-1} \simeq 0.3679$, des valeurs approchées pour les probabilités que `tirageMystere(p)` retourne $1, 2, 3, \ldots$. **Indication :** pour identifier expérimentalement ces probabilités, on pourra essayer de multiplier les fréquences observées par $e \simeq 2.718$