Des bases de données... et des probas

Vendredi 4 mars 2016

Buts du TP

– Pratiquer un peu à nouveau des requêtes SQL.
– Faire des simulations de processus aléatoires.

Exercice 1. *Créer (au bon endroit) un dossier associé à ce TP. Dans ce dossier, placer les fichiers suivants, récupérés sur le réseau :*

`squelette.py` `communes.db` `nobel.db` `foot.db`

Le fichier `squelette.py` contient ce qu'il faut pour que vous puissiez avancer vite dans le TP, sans vous préoccuper des tâches annexes et chronophages.

Exercice 2. *Lancer Spyder (ou Pyzo), sauvegarder immédiatement le fichier édité au bon endroit sous un nom pertinent. Écrire une commande absurde, de type* `print(6*7)` *dans l'éditeur ; sauvegarder.*
Ceux qui travaillent sous Spyder pourront, avec F6, imposer les directives « exécuter dans un nouvel interpréteur dédié » et « interagir avec l'interpréteur Python après exécution ».
Exécuter le fichier de script. Ouvrir également le fichier `squelette.py` *et copier/coller (au fur et à mesure du TP le contenu).*

1 Neuf requêtes

Les différentes bases fournies contiennent comme dans la vraie vie quelques erreurs plus ou moins grosses : des problèmes d'encodage, des doublons ; il y est même question de prix Nobel d'économie, qui n'existent pas !
Pour chaque base, on commencera par écrire sur un papier le schéma relationnel (au moins partiel).

1.1 Communes, départements et régions

Exercice 3. *Construire des commandes SQL permettant de déterminer :*

1. *Le nom, la population en 2010, la surface et l'altitude maximale des communes dont la surface est supérieure ou égale à* $100\,km^2$ *et l'altitude maximale inférieure à* $20\,m$.
2. *La table constituée des noms des régions et leur nombre de départements.* On mettra également le champ ayant permis de réaliser le regroupement après jointure, et on classera cette table par ordre décroissant du nombre de départements.
3. *La table constituée des noms des régions possédant au moins 2000 communes.*

1.2 Prix Nobel

Exercice 4. *Construire des commandes SQL permettant de déterminer :*

1. *Le nom des différents prix ainsi que le nombre de lauréats (pour chacun des 5 véritables prix Nobel ; il faut donc exclure ceux étiquetés « Economie »).*
2. *Les lauréats cités au moins deux fois dans la table (mais en enlevant les doublons de la table* `prix`*).*
3. *Les lauréats nommés dans deux domaines distincts.*

1.3 Champions league virtuelle (optionnel)

Exercice 5. *Construire des commandes SQL permettant de déterminer :*

1. *La table des 185 rencontres dans les années 80 où une équipe espagnole jouait à domicile.*
2. *La table des matchs sous la forme d'un quintuplet contenant l'année, le nom des deux clubs et leurs scores. Exemple d'entrée :*
   `1971,Olympique Lyonnais,0,Real Madrid,1`
3. *La table des 91 matchs où une équipe française à gagné contre une équipe anglaise... en Angleterre.*

2 Cinq simulations

Exercice 6. *Importer la sous-librairie* `random` *de la librairie* `numpy`*. Tester le comportement de* `randint`*.*

```
import numpy.random as rd

>>> rd.randint(0, 3, 10)
...
```

2.1 Dés honnêtes jetés honnêtement

On lance un dé non pipé à six faces un certain nombre de fois. La moyenne des lancers est censée [1] tendre vers $\frac{7}{2}$ (espérance d'une variable suivant une loi uniforme sur $[\![1,6]\!]$).
On va évaluer la moyenne sur de grandes séries de lancers, ainsi que la moyenne du carré de l'écart à la moyenne (en fait, l'écart à l'espérance, ce qui est légèrement diffèrent). Enfin, on va évaluer des fréquences heuristiques de l'événement $\left|\frac{S_N}{N} - m\right| > \varepsilon$, avec S_N la somme sur N lancers : cette fréquence est censée être petite (et est contrôlée par Bienaymé-Tchebychev).

Exercice 7. *Écrire une fonction prenant en entrée un entier $N \geqslant 1$, réalisant N lancers de dés et renvoyant la moyenne des lancers.*

```
>>> moyenne_lancers(1000)
3.4209999999999998
```

Exercice 8. *Écrire une fonction prenant en entrée un entier $N \geqslant 1$, réalisant N lancers de dés $(X_1, ..., X_N)$ et renvoyant la moyenne des $(X_i - 7/2)^2$.*
L'espérance de cet écart est $\sigma^2 = \frac{n^2-1}{12} = \frac{6^2-1}{12} \simeq 2,917$.

```
>>> ecart_moyenne(1000)
2.9079999999999999
```

Exercice 9. *Écrire une fonction prenant en entrée un entier $N \geqslant 1$ (la longueur de chaque série de lancers), un autre entier N_{exp} (le nombre de séries de lancers), un flottant $\varepsilon > 0$, réalisant N_{exp} séries de N lancers, et renvoyant la fréquence des séries pour lesquelles la moyenne s'est écartée de $7/2$ de (strictement) plus de ε.*

```
>>> frequence_eloignement(1000, 100, 0.1)
0.04
```

Ici, sur 100 séries de 1000 lancers, 4 ont eu une moyenne en dehors de l'intervalle $[3.4, 3.6]$.

2.2 Triche avec un dé honnête

Ici, le dé reste non pipé, mais celui qui lance le dé va tricher : si le résultat du premier lancer est inférieur ou égal à seuil fixé, alors il relance le dé (il ne peut le faire qu'une seule fois) et garde ce deuxième résultat. Par exemple, si le seuil vaut 4 :
– si le premier tirage est $\geqslant 5$ alors c'est ce résultat qu'il retient ;
– sinon, il lance le dé une deuxième fois et retient ce résultat (même s'il est inférieur au premier tirage).

Exercice 10. *Écrire une fonction prenant en entrée le seuil et simulant l'expérience décrite plus haut.*

Dans le script cadeau, on fournit quelques lignes de code permettant de tester cette fonction en réalisant 10^5 lancers pour chaque seuil de $[\![0,6]\!]$. Les résultats affichés sont les fréquences des différents résultats des expériences, avec la moyenne en fin de ligne.

```
0.16 0.17 0.17 0.17 0.17 0.17 -> 3.51
0.03 0.20 0.19 0.19 0.20 0.19 -> 3.91
0.05 0.06 0.22 0.22 0.22 0.22 -> 4.17
0.08 0.08 0.08 0.25 0.25 0.25 -> 4.26
0.11 0.11 0.11 0.11 0.28 0.28 -> 4.16
0.14 0.14 0.14 0.14 0.14 0.31 -> 3.91
0.17 0.17 0.17 0.17 0.17 0.17 -> 3.50
```

La théorie (exo que certains ont peut-être eu en colle...) dit que l'espérance est maximale lorsque le seuil vaut 3.

1. Plus précisément : la loi faible des grands nombres dit que la probabilité d'être en dehors de $[m - \varepsilon, m + \varepsilon]$ tend vers 0. La loi forte dit que la probabilité de tendre vers m vaut 1.

2.3 Problème d'ascenseur

Un ascenseur prend en charge N_p personnes ; chacun choisit un étage parmi N_e possibles. Combien de fois l'ascenseur va-t-il s'arrêter ?

Exercice 11. *Écrire une fonction réalisant cette expérience ! En entrée, on prendra N_p et N_e, on créera un tableau de booléens (indexé de 0 à N_e pour ne pas s'embêter) indiquant en position p si un arrêt est prévu à l'étage p. Chaque personne met à* `True` *un étage au hasard, et à la fin on compte le nombre de* `True`*, qu'on renvoie.*

Le script donné en cadeau réalise 10^5 fois l'expérience pour 60 personnes et 50 étages (et donne la *moyenne* du nombre d'arrêts) ; on calcule ensuite la valeur de l'*espérance* du nombre d'arrêts.

```
Moyenne pour 100000 expériences, 60 personnes et 50 étages : 35.126

>>> 50 * ( 1 - (1-1/50.)**60 )
35.12234286539398
```

2.4 Permutations

Parmi les permutations de $[\![1, n]\!]$, le nombre de points fixes oscille entre 0 et n, et vaut 1 en moyenne (au sens de l'espérance). La proportion de dérangements (permutations sans points fixes) vaut par ailleurs $\sum_{k=0}^{n} \frac{(-1)^k}{k!} \underset{n\to+\infty}{\longrightarrow} \mathrm{e}^{-1}$.

Exercice 12. *Écrire une fonction prenant en entrée une permutation σ et calculant le nombre de points fixes.*
La permutation σ sera une liste de n entiers, qui doivent être (inutile de le vérifier) les éléments de $[\![0, n-1]\!]$.

```
>>> points_fixes([1, 0, 2, 4, 3])
1
```

Exercice 13. *Écrire une fonction déterminant si une permutation est ou non un dérangement.*
On n'utilisera pas la fonction précédente : on doit cesser de parcourir la liste dès qu'on trouve un point fixe.

Le script fourni en cadeau génère aléatoirement des permutations de $[\![0, 9]\!]$ (fonction `numpy.random.permutation`) et fait quelques statistiques...

```
Nombre moyen de points fixe de S[10] (100000 expériences) : 0.99949

Fréquence de dérangements dans S[10] (100000 expériences) : 0.36735
>>> sum((-1)**n/float(factorial(n)) for n in range(11)), exp(-1)
(0.3678794642857144, 0.36787944117144233)
```

2.5 Ivrogne dans Manhattan

Un ivrogne fait des pas aléatoires parmi quatre possibles : Est/Ouest/Nord/Sud. Après n pas, il est clair que l'espérance de sa position est... la position initiale. Il n'est pas très difficile de montrer que l'espérance du carré de sa distance à l'origine vaut n... et que l'espérance de la distance est inférieure [2] à $\sqrt{n}$.

Exercice 14. *Écrire une fonction réalisant une marche de n pas (n donné en paramètre) et renvoyant la position finale (l'initiale étant $(0, 0)$).*

```
>>> marche(100)
(-15, 7)
```

On fournit quelques lignes permettant de faire quelques stats :

```
>>> stats(100, 1000)
[-0.186, -0.056, 8.868129147550723, 100.116]
>>> stats(1000, 1000)
[-0.042, 0.452, 27.712843503787838, 989.044]
>>> stats(10000, 1000)
[-0.321, 1.775, 87.42391766286032, 9671.998]
```

Pour 1000 expériences de $n \in \{10^2, 10^3, 10^4\}$, on regarde les abscisses et ordonnées moyennes à l'arrivée, ainsi que la distance (et la distance au carré) moyenne à l'issue des marches.

2. Si f est concave (typiquement : si $f'' \leqslant 0$...), alors $\mathbb{E}(f(X)) \leqslant f(\mathbb{E}(X))$. Appliquer ceci à $t \mapsto \sqrt{t}$ et $X = \left(\sum_{k=1}^{n} X_i\right)^2 + \left(\sum_{k=1}^{n} Y_i\right)^2$.