Psi 999 – 2021/2022
http://www.psi945.fr

TP Python

HO ! HO ! HO !

Vendredi 10 et samedi 11 décembre 2021

Buts du TP

Tester la procédure de tirage de noms pour le « secret Santa ». Constater que certaines propriétés souhaitables... ne sont pas vérifiées ! Essayer de réparer cela.

Exercice 1. *Créer (au bon endroit) un dossier associé à ce TP. Lancer Spyder/Pyzo/Idle, sauvegarder au bon endroit le fichier* `tdsantaouuntruccommeça.py` *; écrire une commande absurde, de type* `print( 6*7 )` *dans l'éditeur, sauvegarder et exécuter.*

Exercice 2. *Vérifier que le premier exo a effectivement bien été fait : tout manquement donnera lieu de ma part à une agitation néfaste pour tout le monde...*

1 Permutations et points fixes

On parle ici, pour des raisons techniques, de permutations (bijections) de $E = [\![0, n-1]\!]$. Il s'agit formellement de bijections de E dans lui-même, mais elles peuvent être vues comme des « tableaux/listes/tuples » indexé(e)s de 0 à $n-1$. Par exemple l'identité sera représentée au choix par la liste `[0, 1, 2]` ou par le triplet `(0, 1, 2)` voire le tableau `array([0, 1, 2])` alors que la permutation échangeant 0 et 2 et fixant 1 sera représentée au choix par `[2, 1, 0]`, `(2, 1, 0)` ou `array([2, 1, 0])`.
Dans un cas comme dans l'autre (liste/tableaux/tuple) les indices vont de 0 à $n-1$, et on peut compter les points fixes en cherchant les indices i tels que $t[i] = i$.

Exercice 3. *Écrire une fonction calculant le nombre de points fixes d'une permutation.*

```
>>> pts_fixes([2, 1, 0]), pts_fixes( (0, 2, 1, 3) )
(1, 2)
```

Il est possible de parcourir l'ensemble des $n!$ permutations d'un ensemble (liste) à n éléments grâce à la fonction `permutations` de la bibliothèque `itertools` :

```
>>> permutations( [0, 1] )
<itertools.permutations at 0x7f0e7dbfd950>

>>> list(permutations( [0, 1] ))
[(0, 1), (1, 0)]

>>> for p in permutations( [0, 1] ):
        print(p)
(0, 1)
(1, 0)
```

Combien une permutation de $[\![0, n-1]\!]$ peut-elle avoir de points fixes (des i tels que $p[i] = i$) ? Entre 0 et n certes... mais en moyenne ?

Exercice 4. *Écrire une fonction prenant en entrée un entier $n \in \mathbb{N}^*$ et calculant le nombre moyen de points fixes pour les $n!$ permutations de $[\![0, n-1]\!]$*

```
>>> pts_fixe_exhaustif(5)
1.0
```

Rappel : `list(range(5))` est une commande fort intéressante. Vous disposez par ailleurs de la fonction `factorial` dans la bibliothèque `math`

Pour $n = 20$ il est hors de question de lister les $20! > 10^{18}$ permutations. On peut donc échantillonner, en prenant N permutations aléatoires et en faisant la moyenne sur ces N expériences. De telles permutations aléatoires peuvent être construites à la main (voir plus loin), ou bien en utilisant la fonction `permutation` de la bibliothèque `numpy.random` :

```
>>> permutation([0, 1, 2, 3])
array([1, 0, 3, 2])

>>> permutation([0, 1, 2, 3])
array([3, 1, 2, 0])

>>> permutation([3, 5, 42, 999])
array([  5, 999,  42,   3])

>>> permutation(6)
array([4, 1, 0, 3, 2, 5])
```

Exercice 5. *Écrire une fonction estimant le nombre de points fixes par échantillonage en utilisant ces permutations aléatoires « clés en main ».*

```
>>> pts_fixe_echantillonnage1(5, 10**6)
1.000303

>>> pts_fixe_echantillonnage1(20, 10**6)
0.998938
```

Voici une façon de créer une permutation aléatoire : on part d'une liste de valeurs à choisir (les entiers entre 0 et $n-1$), puis on les place les unes après les autres dans une nouvelle liste (initialement vide). À chaque étape (disons pour i allant de 0 à $n-1$), on choisit une valeur de la liste des $n-i$ éléments qui restent, on l'enlève de la première liste et on le place dans la seconde :

```
def perm(n):
    L = list(range(n))
    P = []
    for i in range(n):
        x = L[randint(n-i)]
        P.append(x)
        L.remove(x)
    return P
```

Exercice 6. *Comprendre précisément le code précédent, l'écrire, et l'utiliser pour estimer le nombre moyen de points fixes d'une permutation.*

```
>>> perm(5)
[2, 3, 1, 0, 4]

>>> pts_fixe_echantillonnage2(5, 10**6)
0.999583

>>> pts_fixe_echantillonnage2(20, 10**5)
0.99591
```

2 Comment éviter les points fixes ?

Pour « secret Santa », la procédure de tirage des noms ressemble à la fonction écrite plus haut, à ceci près que quand quelqu'un tire son propre nom ($t[i] = i$), il le replace dans l'urne et recommence... jusqu'à

ce qu'il tire un autre nom. Bien entendu il est possible que le dernier nom qui reste.. soit celui du dernier qui tire un nom ! Avec quelle probabilité d'ailleurs ? On pourrait être tenté de répondre : $1/n$... mais c'est plus subtil : en fait les précédents n'ont pas sélectionné leur nom (quitte à retirer)... donc la probabilité pour que le dernier numéro restant soit $n-1$ est légèrement inférieure à $1/n$.

Exercice 7. *Écrire une fonction réalisant une permutation aléatoire en introduisant la règle : « si je prends mon numéro (et que je ne suis pas le dernier) alors je le replace dans l'urne puis je retire un autre numéro ».* Vous pouvez vous contenter d'adapter la fonction créant une permutation aléatoire !

```
>>> regle2(3)
[1, 2, 0]

>>> regle2(3)
>>> [1, 0, 2] # FAIL !!!
```

On vient de voir qu'effectivement on peut avoir un problème au dernier tirage. Avec quelle probabilité ?

Exercice 8. *Écrire une fonction estimant la probabilité pour que le dernier numéro choisi soit un point fixe (tirer de nombreuses permutations aléatoires, et compter celles pour lesquelles le dernier indice est point fixe).*

```
>>> 1/37
0.02702702702702703

>>> fail(37, 10**4)
0.0239  # léger biais

>>> fail(37, 10**5)
0.02415 # encore un biais suspect

>>> fail(37, 10**6)
0.024162 # et encore. La probabilité est strictement plus petite que 1/n
```

On peut éviter le problème du « point fixe ultime » en échangeant si nécessaire les deux derniers tirages (dans le cas où $t[n-1] = n-1$).

Exercice 9. *Écrire une fonction mettant en place cette nouvelle règle (en modifiant la fonction précédente). Pour tester le caractère uniforme des permutations, on peut évaluer (par échantillonage) la probabilité pour que $\sigma(i) = j$: c'est normalement $1/(n-1)$. Écrire une fonction réalisant ce travail.*

```
>>> regle3(5)
[4, 0, 1, 2, 3]

>>> 1/36
0.027777777777777776

>>> test(37, 0, 1, 10**5)
0.02726

>>> test(37, 0, 1, 10**6)
0.027618

>>> test(37, 35, 36, 10**5)
0.0499

>>> test(37, 10, 20, 10**5)
0.02733
```

On peut enfin décider que si quelqu'un tire son nom, alors on reprend le tirage depuis le début : ça fera un peu grogner les premiers participants, mais le cours de probabilités nous dit que nous n'aurons normalement pas à recommencer... trop de fois !

Exercice 10. *Écrire une fonction fournissant un tirage aléatoire sans point fixe à l'aide du générateur* `permutation`*, qu'on appelle tant qu'il nous donne une permutation avec au moins un point fixe ! Tester comme à l'exercice précédent le caractère uniforme des tirages sur quelques couples.*

```
>>> regle4(5)
array([1, 2, 4, 0, 3])

>>> test2(37, 35, 36, 10**5)
0.02761

>>> test2(37, 35, 36, 10**6)
0.027588
```

Joyeuses fêtes à tous !