Psi 999 – 2021/2022 ψ **TP Python**

Des probabilités

Vendredi 1er et samedi 2 avril 2022

Buts du TP

— Simuler des expériences probabilistes.
— Visionner les résultats expérimentaux.
— Calculer avec `numpy` les résultats théoriques.

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

Si vous êtes sous Spyder, changez (via F6) les options d'exécution, pour avoir à chaque exécution une nouvelle console et garder la main dessus. (il y a donc deux cases à vérifier/cocher).

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...*

La fonction qui sera la plus utilisée est `randint`, de la bibliothèque `numpy.random`. Elle fournit sur demande un nombre entier aléatoire entre les deux bornes (première incluse, deuxième exclue, attention !) qu'on lui donne comme paramètres. On utilisera également dans la dernière partie la fonction `random` de la bibliothèque `numpy.random` (non, ce n'est pas une typo...) qui retourne un flottant aléatoire entre 0 et 1.

1 Paradoxe de Penney

On s'intéresse ici à une série infinie de tirages à pile ou face, en s'arrêtant dès qu'on rencontre la séquence PFP ou bien FFP. Pour un tirage aléatoire de 3 pièces, chaque séquence a la même probabilité de présence (1/8), mais l'arrivée de **première** occurrence est asymétrique. On va le voir :
— par la simulation, en réalisant 10^6 séries qu'on espère pas trop infinies !
— en appliquant les résultats théoriques qui nous disent quelle matrice évaluer à une bonne puissance... et comment récupérer les probabilités de victoire de PFP vs. FFP à partir de cette matrice.

Pour la simulation, on tire au sort les trois premiers lancers. Ensuite (et tant que le motif en cours n'est ni PFP ni FFP), on tient à jour en permanence les trois derniers lancers. On adopte par exemple la convention : P=0, F=1.

Exercice 3. *Écrire une fonction ne prenant pas d'argument, et retournant* `True` *si le premier motif rencontré (parmi PFP et FFP) est PFP ; et* `False` *sinon.*
Cette fonction initialise donc puis tient à jour trois variables entières, représentent à chaque instant les trois derniers tirages. À chaque étape on remplace (a, b, c) par (b, c, d) avec d le nouveau tirage.
Dans l'exemple suivant, on a réalisé 10 expériences et et PFP est sorti avant FFP quatre fois.

```
>>> [penney() for _ in range(10)]
[True, True, True, False, False, False, False, False, False, True]
```

Exercice 4. *Effectuer* 10^6 *expériences ; quelle sont les fréquences de victoire des deux candidats ?*

L'approche « chaîne de Markov » de ce problème consiste à représenter l'évolution du tirage par une promenade sur le graphe suivant (les sommets décrivent les deux derniers lancers, les arêtes décrivent les probabilités de mouvement à chaque étape) :

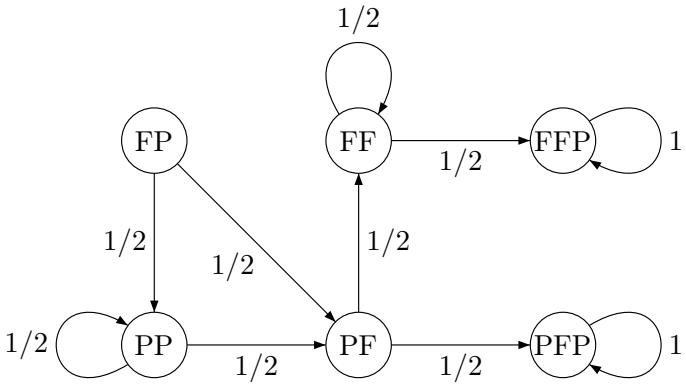

FIGURE 1 – Une chaîne de Markov

La théorie dit qu'avec probabilité 1, on va se retrouver dans l'un des états PFP ou FFP. Elle précise même un peu les choses : les probabilités $(p_i^{(n)})_{1\leqslant i\leqslant 6}$ de présence dans les différents états après $2+n$ lancers sont données par

$$L_n = \begin{pmatrix} p_1^{(n)} & \cdots & p_6^{(n)} \end{pmatrix} = \begin{pmatrix} 1/4 & 1/4 & 0 & 1/4 & 1/4 & 0 \end{pmatrix} \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^n = L_0 A^n$$

(les états sont numérotés selon l'ordre [PP,PF,PFP,FF,FP,FFP])

Exercice 5. *Calculer $L_0 A^{2^{20}}$ en réalisant 20 mises au carré successives de A (on utilisera la méthode `dot` des tableaux numpy pour faire des produits matriciels). Comparer avec les simulations.*

Une analyse plus fine de la situation permet en fait de calculer explicitement la probabilité de terminer en PFP : il y a une chance sur deux si l'état initial (l'une des quatre situations équiprobables après les deux premiers tirages) était FP, PP ou PF... En fait, une résolution de système linéaire permet de calculer de façon automatique ces espérances... ainsi que la longueur moyenne (l'espérance) des séries de lancers.

2 Filles vs. garçon

On teste ici la politique nataliste « on s'arrête de faire des enfants dès qu'on a un fils », et on s'intéresse en particulier à l'espérance de la proportion de garçon par fratrie.

Exercice 6. *Écrire une fonction réalisant une série de naissances et retournant le premier rang où un garçon est né.*

```
>>> [premier_fils() for _ in range(10)]
[4, 3, 2, 1, 1, 2, 2, 4, 1, 1]
```

Théoriquement, le nombre moyen (l'espérance du nombre) de filles est de 1, et la proportion moyenne (son espérance) de fils est de $\ln 2$.

Exercice 7. *Sur un million de couples pratiquant cette politique nataliste, évaluer la moyenne du nombre de filles et la moyenne de la proportion de garçon par couple.*

3 Problèmes à l'embarquement

On va voir ici deux scénarios indépendants, pour le remplissage d'un avion avec 100 passagers :

— La compagnie a vendu 104 places, et chaque passager a une probabilité $p = 0,95$ de se présenter à l'embarquement. Est-ce qu'il y aura un passager sans place (auquel cas la compagnie va lui rembourser 3 fois son billet, ce qui va lui couter de l'argent) ?
— Sur 100 passagers ayant une place numérotée, le premier a oublié son numéro, donc se place n'importe où. Ensuite, les passagers entrent, et prennent leur place normale si elle est libre, et une autre libre au hasard sinon. On veut savoir si le numéro 100 aura sa place.

Exercice 8. *Écrire une fonction simulant un embarquement avec ou sans surbooking : on réalise* 104 *tirage d'un réel entre* 0 *et* 1. *Si ce nombre est plus grand que* $0,95$ *alors le passager ne se présente pas. On retourne le nombre de passagers qui n'auront pas de places dans l'avion.*

```
>>> [surbooking(100, 104, 0.95) for _ in range(20)]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 1, 1, 0, 4, 0]
```

À quelle fréquence doit-on s'attendre à avoir des problèmes ?

Exercice 9. *Réaliser* 10^4 *embarquements, et comptabiliser le nombre de passagers en surbooking.*

Dans l'exemple suivant, il y a eu 54 embarquements avec 4 passagers en surbooking, mais 7728 fois la compagnie a pu vendre 4 tickets supplémentaires sans qu'il n'y ait de surbooking.

```
>>> test_surbook(10**4)
[7728, 1243, 735, 240, 54]
```

Pour le dernier exercice – difficile – on pourra utiliser la méthode `remove` des listes Python, qui fait ce qu'on imagine.

Exercice 10. *Écrire une fonction simulant l'embarquement de n passagers (avec le deuxième scénario décrit plus haut), et qui renvoie la place que devra finalement prendre le dernier passager.*

```
>>> [A380(100) for _ in range(10)]
[0, 99, 0, 0, 0, 99, 99, 99, 99, 0]
```

(J'ai choisi de numéroter les passagers de 0 à $n - 1$).