Tennis et Python

Samedi 5 mars 2022

Buts du TP

- Écrire des simulations de processus stochastiques.
- Représenter des résultats numériques avec `matplotlib`
- Utiliser `numpy` pour calculer sur de grosses matrices.
- Constater que le théorie et la simulation ont l'un et l'autre leur intérêt !

Exercice 1. *Créer (au bon endroit) un dossier associé à ce TP. Y placer une copie du fichier récupéré dans le dossier partagé de travail de la classe :* `cadeau_tennis.py`
Lancer Spyder/Pyzo/Idle, sauvegarder au bon endroit le fichier `tp_tennis_ouuntruccommeça.py` *; écrire une commande absurde, de type* `print( 6*7 )` *dans l'éditeur, sauvegarder et exécuter. En profiter pour importer ce qu'il faut : voir l'entête du fichier-cadeau.*

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). Si vous êtes sous Pyzo, la commande CTRL SHIFT S lui expliquera qu'il est prié de sauvegarder les fichiers dans le répertoire courant.

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...*

Pour gagner un match de tennis, on doit remporter un certain nombre de sets : en général 3, et exceptionnellement 5 (tournois masculins du grand chelem). Pour gagner un set il faut remporter un certain nombre de jeux (détails plus loin), et pour remporter un jeu il faut... remporter au moins 4 «échanges», avec deux échanges d'avance.

1 Gain d'un jeu

Le code suivant fournit un programme prenant un réel $p \in [0,1]$ en paramètre (la probabilité pour qu'Alice remporte un échange), et renvoie `True` avec probabilité p : on peut le voir comme une expérience de Bernoulli.

```
def point(p):  # p est la probabilité pour qu'Alice gagne le point
    return random() < p
```

Il s'agit maintenant de simuler un jeu : on réalise des échanges grâce à la fonction précédente, en tenant à jour les scores d'Alice et de Bob. Dès que l'un des deux atteint 4 (boucle `while` portant sur le maximum), on passe en mode «attente que l'un des deux ait au moins 2 points d'avance sur l'autre» (boucle while portant sur la valeur absolue de la différence entre les deux scores). On peut alors renvoyer le résultat (points d'Alice et de Bob).

Exercice 3. *Écrire une fonction réalisant une telle simulation.*

```
>>> [jeu(0.5) for _ in range(8)]
[(4, 0), (4, 2), (2, 4), (7, 5), (2, 4), (6, 4), (2, 4), (2, 4)]
```

Pour visualiser les résultats, on fournit un code (à copier depuis le fichier cadeau) permettant de réaliser (pour certaines valeurs de p), un certain nombre de jeux, et d'observer les fréquences avec lesquelles Alice a gagné le jeu.
On commence par écrire une fonction réalisant des statistiques sur un grand nombre de jeux (y compris sur le nombre d'échanges, ce dont on ne fera rien dans un premier temps).

```
def stats_jeux(p, nbjeux):  # victoires d'Alice et nombre de points
    va, nbpts = 0, 0
    for _ in range(nbjeux):
        pa, pb = jeu(p)
        nbpts += pa+pb
        if pa > pb:
            va += 1
    return va/nbjeux, nbpts/nbjeux

>>> stats_jeux(0.6, 10**4)
(0.7469, 6.4657)
```

Exercice 4. *Copier/coller le code, comprendre son fonctionnement, exécuter, et observer les résultats.* Il faut VRAIMENT que vous ayez compris à la fin ce que sont `les_p` et `les_pj`...

```
pypl.grid()
les_p = linspace(0, 1, 51)
les_pj = [stats_jeux(p, 10**2)[0] for p in les_p]
pypl.plot(les_p, les_pj, '-+', label='10**2 expériences')
les_pj = [stats_jeux(p, 10**3)[0] for p in les_p]
pypl.plot(les_p, les_pj, '-+', label='10**3 expériences')
pypl.xlabel('$p$', fontsize='x-large')
pypl.ylabel('$p_j$', fontsize='x-large')
pypl.legend()
pypl.savefig('pj_simul.pdf')
```

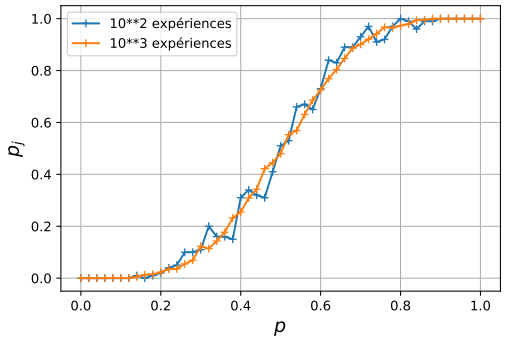
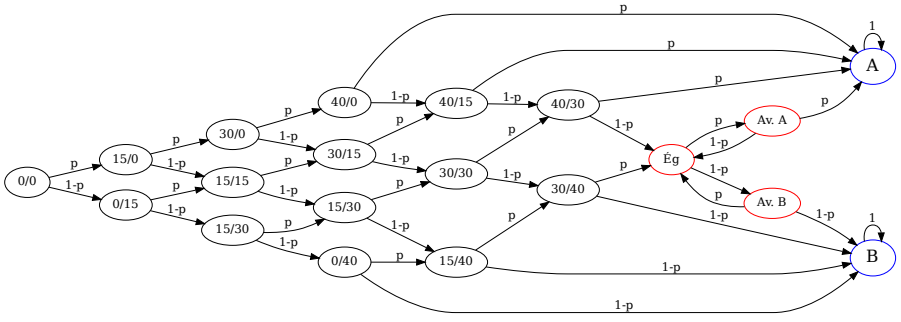


FIGURE 1 – Probabilité expérimentale pour qu'Alice remporte un jeu

On peut attaquer le problème d'un point de vue mathématique, en voyant le déroulement d'un jeu comme une *chaîne de Markov* :



Les sommets/états de ce graphe sont numérotés de 0 à 19, les deux derniers désignant les états A (Alice a gagné le jeu ; les échanges futurs ne changent rien, d'où la transition un peu artificielle de A vers A) et B.
Si M est la matrice décrivant les probabilités de transition d'un état à l'autre, alors $(M^k)_{i,j}$ est la probabilité de passer de l'état i à l'état j en k étapes.

En particulier, et sachant que l'état initial est numéroté 0, on obtient une bonne estimation de la probabilité de victoire d'Alice en prenant $(M^{10^4})_{0,18}$. Cette matrice est un peu pénible à écrire : elle vous est fournie, ainsi que son utilisation : on pourra observer le code fourni.

```
>>> proba_jeu(0.6)
0.7357292307692307
```

Exercice 5. *Ajouter au graphe précédent la valeur théorique de gain d'un jeu par Alice*

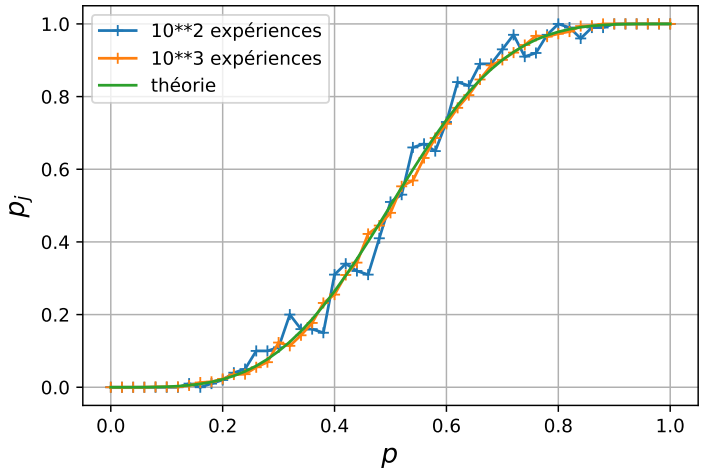


FIGURE 2 – Probabilité expérimentale et théorique pour qu'Alice remporte un jeu

2 Tie-break

On gagne un set quand on a gagné 6 jeux, avec 2 d'avance, mais attention : à (6,6) on va jouer un «tie-break», qui est un jeu spécial, donc le vainqueur gagne le set. Le gagnant du tie-break est le premier à atteindre 7 points avec 2 d'avance. Des scores de tie-break possibles sont donc : $(7,4), (5,7), (12,10)$... Finalement, le score d'un set (nombre de jeux gagnés par Alice et Bob) pourra donc être $(4,6), (6,2), (7,5), (6,7)$ (dans le dernier cas, Bob a gagné le tie-break).

Exercice 6. *Écrire une fonction* `tiebreak` *et adapter le script graphique des jeux pour visualiser les résultats.*

```
>>> [tiebreak(0.5) for _ in range(8)]
[(4, 7), (7, 3), (2, 7), (7, 2), (7, 2), (7, 9), (7, 3), (7, 2)]

>>> stats_tb(0.6, 10**4)
(0.7831, 11.1782)
```

On fournit également la matrice permettant de calculer la valeur théorique de la probabilité pour qu'Alice remporte un tie-break : elle consiste à travailler avec une chaîne de Markov à 53 états (oui, comptez bien...).

Exercice 7. *Écrire un programme calculant la probabilité de gain d'un tie-break par Alice (toujours en fonction de sa probabilité de gain d'un échange).*

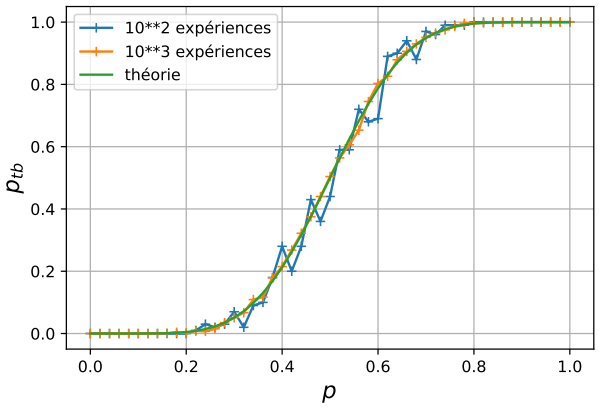


FIGURE 3 – Probabilité expérimentale et théorique pour qu'Alice remporte un tie-break

3 Set puis match

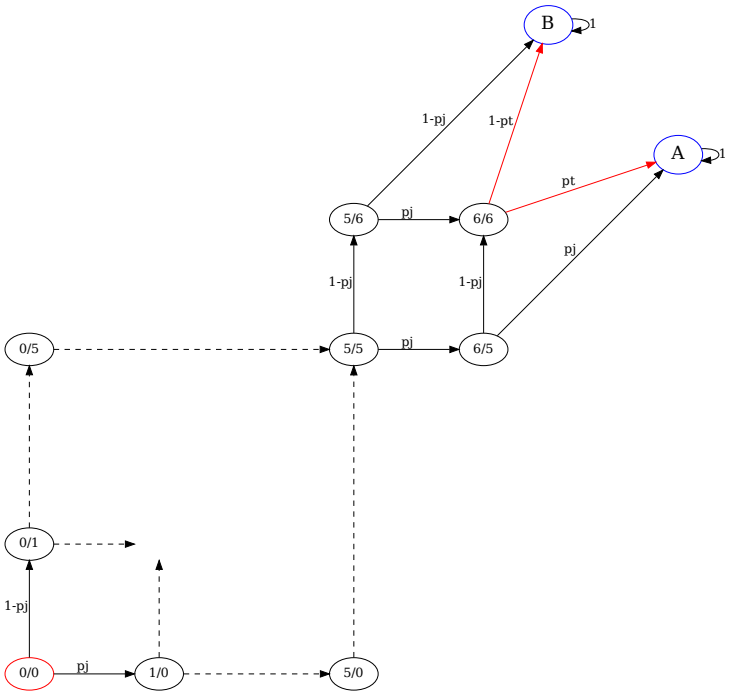
Exercice 8. *Avec les règles rappelées plus haut, simuler des sets.*

Dans les appels suivants, les statistiques donnent le nombre moyen de jeux qui ont été joués par set. Par exemple si Alice gagne 60% des échanges, 10^4 sets ont été joués et elle en a gagné plus de 96%.

```
>>> unset(0.5) for _ in range(8)]
[(6, 2), (6, 4), (3, 6), (6, 4), (3, 6), (6, 1), (3, 6), (4, 6)]


>>> stats_sets(0.6, 10**4)
(0.9627, 8.1433)
```

Les résultats théoriques (fournis en cadeau) sont calculés à l'aide de la chaîne de Markov suivante :



Exercice 9. *Représenter les résultats expérimentaux et théoriques pour gagner un set.*

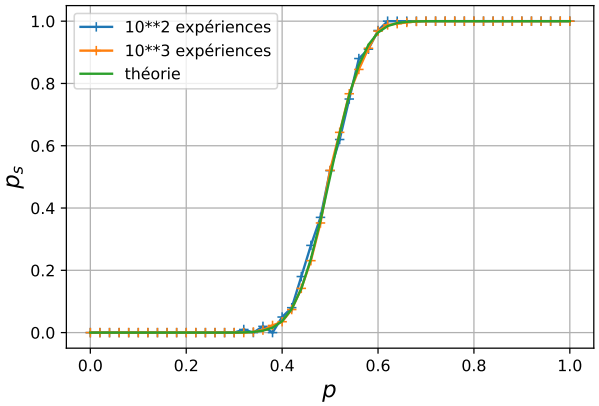
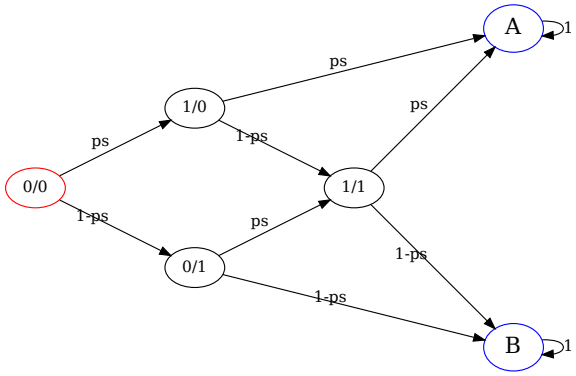
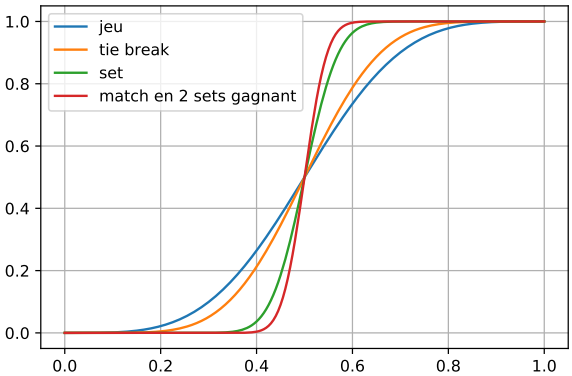


FIGURE 4 – Probabilité expérimentale et théorique pour qu'Alice remporte un set

Exercice 10. *Assez de simulation ! Calculer la probabilité théorique pour qu'Alice gagne le match, grâce à la chaîne de Markov suivante :*



Exercice 11. *Représenter finalement sur un même graphique les probabilités théoriques pour qu'Alice remporte respectivement un jeu, un tie-break, un set puis un match en deux sets gagnants.*



4 Extensions possibles

Voici quelque points supplémentaires sur lesquels vous pouvez travailler, si ce sujet vous a plu et que vous voulez y passer encore plus de temps !

- Implémenter (simulation et calcul théorique) les matchs en 5 sets.
- Attention, le dernier set d'un match ne comporte pas de tie-break, et on continue de faire des jeux normaux jusqu'à ce qu'un joueur en ait gagné (au moins 6 et) deux de plus que son opposant : vous pouvez implémenter cette règle !
- Vous pouvez simuler des jeux/tie-break/set/matchs pour évaluer le nombre moyen d'échanges. Vous constaterez que l'intuition se révèle exacte : plus p est éloigné de $1/2$ et moins il y a d'échanges. Je vous laisse réfléchir au calcul théorique avec la matrice de la chaîne de Markov...
- On peut aussi adapter