Ecole Polytechnique, Cycle Ingénieur (2A)
MAP471A - Problem solving en mathématiques appliquées
Enseignants : Lucas Gerin (mail) (mailto:lucas.gerin@polytechnique.edu) Teddy Pichard (mail) (mailto:teddy.pichard@polytechnique.edu) Nicole Spillane (mail) (mailto:nicole.spillane@polytechnique.edu)

```
# css style
from IPython.core.display import HTML
def css_styling():
    styles = open("./style/custom2.css").read()
    return HTML(styles)
css_styling()
```

```
# load the libraries
import matplotlib.pyplot as plt # 2D plotting library
import numpy as np              # package for scientific computing
import random
%matplotlib inline
```

Apprentissage par renforcement

Table des matières

L'objectif de ce TP est d'étudier de façon expérimentale deux problèmes simples qui se traitent naturellement par *Apprentissage par renforcement*.

1. Le problème du bandit

Considérons le problème suivant : un annonceur a le choix d'afficher sur une page web une publicité choisie parmi $\{A, B\}$, l'annonceur est payé au clic et l'objectif est d'afficher la publicité la plus attractive.

On modélise le problème de la façon suivante : chaque utilisateur se comporte de façon indépendante des autres et clique sur la publicité A (resp. B) avec probabilité p_A (resp. p_B), on suppose que p_A, p_B sont inconnues.

On note $E_i \in \{A, B\}$ la publicité affichée sur le site lorsque le i-ème client se connecte. On pose $X_i = 1$ si le i-ème client clique, 0 sinon. On a donc

$$X_i \sim \mathrm{Bernoulli}(p_{E_i}).$$

La stratégie E_i à l'instant i est une fonction (éventuellement aléatoire) de $(E_1, X_1), \dots, (E_{i-1}, X_{i-1})$. On cherche à définir une stratégie efficace pour l'annonceur, c'est-à-dire qu'asymptotiquement on propose la meilleure publicité :

$$\frac{X_1 + \dots + X_n}{n} \overset{n\to+\infty}{\to} \max\{p_A, p_B\}$$

(convergence presque-sûre ou en probabilité). Encore mieux : on souhaite maximiser les gains moyens à horizon fini

$$\max_{(E_i)_i \text{ stratégies}} \mathbb{E}[X_1 + \dots + X_n].$$

Une méthode sous-optimale : la ε-exploration

Considérons la stratégie suivante :

- On fixe un paramètre $\varepsilon > 0$ "petit".
- On choisit $E_1 = A, E_2 = B$.
- Pour $i \geq 3$, on note M_i la publicité qui a eu le meilleur "taux de clic" jusque-là (en cas d'ex-aequo entre A, B on se donne une règle)
  - Avec proba $1 - \varepsilon$, on prend $E_i = M_i$,
  - Avec proba ε on prend $E_i = \mathrm{non}(M_i)$.

(On considère qu'avec probabilité ε on "explore", alors qu'avec probabilité $1 - \varepsilon$ on "exploite".)

Do it yourself.

1. Intuitivement, quelle est la limite de $(X_1 + \dots + X_n)/n$? Asymptotiquement (en n), quel semble être le meilleur choix pour ε?
2. On fixe $n = 1000$, $p_A = 0.4$, $p_B = 0.6$. Choisir quelques valeurs de ε et tracer quelques trajectoires
$$i \in \{1, \dots, n\} \mapsto \frac{1}{i}(X_1 + \dots + X_i)\,.$$
Est-ce que les courbes sont compatibles avec votre intuition de la question précédente?
3. Pour $N = 100$, $p_A = 0.4$, $p_B = 0.6$, déterminer par simulation la valeur de ε qui optimise $\mathbb{E}\,[X_1 + \dots + X_i]$. (Prendre 10000 simulations pour chaque valeur de ε.)

Answers.

1.

```
# Question 2
```

```
# Question 3
```

Une stratégie asymptotiquement optimale?

On fait maintenant varier la probabilité d'exploration. L'algorithme est le même que précédemment sauf qu'on remplace ε par une suite $(\varepsilon_i)_{i\geq 3}$ d'éléments de $(0, 1)$. On pourra essayer

- $\varepsilon_i = 1/i^2$
- $\varepsilon_i = 1/\sqrt{i}$
- $\varepsilon_i = 1/\log i$

> **Do it yourself.** On fixe à nouveau $n = 1000$, $p_A = 0.4$, $p_B = 0.6$. Pour les exemples des 3 suites ci-dessus, tracer quelques trajectoires de courbes
> $$i \in \{1, \dots, n\} \mapsto \frac{1}{i}(X_1 + \cdots + X_i)\,.$$

```

```

> **Do it yourself.** Que doit vérifier la suite (ε_i) pour que l'on ait
> $$\frac{X_1 + \cdots + X_n}{n} \overset{n\to+\infty}{\to} \max\{p_A, p_B\}?$$

> **Answers.**

2. Apprentissage markovien : Pierre-Feuille-Ciseaux

Préliminaires et notations

On considère 2 joueurs X, Y qui jouent à *Pierre-Feuille-Ciseaux*. Pour $t \geq 1$ on note $X_t \in \{\text{Pi,Fe,Ci}\}$ le choix du joueur à l'instant t et Y_t le choix de son adversaire.

On appelera `Historique` la liste

$$[[X_1, Y_1], [X_2, Y_2], \dots, [X_T, Y_T]]$$

des choix des joueurs jusqu'à l'instant actuel.

Dans la présentation on représentera aussi `Historique` schématiquement de la façon suivante :

$$\begin{array}{c} (X_t) \\ (Y_t) \end{array} \begin{bmatrix} \mathrm{Pi} & \mathrm{Ci} & \mathrm{Pi} & \mathrm{Fe} & \mathrm{Ci} & \mathrm{Pi} & \mathrm{Pi} \\ \mathrm{Fe} & \mathrm{Fe} & \mathrm{Ci} & \mathrm{Pi} & \mathrm{Fe} & \mathrm{Ci} & \mathrm{Fe} \end{bmatrix}$$

désigne également la liste

$$[[\mathrm{Pi}, \mathrm{Fe}], [\mathrm{Ci}, \mathrm{Fe}], \dots, [\mathrm{Pi}, \mathrm{Fe}]]$$

avec $T = 7$.

On suppose que l'adversaire fait ses choix suivant une mémoire "courte" et une source d'aléa :

- Y_1, Y_2 sont arbitraires.
- pour tout $t \geq 2$,

  $$Y_{t+1} = \phi\left(X_{t-1}, X_t, Y_{t-1}, Y_t, A_{t+1}\right),$$

  où

  - ϕ est une fonction déterministe $\{\mathrm{Pi,Fe,Ci}\}^4 \times [0,1] \to \{\mathrm{Pi,Fe,Ci}\}$
  - (A_t) est une suite de variables i.i.d. uniformes dans $[0,1]$.

Par exemple, la stratégie (assez efficace...) consistant à toujours tirer uniformément au hasard (indépendamment des choix précédents) correspond à

$$\phi(X_{t-1}, X_t, Y_{t-1}, Y_t, A_{t+1}) = \mathrm{Pi} \times \mathbf{1}_{A_{t+1}<1/3} + \mathrm{Fe} \times \mathbf{1}_{1/3<A_{t+1}21/3} + \mathrm{Ci} \times \mathbf{1}_{2/3<A_{t+1}}.$$

alors que la stratégie (idiote) consistant à jouer ce que l'adversaire vient de jouer est

$$\phi(X_{t-1}, X_t, Y_{t-1}, Y_t, A_{t+1}) = X_t.$$

Réponse à une stratégie

On introduit donc une fonction $(p,q,r) \mapsto \mathtt{ReponseStrategie}(p,q,r) \in [0,1]^3$ de la façon suivante :

$$\mathtt{ReponseStrategie}(p,q,r) = \mathrm{argmax}_{(x,y,z)} \mathbb{E}[\text{Gain pour } X \text{ si } X \text{ tire selon } (x,y,z) \text{ et } Y \text{ tire}$$

(Dans cette espérance on considère qu'un victoire vaut 1 et une défaite -1.)

Par exemple, vous pouvez vérifier que si $(p,q,r) = (0.01, 0, 0.99)$ alors cela signifie que Y va jouer `Ci` presque à tous les coups et donc X joue `Pi` :

$$\mathtt{ReponseStrategie}(0.01, 0, 0.99) = (1, 0, 0).$$

Do it yourself.
Que vaut la fonction `ReponseStrategie()` ? *(Indication : les calculs sont assez simples.)*
Implémenter la fonction `ReponseStrategie()` .

Answers.

Pour aller plus vite voici le tableau des gains de X :

X/Y	Pi (p)	Fe (q)	Ci (r)
Pi (x)	0	-1	1
Fe (y)	1	0	-1
Ci (z)	-1	1	0

```python
def ReponseStrategie(p,q,r):
    return


# Test
print(ReponseStrategie(0.33,0.33,0.33))
```

```
None
```

Pour commencer à coder une stratégie d'apprentissage on définit une fonction `Stats` qui en fonction d'un historique `Historique` donné et de `PasseRecent` $=[[X_{t-1},X_t],[Y_{t-1},Y_t]]$ renvoie le nombre de fois où le joueur i ($i=0$ correspond au joueur X) a joué `'Pi'`, `'Fe'` ou `'Ci'`.

Par exemple, avec l'exemple précédent pour `Historique` et

```python
TestHistorique=[['Pi','Fe'],['Ci','Fe'],['Pi','Ci'],['Fe','Pi'],['Ci','Fe'],['Pi','Ci'],['Pi','Fe']]
TestPasse=[['Ci','Fe'],['Pi','Ci']]
```

on doit obtenir

```python
Stats(TestHistorique,TestPasse,1)
>[1, 1, 0]
```

car dans la situation de ce passé récent le joueur 1 a joué 1 fois `Pi` et une fois `Fe` :

$$\begin{array}{c} (X_t) \\ (Y_t) \end{array} \begin{bmatrix} \text{Pi} & \text{Ci} & \text{Pi} & \boxed{\text{Fe}} & \text{Ci} & \text{Pi} & \boxed{\text{Pi}} \\ \text{Fe} & \text{Fe} & \text{Ci} & \text{Pi} & \text{Fe} & \text{Ci} & \text{Fe} \end{bmatrix}$$

Do it yourself.

Implémenter la fonction `Stats`.

```python
def Stats(Historique,PasseRecent,i):
    # Historique : liste des coups précédents
    # PasseRecent : liste de la forme [[x1,x2],[y1,y2]]
    # i =0 ou 1 : joueur dont on veut calculer les stats
```

```
    return

TestHistorique=[['Pi','Fe'],['Ci','Fe'],['Pi','Ci'],['Fe','Pi'],['Ci','Fe'],['Pi','Ci']
TestPasse=[['Ci','Fe'],['Pi','Ci']]

# Test
print(Stats(TestHistorique,TestPasse,1))
```

L'algorithme d'apprentissage

On considère la stratégie suivante. Pour un paramètre $\varepsilon > 0$ (petit),

- Si $t = 1, 2, 3$ on tire au hasard uniformément entre Pi,Fe,Ci.
- Pour $t \geq 4$
  - **avec probabilité $1 - \varepsilon$ on "exploite"** : on renvoie $\texttt{ReponseStrategie}(p, q, r)$ calculée sur les stats de Y dans l'historique. Plus précisément
    - En notant $(X_{t-2} = x_{t-2}, X_{t-1} = x_{t-1})$ et $(Y_{t-2} = y_{t-2}, Y_{t-1} = y_{t-1})$, on regarde toutes les fois où Y s'est retrouvé dans la situation $(X_{s-2} = x_{t-2}, X_{s-1} = x_{t-1})$ et $(Y_{s-2} = y_{t-2}, Y_{s-1} = y_{t-1})$. On calcule alors les fréquences $\hat{p}, \hat{q}, \hat{r}$ avec lesquelles Y a joué Pi-Fe-Ci.
    - On renvoie $\texttt{ReponseStrategie}(\hat{p}, \hat{q}, \hat{r})$
  - **avec probabilité ε on "explore"** : on tire au sort uniformément au hasard entre Pi,Fe,Ci.

Tirages indépendants

> **Do it yourself.**
> Ecrire une fonction `TiragePiFeCi(p,q,r)` qui renvoie Pi,Fe,Ci avec les probabilités p, q, r.

```
def TiragePiFeCi(p,q,r):
    # Renvoie 'Pi','Fe','Ci' avec probas p,q,r
    return
```

Do it yourself.
Ecrire une fonction

```
JoueurApprentissage(Historique,eps,i)
```

qui en fonction de l'historique et du paramètre ε renvoie le coup de X (si $i = 0$) ou Y (si $i = 1$) suivant l'algorithme d'apprentissage ci-dessus.

```python
def JoueurApprentissage(Historique,eps,i):
    # input: i = numéro du joueur
    # output: Pi,Fe,Ce
    return
```

X apprend / Y joue i.i.d.

On va jouer une partie de longueur T entre X (qui apprend) et Y (qui joue de façon indé.)

Do it yourself.
Faire jouer X selon l'algorithme d'apprentissage face à Y qui joue des coups i.i.d. de probabilités $(0.6, 0.2, 0.2)$. Tracer l'évolution des proportions des coups joués par X.

X apprend / Y apprend

Do it yourself.
Faire jouer X et Y m'un contre l'autre, chacun suivant selon l'algorithme d'apprentissage. Tracer l'évolution des proportions des coups joués par X.