TP d'introduction aux bandits
Cours APR, M2 Info

H. Kolher & Ph. Preux

16 octobre 2022

1 Préliminaires

Avec ce sujet, vous avez reçu le fichier `bandits.py` que vous allez utiliser pour le TP.

2 Prise en main de l'environnement

2.1 Définition d'un problème de bandits

Nous allons utiliser des bandits avec des bras Bernoulli. Pour créer un problème de bandits à 3 bras de Bernoulli avec respectivement les probabilités 0,3, 0,42 et 0,4 de succès, on tape :

```
import bandits


b = bandits.BernoulliBandit (np.array ([0.3, 0.42, 0.4]))
```

Ensuite, on peut tirer un bras parmi les 3 en appelant la fonction `pull` avec le numéro du bras en paramètre :

```
b.pull (0)
b.pull (1)
b.pull (2)
```

`b.nbr_arms` est le nombre de bras du bandit `b` :

```
>>> b.nbr_arms
3
```

`b.best_arm` est le numéro du meilleur bras du bandit `b` :

```
>>> b.best_arm
3
```

`b.best_reward` est l'espérance de retour associé au meilleur bras du bandit `b`, autrement dit la probabilité de gain du meilleur bras (pour des bras Bernoulli) :

```
>>> b.best_reward
3
```

2.2 Rappel concernant la génération de nombres pseudo-aléatoires

On pourra utiliser ce que l'on a vu dans le TP sur les nombres pseudo-aléatoires : `np.random.rand (n)` retourne `n` nombres pseudo-aléatoires dans $[0, 1[$. Pour générer un nombre entier, on pourra utiliser `np.random.randint (n)` qui retourne un entier aléatoire dans l'ensemble `range (n)`.

On peut aussi utiliser les fonctions du paquetage `random` à la place du paquetage `numpy` : `random.rand ()` et `random.randint (a, b)` qui retournent chacune un nombre pseudo-aléatoire à la fois.

On n'oubliera pas ce que l'on a vu lors du premier TD concernant la reproductibilité des expériences (voir aussi le poly, section 7.10). Si vous me donnez votre code, en l'exécutant, je dois obtenir les mêmes résultats que vous.

Il faut donc absolument initialiser le/les générateurs de nombre pseudo-aléatoires que vous utilisez : `random.seed (xxx)` pour utiliser le paquetage `random`, où `xxx` est un nombre naturel.

3 Stratégie aléatoire

On commence par implanter une simple stratégie aléatoire, c'est-à-dire, on choisit un bras au hasard et on le tire.

Cette stratégie n'a aucun intérêt si ce n'est d'être très facile à implanter et toute stratégie « intelligente » doit avoir une meilleure performance que la stratégie uniformément aléatoire. C'est ce que l'on appelle une « ligne de base » (*baseline* en anglais).

3.1 Première version

Écrire une fonction qui prend en paramètre un bandit et un entier `n` et qui réalise `n` tirages de bras de ce bandit selon la stratégie aléatoire.

Cette fonction affiche deux graphiques. Tous deux indiquent les itérations en abscisses. Leurs ordonnées sont :

— la somme des retours,

— le regret cumulé.

En cas de besoins, l'annexe indique comment on réalise une représentation graphique.

Rappelons que le regret cumulé est la somme des regrets immédiats, c'est-à-dire le regret associé à chaque tirage de bras. Pour des bras dont la récompense est distribuée selon des lois de Bernoulli de paramètre p_k (probabilité de succès lors d'un tirage du bras k), en notant $p^* = max_k p_k$, le regret immédiat correspondant au tirage du bras k est $p^* - p_k$.

3.2 Version de travail

Dans la suite du TP, on va vouloir comparer différentes stratégies. On va comparer leurs performances en comparant leurs regrets cumulés. On va donc définir un ensemble de fonctions pour différentes stratégies. Chacune renvoie le regret cumulé au fil du temps. Les graphiques seront réalisés en dehors de la fonction. Par ailleurs, comme l'environnement est stochastique, la performance d'une stratégie est variable si on l'exécute plusieurs fois. On exécutera donc plusieurs fois chaque stratégie.

En résumé :

— retirez les instructions graphiques de la fonction que vous venez d'écrire.

— Ajoutez un paramètre `N` indiquant le nombre d'expériences à réaliser.

— La fonction renvoie le regret cumulé au fil des itérations pour chacune de ces `N` exécutions (un tableau bi-dimensionnel).
— Vous utilisez cette fonction en l'appelant puis en réalisant des graphiques (si on vous le demande ou si vous êtes curieux, ce qui n'est pas du tout un vilain défaut ici).

Faites tout cela. Assurez-vous que tout fonctionne bien. Exécuter votre fonction avec `N = 100` et représentez graphiquement le regret cumulé de l'ensemble de ces 100 exécutions. Qu'observez-vous ?

4 Stratégies gloutonnes

En pratiquant comme pour une stratégie de choix uniformément aléatoire :

1. implanter la stratégie *follow the leader*. Comparez avec la stratégie aléatoire. Concluez.
2. implanter une stratégie de choix ϵ-glouton. Vous passerez la valeur d'ϵ en paramètre. Vous comparerez expérimentalement (à l'aide de graphiques) les performances de cette stratégie pour différentes valeurs de ϵ (par exemple $0{,}1, 0{,}5, 0{,}85$). Comparez également avec la stratégie aléatoire. Concluez.
3. implanter une stratégie de choix gloutonne avec ϵ décroissant. Vous ferez décroître ϵ en multipliant sa valeur à chaque itération par un facteur tel que 0,99. Vérifier que cette valeur est adéquate : ϵ ne doit pas décroître trop vite ... ni trop lentement.

Rappel : on note $N_b(t)$ le nombre de tirages du bras b jusqu'au temps $\leq t$ et $S_b(t)$ le nombre de retours $= 1$ collectés en tirant le bras b jusqu'au temps t.

Une stratégie ϵ-gloutonne consiste :
— à tirer un bras au hasard avec une probabilité ϵ ;
— à sélectionner le bras ayant le rapport $\frac{S_b(t)}{N_b(t)}$ maximal avec une probabilité $1 - \epsilon$,

5 Stratégie proportionnelle et softmax

5.1 Stratégie proportionnelle

Mêmes questions pour la stratégie proportionnelle.

Rappel : cette stratégie consiste à associer la probabilité $\rho_b(t) = \frac{S_b(t)}{N_b(t)}$ à chaque bras b et à tirer au hasard un bras en utilisant cette distribution de probabilités.

La fonction `choices ()` du paquetage `random` peut vous être utile.

5.2 Stratégie de Boltzmann

Mêmes questions pour la stratégie de Boltzmann. On testera plusieurs valeurs de τ (par exemple, 1000, 1 et 0,001).

Rappel : c'est le même principe que la stratégie proportionnelle sauf que la distribution de probabilités est maintenant définie par : $p_b(t) = \frac{q_b(t)}{\sum_b q_b(t)}$, avec $q_b(t) = e^{\rho_b(t)/\tau}$.

6 Stratégie optimiste

Mêmes questions pour UCB.

Rappel : UCB consiste à tirer le bras qui maximise $\frac{S_b(t)}{N_b(t)} + \sqrt{\alpha \frac{\log(t)}{N_b(t)}}$.
Étudier la performance en faisant varier α, par exemple en prenant par exemple $\alpha \in \{0{,}5, 1, 2\}$.

7 Stratégie d'échantillonnage de Thompson

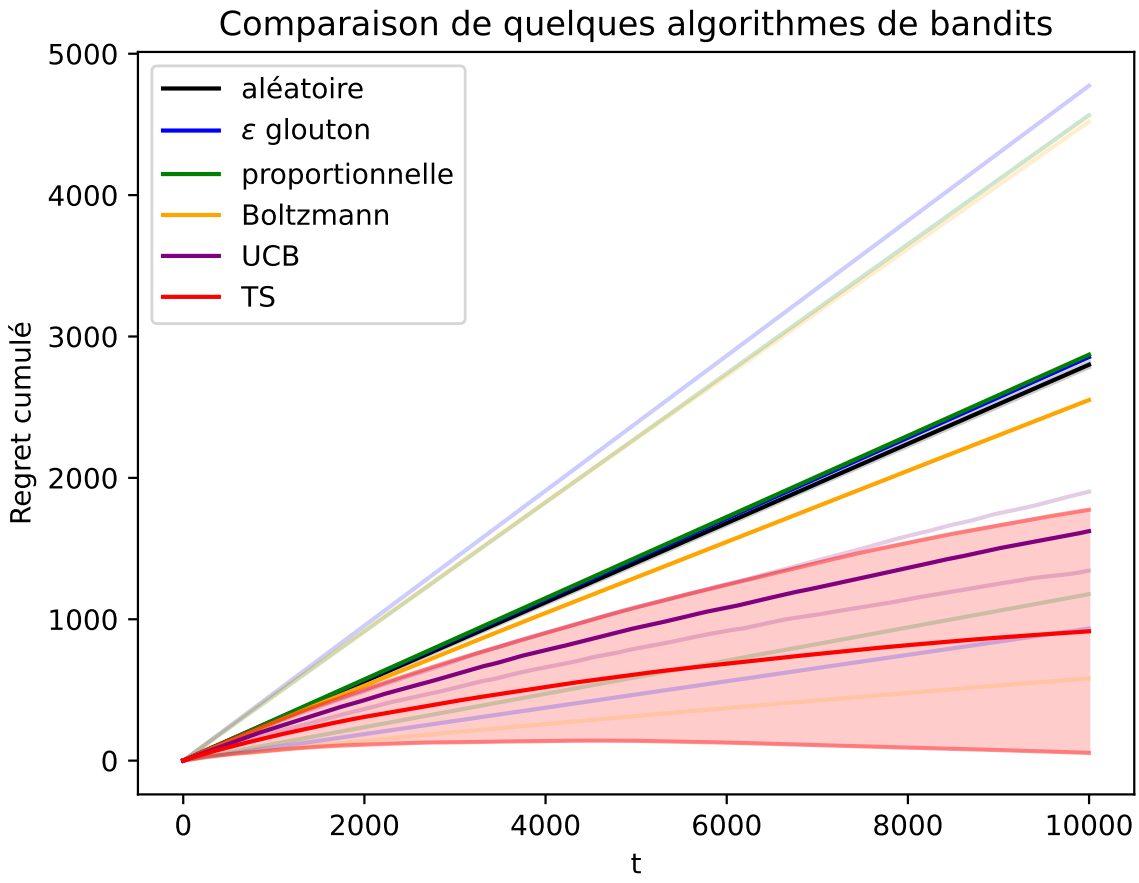
Même question pour l'échantillonnage de Thompson.
Rappel : pour choisir le bras à tirer à l'instant $t \geq 1$:

1. on échantillonne chacune des distributions $\mu_b(t) = \text{Bêta}(S_b(t-1), N_b(t-1) - S_b(t-1) + 1)$;
2. on joue le bras pour lequel $\mu_b(t)$ est maximal ;
3. on met à jour les $S_b(t)$ et $N_b(t)$ pour tous les bras.

Pour générer un nombre pseudo-aléatoire selon une loi Bêta (a, b), on utilise la fonction `betavariate (a, b)` du paquetage `random`.

8 Pour terminer

Comparez ces différentes stratégies. Discutez vos résultats. Rédigez un petit document.
Pour illustrer votre rapport, vous pouvez générer un graphique comme celui-ci :



En traits pleins, on indique le regret cumulé moyen de chaque algorithmes à chaque itération. En traits de même couleur plus fins, on a tracé la moyenne ± 1 écart-type.

A Représentation graphique

On donne un exemple très simple de représentation graphique. Dans l'exemple ci-dessous, la variable `les_x` contient les abscisses des points à représenter, la variable `les_y` leurs ordonnées. Ici, les ordonnées sont une fonction mathématique des abscisses, mais cela pourrait être des points définis par des couples (x_i, y_i).

Détail technique : le paramètre `alpha` dans l'appel de la fonction `ax.plot` est un facteur de transparence. Dans l'exemple donné ici, ce paramètre n'est pas vraiment utile. Il devient très utile quand on représente plusieurs tracés qui se chevauchent sur un même graphique, comme on le fait dans le TP.

```
from math import sin
import matplotlib.pyplot as plt
import numpy as np

n = 100
fig, ax = plt.subplots ()
les_x = range (n)
les_y = les_x * np.sin (les_x)

ax.set_title ('Un exemple de tracé de fonction')
ax.xaxis.set_label_text ('x')
ax.yaxis.set_label_text ('x sin (x)')
ax.plot (les_x, les_y, color = 'red', alpha = 0.2)
fig.show ()
```

Vous pouvez sauvegarder un tel graphique dans un fichier pdf en tapant :
`fig.savefig ('mon-joli-graphique.pdf')`.