TP Bandits Manchots

BARREYRE Alexis – FAUCHERY Benoit – GOJARD Romain

[Github]

Exercice 1 : Le bandit manchot

1.

```
class Bandit:
    def __init__(self):
        self.avg = random.gauss(0, 1)
```

2.

```
def play(self):
        return random.gauss(self.avg, 1)
```

3.

```
from classes import Bandit
import matplotlib.pyplot as plt

bandit1, bandit2, bandit3 = Bandit(), Bandit(), Bandit()
points1, points2, points3 = [], [], []

for i in range(1000):
    value1, value2, value3 = bandit1.play(), bandit2.play(), bandit3.play()
    points1.append(value1) # affichage matplotlib
    points2.append(value2)
    points3.append(value3)
    print(value1, value2, value3)
```

Exercice 2 : Le Ban-10

1.

```
class BanDix:
    def __init__(self):
        self.tab = []
        for i in range(10):
            self.tab.append(Bandit())
```

2.

```
def __init__(self):
        self.tab = []
        self.maxAvg = None
        self.maxBanditIndex = 0
        for i in range(10):
            newBandit = Bandit()
            self.tab.append(newBandit)
            if self.maxAvg is None or newBandit.avg > self.maxAvg:
                self.maxAvg = newBandit.avg
                self.maxBanditIndex = i

        self.banditMaxAvg = self.maxBanditIndex
```

3.

```
def play(self, arm_number):
        if arm_number > 9 or arm_number < 0:
            raise ValueError("Valeur impossible, erreur")

        return self.tab[arm_number].play()
```

Exercice 3 : Algorithme ε-greedy

1.

```
class GreedyPlayer:

    def __init__(self, n, eps):
        self.n = n
        self.eps = eps
```

2.

```
        self.action_values = [0] * 10
        self.eval_count = [0] * 10
```

3 & 4.

```
def get_action(self):
        explore = random.random() < self.eps
        return explore
```

5.

```
def _greedy_action(self):
        best_actions = []
        highest_value = -float("inf")

        for i in range(len(self.action_values)):
```

```
            if (self.action_values[i] > highest_value):
                best_actions = []
                best_actions.append(i)
                highest_value = self.action_values[i]
            elif (self.action_values[i] == highest_value):
                best_actions.append(i)

        return random.choice(best_actions)
```

6.

```
def _random_action(self):
        return random.randint(0, self.n - 1)
```

7.

```
        if explore < self.eps:
            self._random_action()
        else:
            self._greedy_action()
```

8.

```
def reward(self, action, reward):
        self.eval_count[action] += 1
        self.action_values[action] += (reward - self.action_values[action]) /
self.eval_count[action]
```

9. & 10

```
ban10 = BanDix()
greedy = GreedyPlayer(0.1)

for i in range(1000):
    action = greedy.get_action()
    reward = ban10.play(action)
    greedy.reward(action, reward)
    ban10.__str__()
    greedy.__str__()
```

11.

```
n 10

eps 0.1

action values [-9.21869284012083, 16.19406541010018, 10.011102665373734, 2.580831575774214, 18.013587508321237, 29.461709064672455, -8.678150576175758, -3.2500138450
37964, 35.06133338690486, 753.5069563754992]

eval count [11, 10, 9, 13, 13, 17, 13, 12, 89, 813]
```

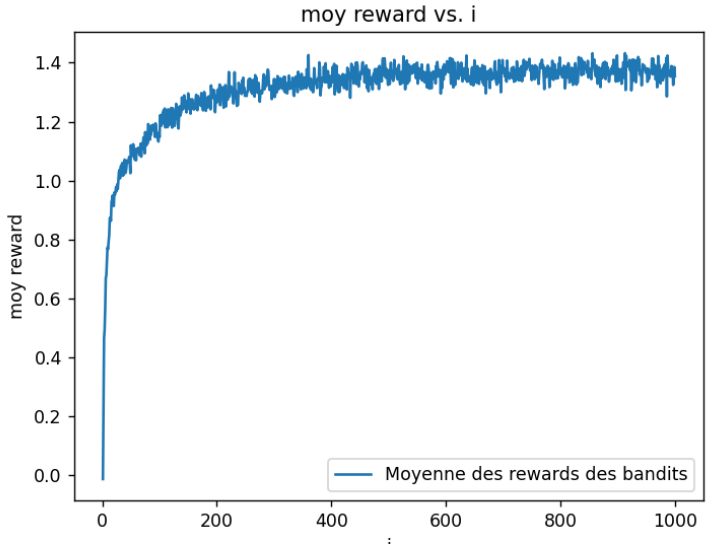
Exercice 4 : Graphiques simples

1.

```
points = []

for i in range(1000):
    action = greedy.get_action()
    reward = ban10.play(action)
    points.append(reward)
    greedy.reward(action, reward)
    ban10.__str__()
    greedy.__str__()

# Créer 1 sous-graphique
fig, axs = plt.subplots(1, 1, figsize=(8, 12))

axs.plot(range(1, 1001), points, label='Rewards')
axs.set_xlabel('i')
axs.set_ylabel('value')
axs.set_title('Rewards')
axs.legend()

# Afficher les graphiques
plt.show()
```

2. & 3.

```
nbOfGreedy = 2000

for i in range(nbOfGreedy):
    tableBan10.append(BanDix())
    tableGreedyP.append(GreedyPlayer(0.1))
    print([tableBan10[-1].tab[j].avg for j in range(10)])

for i in tqdm(range(1000)):
    for j in range(nbOfGreedy):
        action = tableGreedyP[j].get_action()
        reward = tableBan10[j].play(action)
        tabPointsDesGreedy[i] += reward / nbOfGreedy
        tableGreedyP[j].reward(action, reward)
```



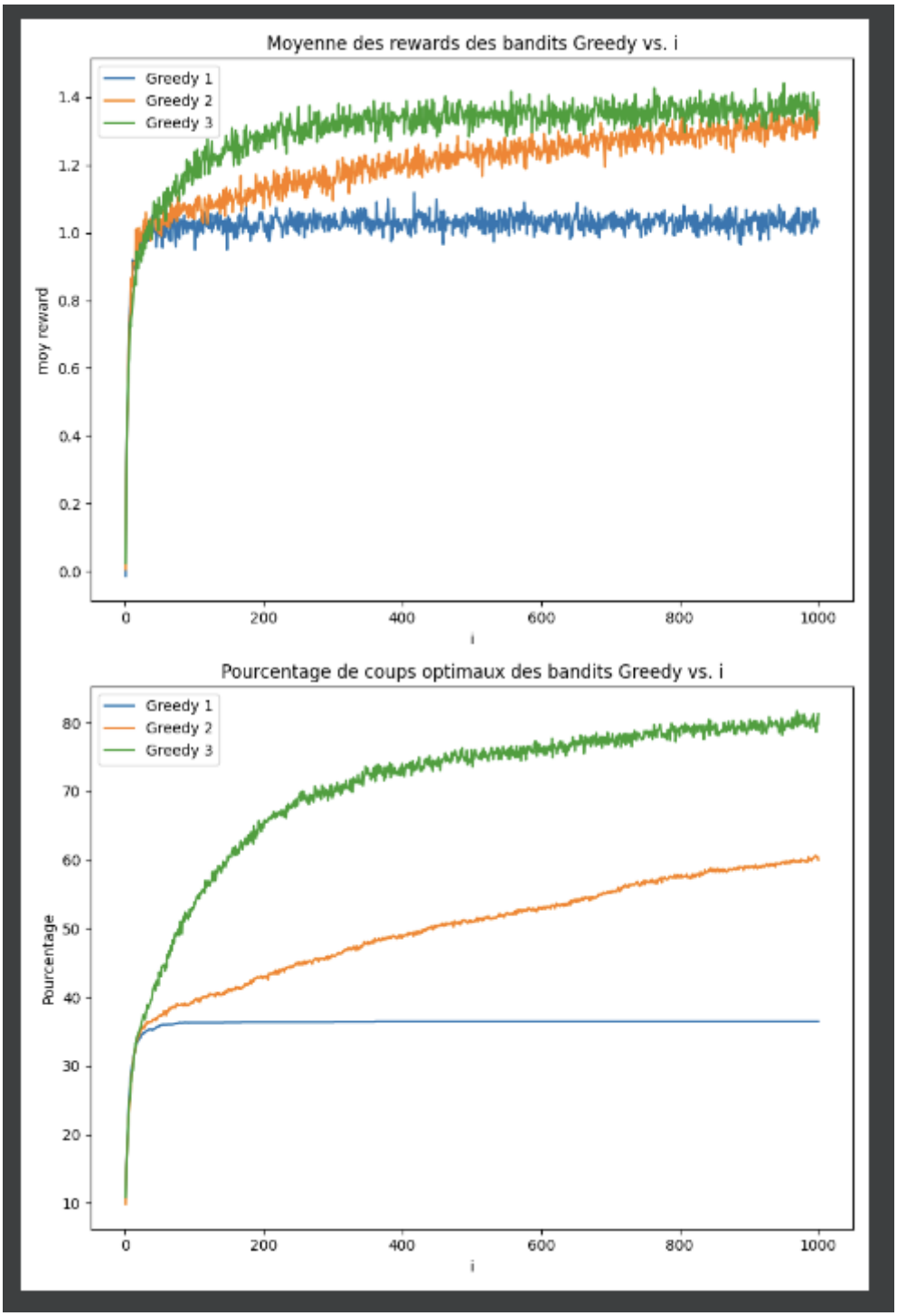
4.

```
for i in tqdm(range(1000)):
    for j in range(nbOfGreedy):
        action = tableGreedyP[j].get_action()
        reward = tableBan10[j].play(action)
        tabPointsDesGreedy[i] += reward / nbOfGreedy
        tableGreedyP[j].reward(action, reward)
        if action == tableBan10[j].maxBanditIndex:
            tabPourcentageGreedy[i] += 1 / 20
```

5.

```
nbTabTabGreedy = 3
nbOfGreedy = 2000
tabPointsDesGreedy = [[0] * 1000 for _ in range(nbTabTabGreedy)]
tableTableBan10 = [[] for _ in range(nbTabTabGreedy)]
tableTableGreedy = [[] for _ in range(nbTabTabGreedy)]
tabTabPourcentageGreedy = [[0] * 1000 for _ in range(nbTabTabGreedy)]

for k in range(nbTabTabGreedy):
    for i in range(nbOfGreedy):
        tableTableBan10[k].append(BanDix())
        if k == 0:
            eps = 0
        elif k == 1:
            eps = 0.01
        else:
            eps = 0.1
        tableTableGreedy[k].append(GreedyPlayer(eps))

for i in tqdm(range(1000)):
    for k in range(nbTabTabGreedy):
```

```
        for j in range(nbOfGreedy):
            action = tableTableGreedy[k][j].get_action()
            reward = tableTableBan10[k][j].play(action)
            tabPointsDesGreedy[k][i] += reward / nbOfGreedy
            tableTableGreedy[k][j].reward(action, reward)
            if action == tableTableBan10[k][j].maxBanditIndex:
                tabTabPourcentageGreedy[k][i] += 1 / (nbOfGreedy / 100)
```



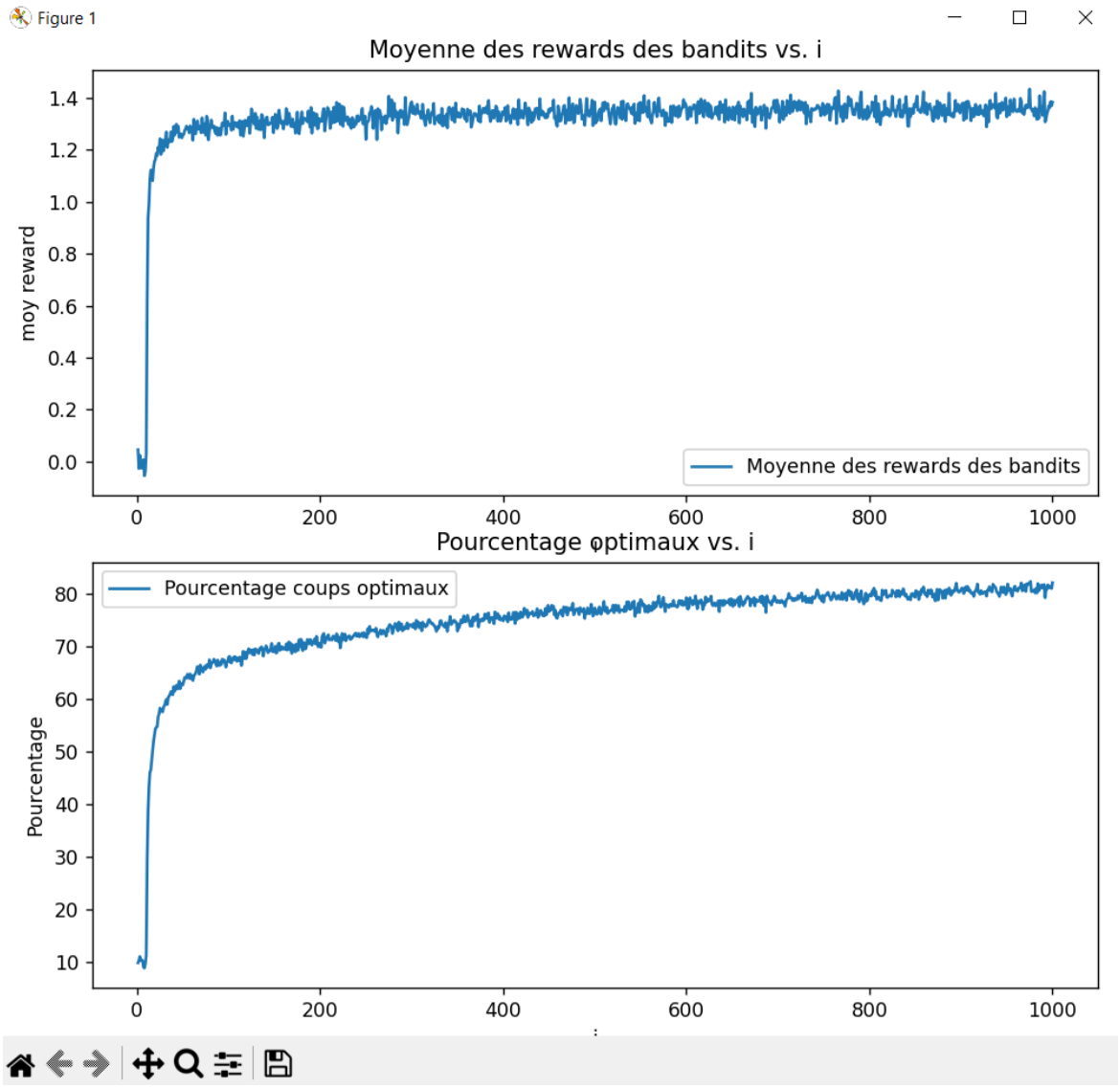
6. On constate que pour la courbe bleue (epsilon = 0), on n'a pas d'exploration donc la courbe stagne autour de 35% de choix optimal. La courbe orange (epsilon = 1) représente une exploration totale, permettant une croissance légère mais bloquant autour de 55%. La courbe verte (epsilon = 0.1) est la même qu'obtenue précédemment, et la plus satisfaisante des trois à long terme de part un maximum de 80% atteint.

Exercice 5 : Initialisation optimiste

1. & 2.

```
class OptimistGreedyPlayer:

    def __init__(self, eps, n=10):
        self.n = n
        self.eps = eps
        self.action_values = [5] * n
        self.eval_count = [0] * n
```

3.



Courbes de la classe OptimistGreedyPlayer

5.

On constate que sur les premières itérations, on a une légère descente avant de très vite converger à 80, beaucoup plus tôt qu'en greedy classique.

Le comportement asymptotique lui est le même qu'avant.

On en conclut que la version optimiste permet de beaucoup plus vite converger vers de meilleure valeurs avant une légère recherche avec des moins bonnes rewards.