SCIENCE DU NUMERIQUE

S9 - Introduction to Reinforcement Learning

Apprentissage par renforcement

Projet final - Load Balancing

Auteurs :
Bryan CHEN

25 octobre 2024

Toulouse INP - ENSEEIHT | Science du Numérique
High Performance Computing et Big Data
2023-2024

Table des matières

Le temps est discret. Soient Q_1 et Q_2 le nombre total d'emplois dans les serveurs 1 et 2, respectivement. À chaque intervalle de temps, le répartiteur observe (Q_1, Q_2) et effectue l'action a_i, $i = 1, 2$, qui envoie éventuellement un nouvel emploi entrant au serveur i. Le coût à chaque intervalle de temps est $Q_1 + Q_2$, indépendamment de l'action.

À chaque intervalle de temps, il y a une probabilité λ d'avoir un nouvel emploi, qui sera envoyé au serveur a_i. Si $Q_i > 0$, avec une probabilité μ_i, un emploi du serveur i sera retiré. Pour simplifier, nous supposerons qu'à chaque intervalle de temps, seul un événement peut se produire, c'est-à-dire soit une arrivée, soit un départ des serveurs 1 ou 2.

Par exemple, dans l'état $(2, 1)$, si le répartiteur prend l'action 1, alors avec une probabilité λ, l'état suivant sera $(3, 1)$, avec une probabilité $\mu_1, (1, 1)$, avec une probabilité μ_2, $(2, 0)$, et avec une probabilité de 1 - μ_1 - μ_2 - λ, l'état suivant sera $(2, 1)$.

Nous supposerons qu'il existe une borne supérieure pour à la fois Q_1 et Q_2, égale à 20. Si $Q_1 = 20$ ou $Q_2 = 20$, aucun nouvel emploi entrant n'arrivera dans le système.

Choisissons $\mu_1 = 0.2, \mu_2 = 0.4$ et $\lambda = 0.3$. Tout au long, nous prenons le facteur d'actualisation comme $\gamma = 0.99$.

1 Processus de Décision de Markov

1.1 Évaluation de Politique

Supposons la politique aléatoire qui envoie chaque emploi avec une probabilité de 0.5 vers la file d'attente 1 ou 2.

1.1.1 Écrivez l'équation de Bellman qui caractérise la fonction de valeur pour cette politique

L'équation de Bellman qui caractérise la fonction de valeur pour cette politique est la suivante :

$$\forall k \in \mathbb{N}, V_{k+1}(s) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V_k(s')$$

Où :
- $r(s, a)$ représente la récompense associée à l'état s et à l'action a.
- $V_k(s)$ représente la fonction de valeur à la k-ième itération.
- γ représente le facteur de réduction.
- $p(s'|s, a)$ représente la probabilité, à partir de s, de prendre l'action a et d'arriver à l'état s'.

Pour ce problème, chaque état représente le nombre d'emplois dans les deux serveurs, et la politique est telle que le répartiteur envoie chaque emploi avec une probabilité de 0.5 vers la file d'attente 1 ou la file d'attente 2. Ainsi, nous avons les informations suivantes :
- $s = (Q_1, Q_2)$ pour $(Q_1, Q_2) \in [0, 20] \times [0, 20]$
- $\forall s, a, r(s, a) = -(Q_1 + Q_2)$
- La probabilité de transition est telle que, pour p_1 et p_2 deux probabilités :
  - $p\left((Q_1, Q_2)|(Q_1, Q_2), a_1\right) = p_1(1 - \mu_1 - \mu_2 - \lambda)$
  - $p\left((Q_1, Q_2)|(Q_1, Q_2), a_2\right) = p_2(1 - \mu_1 - \mu_2 - \lambda)$
  - $p\left((Q_1 + 1, Q_2)|(Q_1, Q_2), a_1\right) = p_1\lambda$
  - $p\left((Q_1, Q_2 + 2)|(Q_1, Q_2), a_2\right) = p_2\lambda$
  - $p\left((Q_1 - 1, Q_2)|(Q_1, Q_2), a_1\right) = p_1\mu_1$
  - $p\left((Q_1 - 1, Q_2)|(Q_1, Q_2), a_2\right) = p_2\mu_1$
  - $p\left((Q_1, Q_2 - 1)|(Q_1, Q_2), a_1\right) = p_1\mu_2$
  - $p\left((Q_1, Q_2 - 1)|(Q_1, Q_2), a_2\right) = p_2\mu_2$

1.1.2 Calculez la fonction de valeur pour cette politique en utilisant l'évaluation itérative de politique. En raison du principe de contraction, le vecteur initial peut être arbitraire, vous pouvez donc prendre $V(Q_1, Q_2) = 0$ pour tous (Q_1, Q_2). Pour arrêter les itérations, vous pouvez prendre comme critère que la différence entre deux itérations doit être plus petite que δ, une petite valeur arbitraire.

Après l'implémentation de l'évaluation itérative de la politique pour cette politique, nous obtenons les résultats suivants :

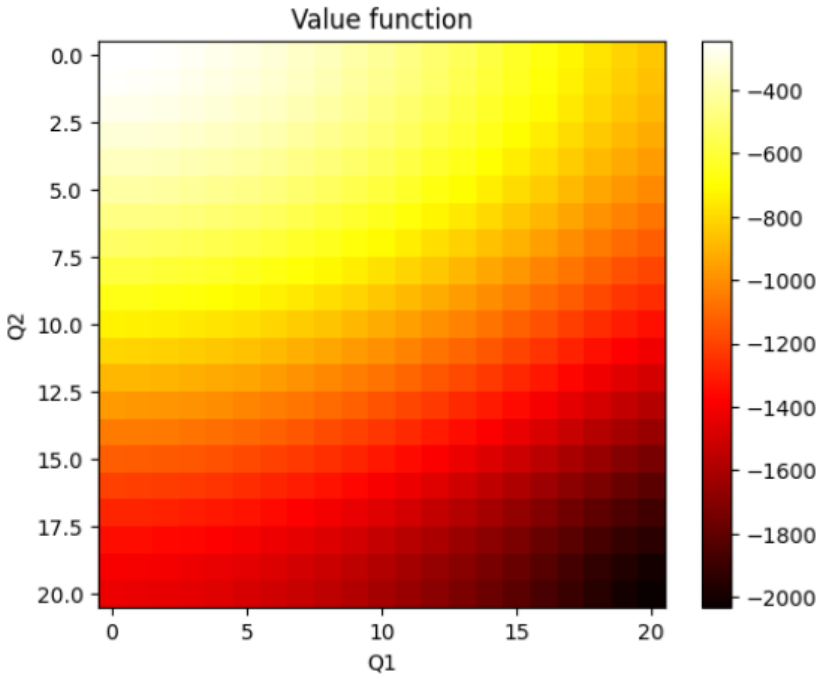

FIGURE 1 – Graphique de la fonction de valeur pour l'évaluation itérative de la politique

Ces résultats sont pour $\mu_1 = 0.2$ et $\mu_2 = 0.4$. Nous pouvons également observer que pour $\mu_1 = \mu_2$, nous avons une ligne pour la distribution.

1.2 Contrôle Optimal

Dans cette partie, on vous demande de trouver la politique optimale pour répartir les emplois entrants.

1.2.1 Écrivez l'équation de Bellman qui caractérise la politique optimale

L'équation de Bellman qui caractérise la politique optimale est la suivante :

$$\forall n \in \mathbb{N}, V_{n+1}(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_n(s') \right)$$

Où :
- $r(s,a)$ représente la récompense associée à l'état s et à l'action a.
- $V_n(s)$ représente la fonction de valeur à la n-ième itération.
- γ représente le facteur de réduction.
- $p(s'|s,a)$ représente la probabilité, à partir de s, de prendre l'action a et d'arriver à l'état s'.

1.2.2 Résolvez numériquement la fonction de valeur optimale par l'algorithme de l'itération de la valeur

Après l'implémentation de l'algorithme de l'itération de la valeur, nous obtenons le graphique suivant pour la fonction de valeur optimale.

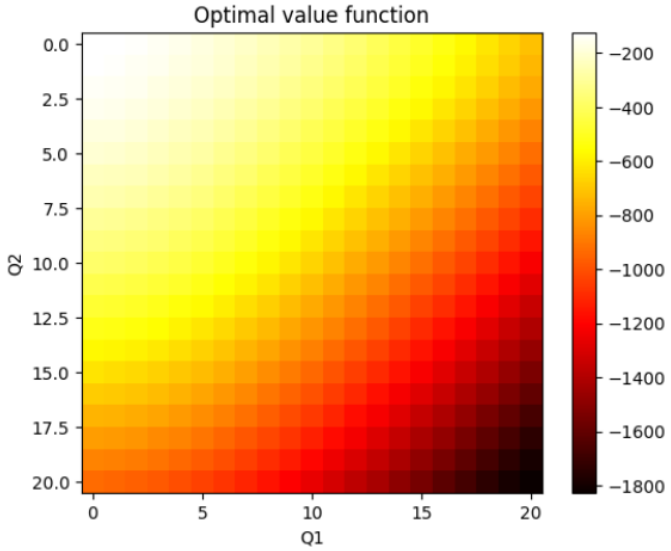

FIGURE 2 – Graphique de la fonction de valeur optimale

Nous avons le graphique suivant pour l'action optimale :

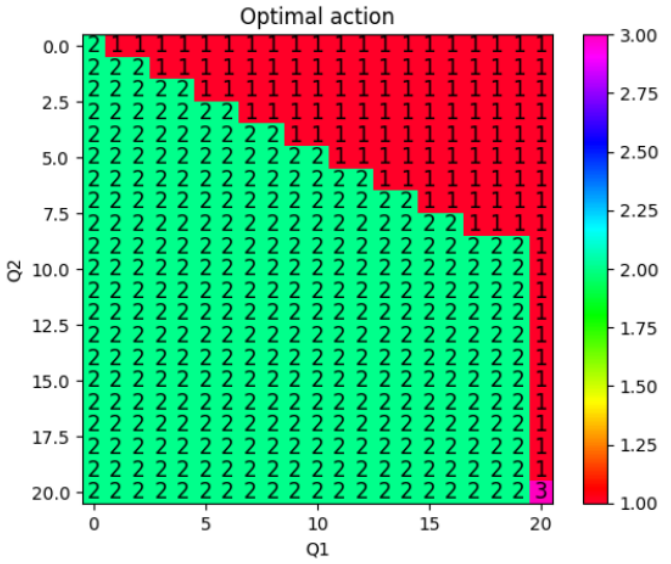

FIGURE 3 – Graphique de l'action optimale

On peut clairement voir que la politique optimale dépend beaucoup des valeurs de μ_1 et de μ_2. La ligne diagonale séparant les actions 1 et 2 est en fait parfaitement au milieu (diagonale parfaite) dans le cas de $\mu_1 = \mu_2$. Les graphiques précédents sont pour les valeurs $\mu_1 = 0.2$ et $\mu_2 = 0.4$.

1.2.3 Représentez sur le plan l'action optimale en fonction de l'état (Q_1, Q_2)

Nous avons la représentation suivante de l'action optimale sur le plan :

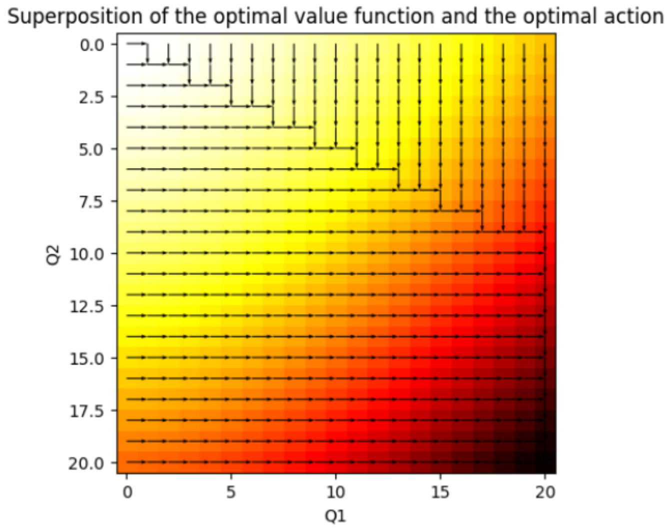

FIGURE 4 – Superposition de la fonction de valeur optimale et de l'action optimale

1.2.4 Comparez les performances obtenues avec la politique aléatoire et la politique optimale, comment pouvez-vous conclure que la politique optimale est meilleure ?

```
print(np.mean(Value_final_optimal))
print(np.mean(Value_final))
print(np.array((Value_final_optimal - Value_final) > 0))
# On peut remarquer que la difference entre la politique optimal et la politique
# aléatoire est bien positive. Ainsi la première est surement meilleure
# que la deuxième.
```

```
-753.3572880791564
-1010.0603848030414
[[ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
...
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True]]
```

FIGURE 5 – Comparaison des performances entre la politique aléatoire et la politique optimale

Nous observons que $\mathbb{E}[V_{\text{optimal}}] > \mathbb{E}[V_{\text{random}}]$, par conséquent, nous pouvons conclure que la politique optimale performe mieux que la politique aléatoire. Ce résultat était à prévoir étant donné que la politique aléatoire distribue de nouveaux emplois avec une probabilité de 0.5, indépendamment du meilleur choix, tandis que la politique optimale étudie la récompense possible pour prendre la meilleure décision.

1.2.5 Effectuez une amélioration d'une étape sur la politique aléatoire, que observez-vous ?

Nous obtenons le graphique suivant pour l'amélioration d'une étape sur la politique aléatoire :

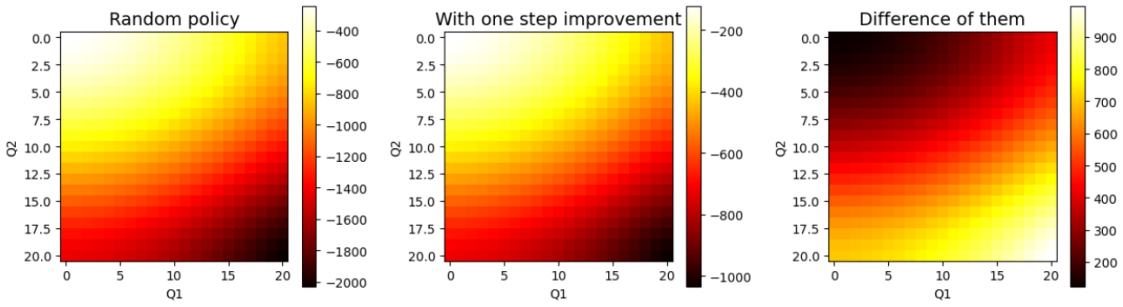

FIGURE 6 – Graphique de la fonction de valeur pour l'amélioration d'une étape

L'algorithme avec une *"amélioration d'une étape"* produit de meilleurs résultats par rapport à la politique aléatoire. Cela peut s'expliquer par plusieurs raisons :

Ajustement Local : La méthode *"amélioration d'une étape"* effectue un ajustement local de la politique en fonction de la valeur actuelle des états, tandis que l'évaluation d'une politique aléatoire peut ne pas exploiter pleinement les informations locales disponibles.

Avantage Immédiat : La procédure *"amélioration d'une étape"* prend en compte les rendements immédiats attendus pour chaque action dans chaque état, ce qui peut rapidement conduire à des améliorations locales significatives.

Utilisation de l'Information Actuelle : En utilisant la fonction de valeur actuelle, l' *"amélioration d'une étape"* prend des décisions plus éclairées, en tenant compte de la connaissance actuelle de l'environnement, ce qui peut conduire à des choix d'actions plus optimaux.

En résumé, cette approche exploite l'information actuelle pour effectuer des ajustements ciblés, ce qui peut rapidement améliorer les performances par rapport à une politique aléatoire.

2 Contrôle sans modèle tabulaire

2.1 Évaluation de Politique

Supposons la politique aléatoire qui envoie chaque emploi avec une probabilité de 0,5 vers la file d'attente 1 ou 2. Pour le paramètre d'apprentissage, vous pouvez utiliser $\alpha_n = \frac{1}{n}$.

2.1.1 Implémentez TD(0)

Pour le TD(0) (apprentissage de la différence temporelle), nous obtenons les résultats suivants pour $\alpha_n = \frac{1}{n}$:

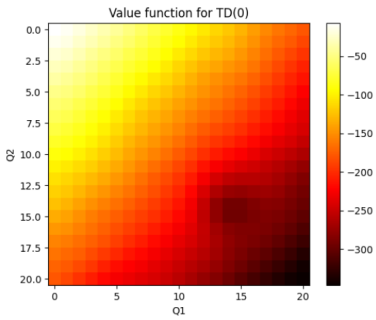

FIGURE 7 – Graphique de la fonction de valeur pour le TD(0), $\alpha = \frac{1}{n}$

2.1.2 Comparez la fonction de valeur obtenue avec les résultats de la Section 1

Nous avons les trois comparaisons suivantes avec les résultats de la Section 1 :

Tout d'abord, entre le TD(0) et la politique aléatoire, nous avons ce graphique :

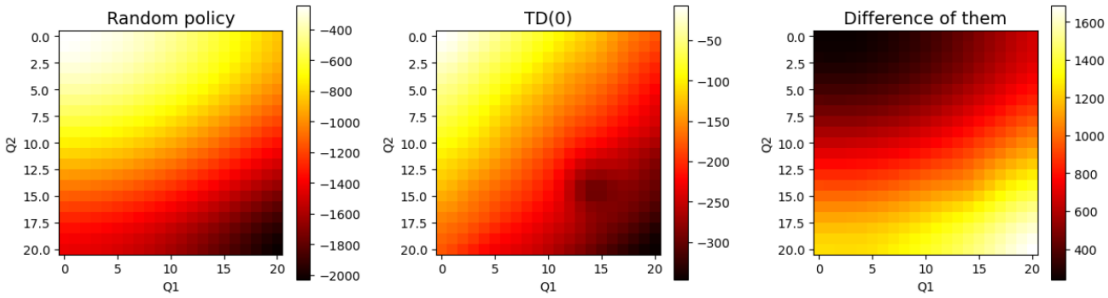

FIGURE 8 – Comparaison entre le TD(0) et la politique aléatoire, $\alpha = \frac{1}{n}$

Entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, nous avons ce graphique :

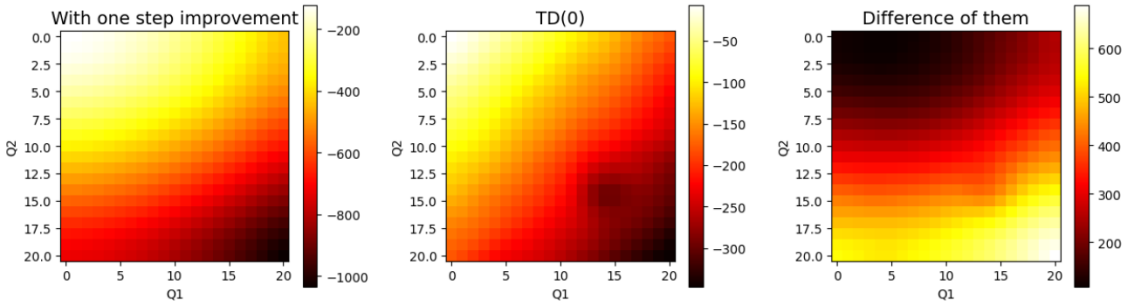

FIGURE 9 – Comparaison entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, $\alpha = \frac{1}{n}$

Enfin, entre le TD(0) et la valeur optimale, nous avons ce graphique :

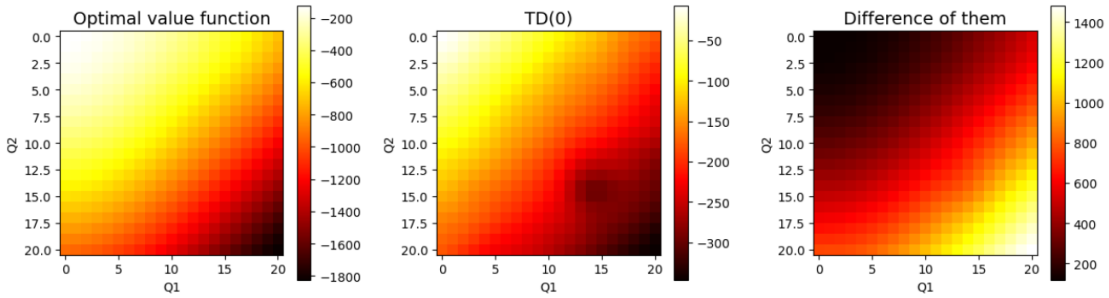

FIGURE 10 – Comparaison entre le TD(0) et la fonction de valeur optimale, $\alpha = \frac{1}{n}$

2.1.3 Explorez d'autres alternatives pour α_n, par exemple une constante, pour voir si la convergence s'améliore

- Pour $\alpha_n = 1$ (constante), nous avons les résultats suivants pour TD(0),

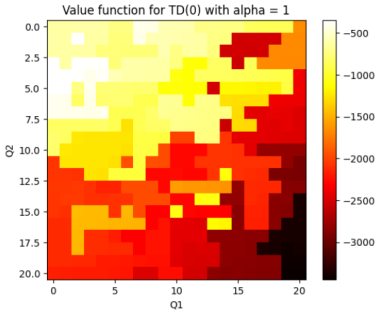

FIGURE 11 – Graphique de la fonction de valeur pour le TD(0), $\alpha = 1$

Tout d'abord, entre le TD(0) et la politique aléatoire, nous avons ce graphique :

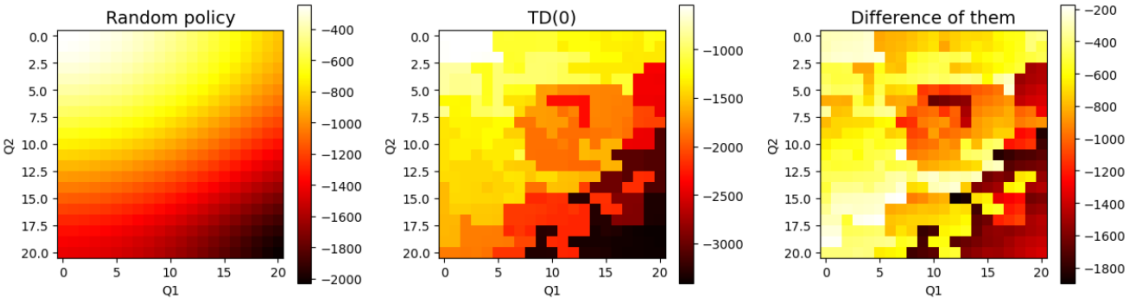

FIGURE 12 – Comparaison entre le TD(0) et la politique aléatoire, $\alpha = 1$

Entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, nous avons ce graphique :

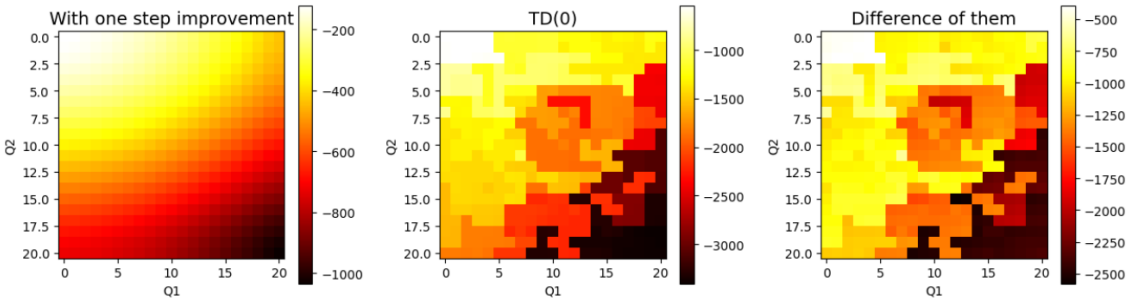

FIGURE 13 – Comparaison entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, $\alpha = 1$

Enfin, entre le TD(0) et la valeur optimale, nous avons ce graphique :

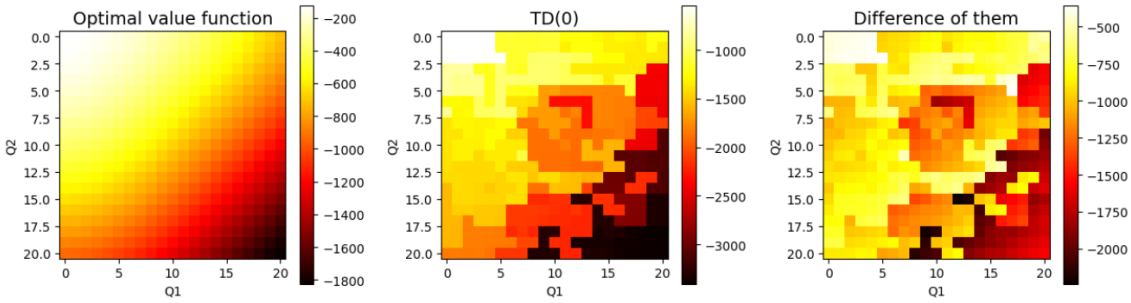

FIGURE 14 – Comparaison entre le TD(0) et la fonction de valeur optimale, $\alpha = 1$

• Pour $\alpha_n = \dfrac{1}{\sqrt{\sqrt{n}}}$, nous avons les résultats suivants pour TD(0),

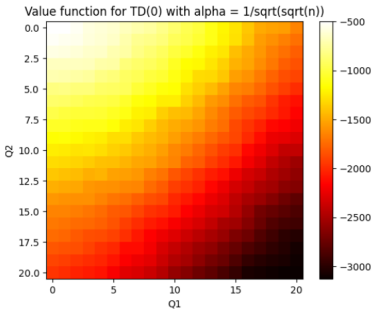

FIGURE 15 – Graphique de la fonction de valeur pour le TD(0), $\alpha = \dfrac{1}{\sqrt{\sqrt{n}}}$

Entre le TD(0) et la politique aléatoire, nous avons ce graphique :

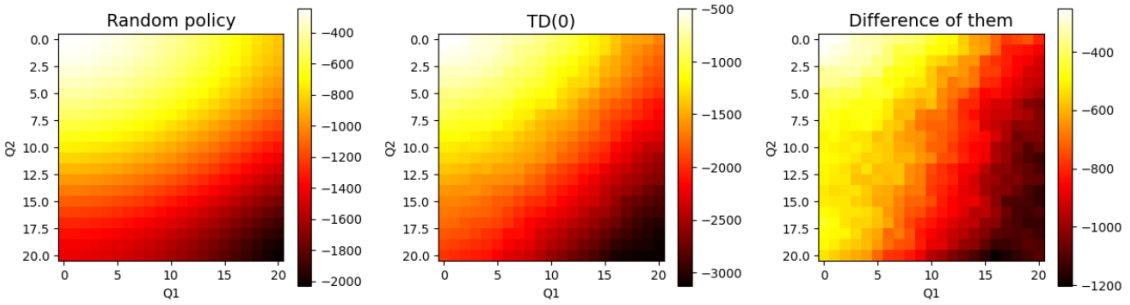

FIGURE 16 – Comparaison entre le TD(0) et la politique aléatoire, $\alpha = \frac{1}{\sqrt{\sqrt{n}}}$

Entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, nous avons ce graphique :

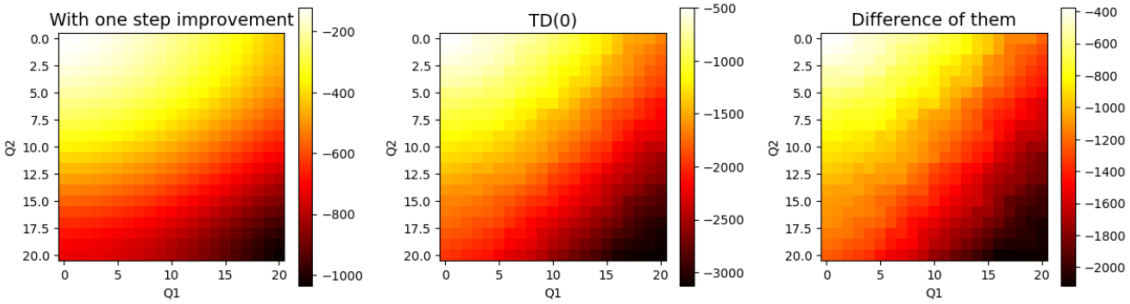

FIGURE 17 – Comparaison entre le TD(0) et la politique aléatoire avec une amélioration d'une étape, $\alpha = \frac{1}{\sqrt{\sqrt{n}}}$

Enfin, entre le TD(0) et la valeur optimale, nous avons ce graphique :

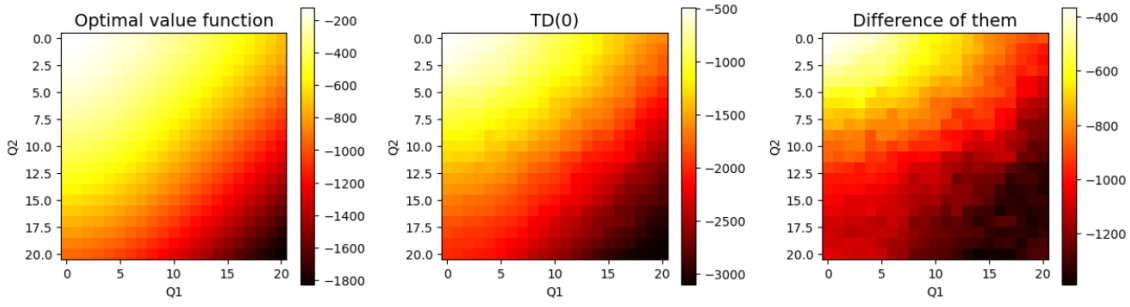

FIGURE 18 – Comparaison entre le TD(0) et la fonction de valeur optimale, $\alpha = \frac{1}{\sqrt{\sqrt{n}}}$

On peut clairement voir que les résultats obtenus par l'algorithme TD(0) ne sont pas très cohérents dans le cas de $\alpha = \frac{1}{n}$. La fonction de valeur trouvée par cette méthode est bien meilleure avec une autre variation où $\alpha_n = \frac{1}{\sqrt{n}}$. Et elle donne ses meilleurs résultats lorsque nous utilisons $\alpha_n = \frac{1}{\sqrt{\sqrt{n}}}$.

2.2 Contrôle optimal

Dans cette partie, on vous demande de trouver la politique optimale pour dispatcher les nouveaux emplois.

2.2.1 Implémentez Q-learning

Pour le Q-learning, nous obtenons les résultats suivants pour $\alpha_n = \frac{1}{n}$:

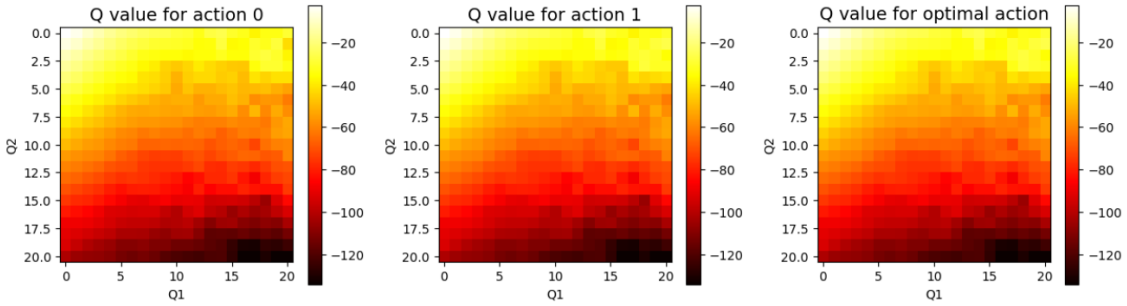

FIGURE 19 – Graphique de la fonction de valeur pour le Q-learning, $\alpha=\frac{1}{n}$

2.2.2 Représentez sur le plan l'action optimale en fonction de l'état (Q_1,Q_2)

Nous avons la représentation suivante de l'action optimale sur le plan :

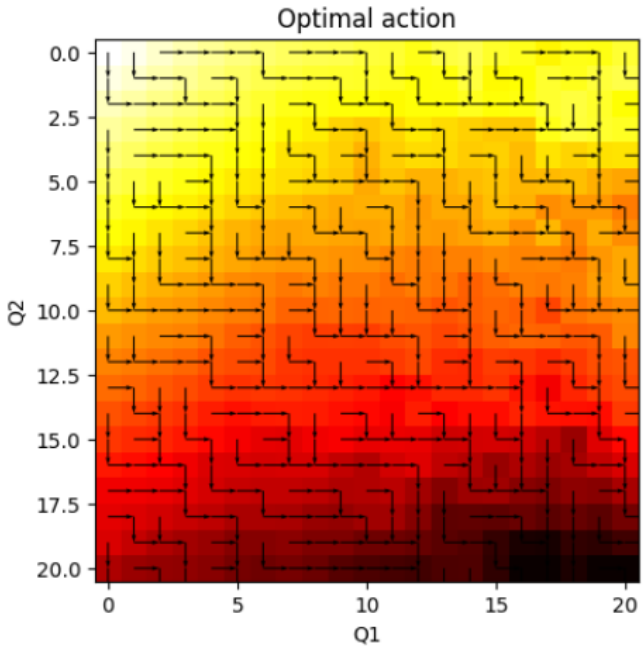

FIGURE 20 – Superposition de la fonction de valeur optimale et de l'action optimale, $\alpha=\frac{1}{n}$

2.2.3 Explorez d'autres alternatives pour α_n, par exemple constante ou $\frac{1}{n^\gamma}$, pour voir si la convergence s'améliore

- Pour $\alpha_n = 1$ (constante), nous avons les résultats suivants pour le Q-learning,

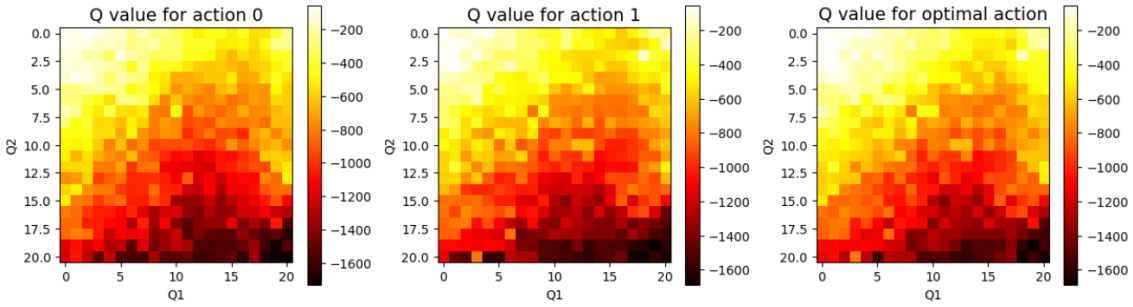

FIGURE 21 – Graphique de la fonction de valeur pour le Q-learning, $\alpha = 1$

Nous avons la représentation suivante de l'action optimale sur le plan :

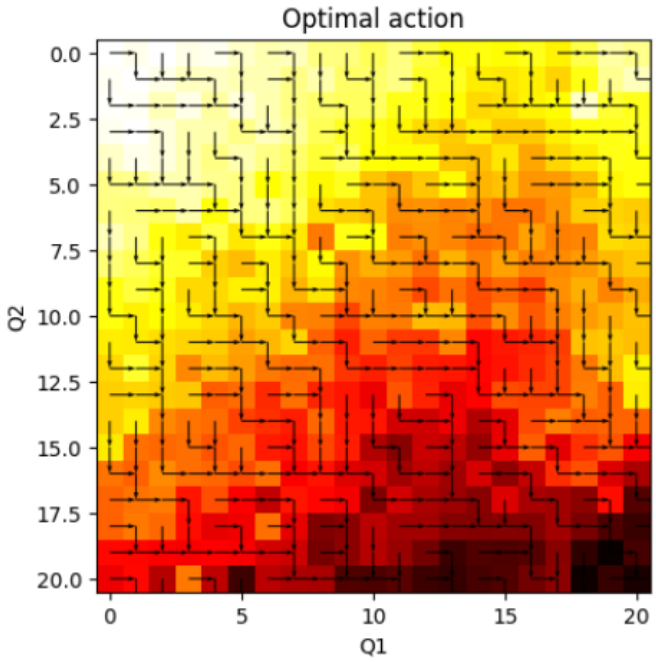

FIGURE 22 – Superposition de la fonction de valeur optimale et de l'action optimale, $\alpha = 1$

• Pour $\alpha_n = \dfrac{1}{n^\gamma}$, nous obtenons les résultats suivants pour le Q-learning,

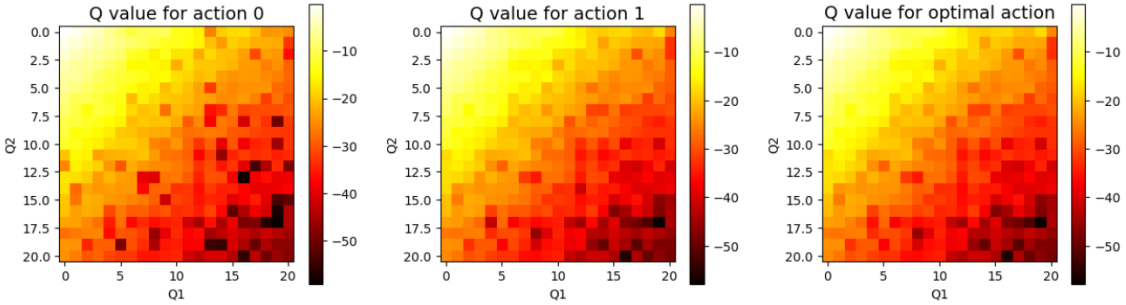

FIGURE 23 – Graphique de la fonction de valeur pour le Q-learning, $\alpha = \dfrac{1}{n^\gamma}$ avec $\gamma = 1.5$

Nous avons la représentation suivante de l'action optimale sur le plan :

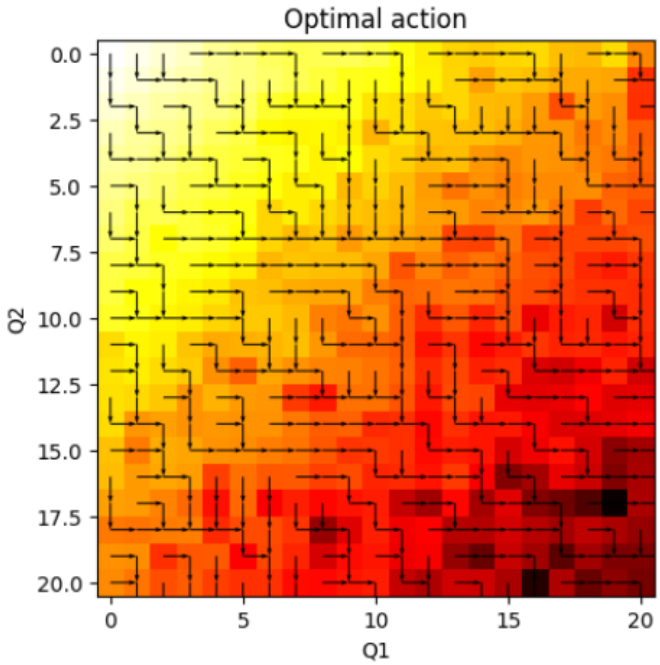

FIGURE 24 – Superposition de la fonction de valeur optimale et de l'action optimale, $\alpha = \frac{1}{n^\gamma}$ avec $\gamma = 1.5$

Le Q-learning est une méthode très utile pour apprendre la politique optimale pour de nombreux problèmes, mais nous pouvons voir qu'il n'est pas le plus adapté pour notre problème actuel. Les résultats du Q-learning sont presque corrects autour de l'origine $(0, 0)$, mais il n'est pas très efficace pour apprendre la politique optimale, surtout plus loin de l'origine. C'est pourquoi nous constatons quelques anomalies autour de la zone $(20, 20)$ de notre graphique.

3 Contrôle sans modèle avec approximation de la fonction de valeur/politique

Cette section est un peu plus ouverte et orientée vers la recherche. On s'attend à ce que vous proposiez et implémentiez votre propre approche. L'objectif est de concevoir un schéma qui peut apprendre la politique optimale de manière plus efficace que dans la section 2 en utilisant une approche non tabulaire. Comme vous l'avez probablement constaté dans la section 2, le Q-learning n'est pas très efficace pour apprendre la politique optimale de ce problème. Certaines approches possibles sont les suivantes :

- Approximation de la fonction de valeur : Approximation linéaire, réseaux neuronaux, etc.
- Approximation de la politique : paramétrisation softmax, ou autre paramétrisation inspirée des résultats structuraux de la section 1.
- Apprentissage par renforcement basé sur un modèle en utilisant les conclusions de la section 1 sur la politique optimale.

Nous décidons de mettre en œuvre l'apprentissage par renforcement basé sur un modèle en utilisant les conclusions de la section 1 sur la politique optimale. En effet, nous avons utilisé la politique optimale, mais la différence est que dans ce cas, nous ne connaissons pas les valeurs des différents paramètres μ_1, μ_2 et λ. L'objectif est d'estimer et d'approximer leurs valeurs au moyen de notre itération en utilisant un simulateur de l'environnement.

Dans notre approche, nous avons utilisé le simulateur de l'environnement pour calculer le nombre de fois où une certaine transition se produit d'un état $(Q1, Q2)$ à un état $(Q1', Q2')$. Nous avons donc utilisé ces nombres pour calculer les probabilités des différentes transitions possibles.

Nous obtenons le graphique suivant de la politique optimale pour $\mu_1 = 0.2$, $\mu_2 = 0.4$ et $\lambda = 0.3$.

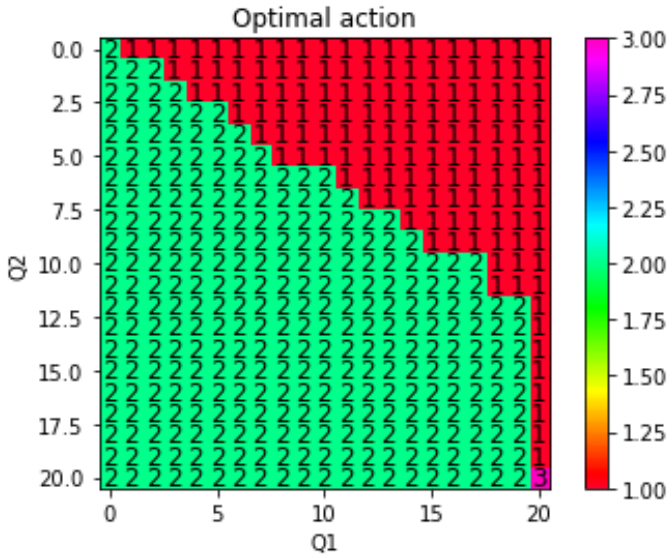

FIGURE 25 – Actions optimales : Extraction de la politique optimale à l'aide de l'apprentissage par renforcement basé sur un modèle

4 Notre code

4.1 Policy Evaluation

```
import numpy as np
import matplotlib.pyplot as plt

lamda = 0.3
mu1 = 0.2
mu2 = 0.4
gamma = 0.99
p1 = 0.5
p2 = 0.5
p_Q1_Q2_a1 = p1 * (1-mu1-mu2-lamda)
p_Q1_Q2_a2 = p2 * (1-mu1-mu2-lamda)

p_Q1p1_Q2_a1 = p1 * lamda
p_Q1_Q2p1_a2 = p2 * lamda

p_Q1m1_Q2_a1 = p1 * mu1
p_Q1m1_Q2_a2 = p2 * mu1

p_Q1_Q2m1_a1 = p1 * mu2
p_Q1_Q2m1_a2 = p2 * mu2
```

```
def value1(epsilon):
    Vn = np.zeros((21,21))
    Vn_p1 = np.zeros((21,21))
    first_boucle = True
    count = 0

    while np.max(abs((Vn_p1 - Vn))) > epsilon or first_boucle:
        count +=1
        first_boucle = False
        action = np.random.randint(0,2)
        Vn = Vn_p1.copy()
        for Q1 in range(21):
            for Q2 in range(21):
                rec = - (Q1 + Q2)

                if Q1 == 20:
                    Vn_Q1p1_Q2 = Vn[Q1,Q2]
                else:
                    Vn_Q1p1_Q2 = Vn[Q1+1,Q2]

                if Q2 == 20:
                    Vn_Q1_Q2p1 = Vn[Q1,Q2]
                else:
                    Vn_Q1_Q2p1 = Vn[Q1, Q2+1]

                if Q1 == 0:
                    Vn_Q1m1_Q2 = Vn[Q1,Q2]
                else:
                    Vn_Q1m1_Q2 = Vn[Q1-1,Q2]

                if Q2 == 0:
                    Vn_Q1_Q2m1 = Vn[Q1,Q2]
                else:
                    Vn_Q1_Q2m1 = Vn[Q1, Q2-1]


                Vn_p1[Q1,Q2] = rec + gamma * (p_Q1_Q2_a1 * Vn[Q1,Q2]
                                    + p_Q1p1_Q2_a1 * Vn_Q1p1_Q2
                                    + p_Q1_Q2p1_a2 * Vn_Q1_Q2p1
                                    + p_Q1m1_Q2_a1 * Vn_Q1m1_Q2
                                    + p_Q1_Q2m1_a1 * Vn_Q1_Q2m1
                                    + (p_Q1_Q2_a2 * Vn[Q1,Q2]
                                    + p_Q1m1_Q2_a2 * Vn_Q1m1_Q2
                                    + p_Q1_Q2m1_a2 * Vn_Q1_Q2m1))

    return Vn_p1, count

epsilon = 1e-5
Value_final, count_final = value1(epsilon)
print(count_final)
```

4.2 Contrôle optimal

```
lamda = 0.3
mu1 = 0.2
mu2 = 0.4
gamma = 0.99

p_Q1_Q2_a1 = (1-mu1-mu2-lamda)
p_Q1_Q2_a2 = (1-mu1-mu2-lamda)

p_Q1p1_Q2_a1 = lamda
p_Q1_Q2p1_a2 = lamda

p_Q1m1_Q2_a1 = mu1
p_Q1m1_Q2_a2 = mu1

p_Q1_Q2m1_a1 = mu2
p_Q1_Q2m1_a2 = mu2

actiontab = np.zeros((21,21))


def optimal_value(epsilon):
    Vn = np.zeros((21,21))
    Vn_p1 = np.zeros((21,21))
    first_boucle = True
    count = 0

    while np.max(abs(Vn_p1 - Vn)) > epsilon or first_boucle:
        count +=1
        first_boucle = False
        action = np.random.randint(0,2)
        Vn = Vn_p1.copy()
        for Q1 in range(21):
            for Q2 in range(21):
                rec = - (Q1 + Q2)

                if Q1 == 20:
                    if Q2 == 20:
                        Vn_Q1p1_Q2 = Vn[Q1,Q2]
                        Vn_Q1_Q2p1 = 0
                        actiontab[Q1,Q2] = 3
                    else:
                        Vn_Q1_Q2p1 = Vn[Q1, Q2+1]
                        Vn_Q1p1_Q2 = 0
                        actiontab[Q1,Q2] = 2
                else:
                    if Q2 == 20:
                        Vn_Q1_Q2p1 = 0
                        Vn_Q1p1_Q2 = Vn[Q1+1 ,Q2]
                        actiontab[Q1,Q2] = 1
                    else:
                        if Vn[Q1, Q2+1] >= Vn[Q1+1,Q2]:
```

```
                            Vn_Q1_Q2p1 = Vn[Q1, Q2+1]
                            Vn_Q1p1_Q2 = 0
                            actiontab[Q1,Q2] = 2
                        else:
                            Vn_Q1_Q2p1 = 0
                            Vn_Q1p1_Q2 = Vn[Q1+1 ,Q2]
                            actiontab[Q1,Q2] = 1

                if Q1 == 0:
                    Vn_Q1m1_Q2 = Vn[Q1,Q2]
                else:
                    Vn_Q1m1_Q2 = Vn[Q1-1,Q2]

                if Q2 == 0:
                    Vn_Q1_Q2m1 = Vn[Q1,Q2]
                else:
                    Vn_Q1_Q2m1 = Vn[Q1, Q2-1]


                Vn_p1[Q1,Q2] = rec + gamma * (p_Q1_Q2_a1 * Vn[Q1,Q2]
                                    + p_Q1p1_Q2_a1 * Vn_Q1p1_Q2
                                    + p_Q1_Q2p1_a2 * Vn_Q1_Q2p1
                                    + p_Q1m1_Q2_a1 * Vn_Q1m1_Q2
                                    + p_Q1_Q2m1_a1 * Vn_Q1_Q2m1)

    return Vn_p1, actiontab, count

epsilon = 0.01
Value_final_optimal, optimal_politic, count = optimal_value(epsilon)
print(optimal_politic)
print(count)
```

4.3 Amélioration en une étape

```
import numpy as np
import matplotlib.pyplot as plt

lamda = 0.3
mu1 = 0.2
mu2 = 0.4
gamma = 0.99
p1 = 0.5
p2 = 0.5
p_Q1_Q2_a1 = p1 * (1-mu1-mu2-lamda)
p_Q1_Q2_a2 = p2 * (1-mu1-mu2-lamda)

p_Q1p1_Q2_a1 = p1 * lamda
p_Q1_Q2p1_a2 = p2 * lamda

p_Q1m1_Q2_a1 = p1 * mu1
p_Q1m1_Q2_a2 = p2 * mu1
```

```
p_Q1_Q2m1_a1 = p1 * mu2
p_Q1_Q2m1_a2 = p2 * mu2


def one_step_improvement(epsilon):
    Vn = np.zeros((21,21))
    Vn_p1 = np.zeros((21,21))
    Vn_p1_a1 = np.zeros((21,21))
    Vn_p1_a2 = np.zeros((21,21))
    first_boucle = True
    count = 0

    while np.max(abs((Vn_p1 - Vn))) > epsilon or first_boucle:
        count +=1
        first_boucle = False
        action = np.random.randint(0,2)
        Vn = Vn_p1.copy()
        for Q1 in range(21):
            for Q2 in range(21):
                rec = - (Q1 + Q2)

                if Q1 == 20:
                    Vn_Q1p1_Q2 = Vn[Q1,Q2]
                else:
                    Vn_Q1p1_Q2 = Vn[Q1+1,Q2]

                if Q2 == 20:
                    Vn_Q1_Q2p1 = Vn[Q1,Q2]
                else:
                    Vn_Q1_Q2p1 = Vn[Q1, Q2+1]

                if Q1 == 0:
                    Vn_Q1m1_Q2 = Vn[Q1,Q2]
                else:
                    Vn_Q1m1_Q2 = Vn[Q1-1,Q2]

                if Q2 == 0:
                    Vn_Q1_Q2m1 = Vn[Q1,Q2]
                else:
                    Vn_Q1_Q2m1 = Vn[Q1, Q2-1]


                Vn_p1[Q1,Q2] = rec + gamma * (p_Q1_Q2_a1 * Vn[Q1,Q2]
                                    + p_Q1p1_Q2_a1 * Vn_Q1p1_Q2
                                    + p_Q1_Q2p1_a2 * Vn_Q1_Q2p1
                                    + p_Q1m1_Q2_a1 * Vn_Q1m1_Q2
                                    + p_Q1_Q2m1_a1 * Vn_Q1_Q2m1
                                    + (p_Q1_Q2_a2 * Vn[Q1,Q2]
                                    + p_Q1m1_Q2_a2 * Vn_Q1m1_Q2
                                    + p_Q1_Q2m1_a2 * Vn_Q1_Q2m1))

    Vn = Vn_p1.copy()
    for Q1 in range(21):
        for Q2 in range(21):
```

```
            rec = - (Q1 + Q2)

            if Q1 == 20:
                Vn_Q1p1_Q2 = Vn[Q1,Q2]
            else:
                Vn_Q1p1_Q2 = Vn[Q1+1,Q2]

            if Q2 == 20:
                Vn_Q1_Q2p1 = Vn[Q1,Q2]
            else:
                Vn_Q1_Q2p1 = Vn[Q1, Q2+1]

            if Q1 == 0:
                Vn_Q1m1_Q2 = Vn[Q1,Q2]
            else:
                Vn_Q1m1_Q2 = Vn[Q1-1,Q2]

            if Q2 == 0:
                Vn_Q1_Q2m1 = Vn[Q1,Q2]
            else:
                Vn_Q1_Q2m1 = Vn[Q1, Q2-1]


            Vn_p1_a1[Q1,Q2] = rec + gamma * (p_Q1_Q2_a1 * Vn[Q1,Q2]
                                + p_Q1p1_Q2_a1 * Vn_Q1p1_Q2
                                + p_Q1m1_Q2_a1 * Vn_Q1m1_Q2
                                + p_Q1_Q2m1_a1 * Vn_Q1_Q2m1)

            Vn_p1_a2[Q1,Q2] = rec + gamma * (p_Q1_Q2p1_a2 * Vn_Q1_Q2p1
                                + p_Q1_Q2_a2 * Vn[Q1,Q2]
                                + p_Q1m1_Q2_a2 * Vn_Q1m1_Q2
                                + p_Q1_Q2m1_a2 * Vn_Q1_Q2m1)

            if Vn_p1_a1[Q1,Q2] >= Vn_p1_a2[Q1,Q2]:
                Vn_p1[Q1,Q2] = Vn_p1_a1[Q1,Q2]
            else:
                Vn_p1[Q1,Q2] = Vn_p1_a2[Q1,Q2]


    return Vn_p1, count

epsilon = 1e-5
Value_final_one_step_imp, count_final_one_step_imp = one_step_improvement(epsilon)
print(count_final_one_step_imp)
```

4.4 Tabular Model-Free Control : TD(0)

```
# Implement TD(0) with probability 0.5 to either queue 1 and 2, and take alpha = 1/n

import numpy as np
import math
```

```
lamda = 0.3

mu1 = 0.2
mu2 = 0.4

gamma = 0.99

p1 = 0.5

p2 = 0.5
# Simulate the environment

def simulate_env(Q1, Q2, action):
    Q1_minus_possible = True
    Q2_minus_possible = True
    Q1_plus_possible = True
    Q2_plus_possible = True
    one_action_done = False

    if Q1 == 20:
        Q1_plus_possible = False
        if Q2 == 20:
            Q2_plus_possible = False
            action = 2
        else:
            Q2_plus_possible = True
            action = 1
    else:
        Q1_plus_possible = True
        if Q2 == 20:
            Q2_plus_possible = False
            action = 0
        else:
            Q2_plus_possible = True

    if Q1 == 0:
        Q1_minus_possible = False
    else:
        Q1_minus_possible = True

    if Q2 == 0:
        Q2_minus_possible = False
    else:
        Q2_minus_possible = True

    if action == 0:
        if np.random.uniform() < lamda and Q1_plus_possible:
            Q1 += 1
            one_action_done = True
    else:
        if np.random.uniform() < lamda and Q2_plus_possible:
            Q2 += 1
            one_action_done = True
```

```
    if np.random.uniform() < mu1 and Q1_minus_possible and not one_action_done:
        Q1 -= 1
        one_action_done = True
    if np.random.uniform() < mu2 and Q2_minus_possible and not one_action_done:
        Q2 -= 1

    return Q1, Q2

# Implement TD(0)

def TD0(alpha, n):
    Vn = np.zeros((21,21))
    Tirage = np.zeros((21,21))
    first_boucle = True
    count = 0
    Q1 = 0
    Q2 = 0

    while count < n:
        count +=1
        for Q1 in range(21):
            for Q2 in range(21):
                Tirage[Q1,Q2] +=1
                alpha = 1/(Tirage[Q1,Q2])
                # alpha = 1/math.sqrt(math.sqrt(Tirage[Q1,Q2]))
                action = np.random.randint(0,2)
                Q1_new, Q2_new = simulate_env(Q1, Q2, action)
                Vn[Q1,Q2] = Vn[Q1,Q2] + alpha * (- (Q1 + Q2) + gamma * Vn[Q1_new,Q2_new] - Vn[Q1,Q2])

                # Q1 = np.random.randint(0,21)
                # Q2 = np.random.randint(0,21)

    return Vn

n = 3000

Value_final_TD0 = TD0(1, n)
print(Value_final_TD0)
```

4.5 Tabular Model-free Control : Q-learning

```
lamda = 0.3

mu1 = 0.2
mu2 = 0.4

gamma = 0.99
epsilon = 0.1


def Q_learning(alpha, n):
    Q_value = np.zeros((21,21,2))
```

```
    Tirage = np.zeros((21,21))
    count = 0
    Q1 = 0
    Q2 = 0

    while count < n:
        count +=1
        Tirage[Q1,Q2] +=1
        alpha = 1/Tirage[Q1,Q2]
        #Deciding the optimal action using epsilon-greedy
        if np.random.rand() < epsilon:
            action = np.random.randint(0,2)
        else:
            action = np.argmax(Q_value[Q1,Q2])

        #Simulating a new state
        Q1_new, Q2_new = simulate_env(Q1, Q2, action)
        #Measuring reward
        reward = - (Q1 + Q2)
        Q_value[Q1,Q2,action] = Q_value[Q1,Q2, action] + alpha * ( reward + gamma * np.max(Q_value[Q1

        Q1 = Q1_new
        Q2 = Q2_new

    return Q_value

n = 1000000

Final_Q_Value = Q_learning(1, n)
print(Final_Q_Value)
```

4.6 Contrôle sans modèle avec fonction de valeur / Approximation de politique

```
# Implement TD(0) with probability 0.5 to either queue 1 and 2, and take alpha = 1/n

import numpy as np
import math

lamda = 0.3

mu1 = 0.2
mu2 = 0.4

gamma = 0.99

# Simulate the environment

def simulate_env(Q1, Q2, action):
    Q1_minus_possible = True
    Q2_minus_possible = True
```

```
    Q1_plus_possible = True
    Q2_plus_possible = True
    one_action_done = False

    if Q1 == 20:
        Q1_plus_possible = False
        if Q2 == 20:
            Q2_plus_possible = False
            action = 2
        else:
            Q2_plus_possible = True
            action = 1
    else:
        Q1_plus_possible = True
        if Q2 == 20:
            Q2_plus_possible = False
            action = 0
        else:
            Q2_plus_possible = True

    if Q1 == 0:
        Q1_minus_possible = False
    else:
        Q1_minus_possible = True

    if Q2 == 0:
        Q2_minus_possible = False
    else:
        Q2_minus_possible = True

    if action == 0:
        if np.random.uniform() < lamda and Q1_plus_possible:
            Q1 += 1
            one_action_done = True
    else:
        if np.random.uniform() < lamda and Q2_plus_possible:
            Q2 += 1
            one_action_done = True

    if np.random.uniform() < mu1 and Q1_minus_possible and not one_action_done:
        Q1 -= 1
        one_action_done = True
    if np.random.uniform() < mu2 and Q2_minus_possible and not one_action_done:
        Q2 -= 1

    return Q1, Q2

# Implement TD(0)


def Model_based_RL(epsilon,It_max = 2000):
    tableau_proba_de_passage = np.zeros((21,21,5))
    total_action = np.zeros((21,21))
    actiontab = np.zeros((21,21))
```

```
Vn = np.zeros((21,21))
Vn_p1 = np.zeros((21,21))
first_boucle = True
count = 0
mu1_estimee = 0
mu2_estimee = 0
lamda_estimee =0

while (np.max(abs(Vn_p1 - Vn)) > epsilon and count <= It_max) or first_boucle:
    count +=1
    first_boucle = False
    action = np.random.randint(0,2)
    Vn = Vn_p1.copy()
    for Q1 in range(21):
        for Q2 in range(21):
            rec = - (Q1 + Q2)

            if Q1 == 20:
                if Q2 == 20:
                    Vn_Q1p1_Q2 = Vn[Q1,Q2]
                    Vn_Q1_Q2p1 = 0
                    actiontab[Q1,Q2] = 3
                else:
                    Vn_Q1_Q2p1 = Vn[Q1, Q2+1]
                    Vn_Q1p1_Q2 = 0
                    actiontab[Q1,Q2] = 2
            else:
                if Q2 == 20:
                    Vn_Q1_Q2p1 = 0
                    Vn_Q1p1_Q2 = Vn[Q1+1 ,Q2]
                    actiontab[Q1,Q2] = 1
                else:
                    if Vn[Q1, Q2+1] >= Vn[Q1+1,Q2]:
                        Vn_Q1_Q2p1 = Vn[Q1, Q2+1]
                        Vn_Q1p1_Q2 = 0
                        actiontab[Q1,Q2] = 2
                    else:
                        Vn_Q1_Q2p1 = 0
                        Vn_Q1p1_Q2 = Vn[Q1+1 ,Q2]
                        actiontab[Q1,Q2] = 1

            if Q1 == 0:
                Vn_Q1m1_Q2 = Vn[Q1,Q2]
            else:
                Vn_Q1m1_Q2 = Vn[Q1-1,Q2]

            if Q2 == 0:
                Vn_Q1_Q2m1 = Vn[Q1,Q2]
            else:
                Vn_Q1_Q2m1 = Vn[Q1, Q2-1]

            Q1_new , Q2_new = simulate_env(Q1,Q2,actiontab[Q1,Q2])
```

```
                if not(Q1 == 0 or Q2 == 0 or Q1 == 20 or Q2 == 20):
                    if Q1_new == Q1:
                        if Q2_new == Q2 + 1:
                            tableau_proba_de_passage[Q1,Q2,1] += 1
                        elif Q2_new == Q2 - 1:
                            tableau_proba_de_passage[Q1,Q2,3] += 1
                        else:
                            tableau_proba_de_passage[Q1,Q2,4] += 1
                    elif Q1_new == Q1 - 1:
                        tableau_proba_de_passage[Q1,Q2,2] += 1
                    elif Q1_new == Q1 + 1:
                        tableau_proba_de_passage[Q1,Q2,0] += 1

                    total_action[Q1,Q2] = np.sum(tableau_proba_de_passage[Q1,Q2,:])
                    mu1_estimee = tableau_proba_de_passage[Q1,Q2,2]/total_action[Q1,Q2]
                    mu2_estimee = tableau_proba_de_passage[Q1,Q2,3]/total_action[Q1,Q2]
                    lamda_estimee = tableau_proba_de_passage[Q1,Q2,0]/total_action[Q1,Q2]


                Vn_p1[Q1,Q2] = rec + gamma * ((1 - mu1_estimee - mu2_estimee - lamda_estimee) * Vn[Q1
                                    + lamda_estimee * Vn_Q1p1_Q2
                                    + lamda_estimee * Vn_Q1_Q2p1
                                    + mu1_estimee * Vn_Q1m1_Q2
                                    + mu2_estimee * Vn_Q1_Q2m1)

    return Vn_p1, actiontab, count, mu1_estimee, mu2_estimee, lamda_estimee, tableau_proba_de_passage

epsilon = 0.01
Value_Model_based, Model_based_politic, count, mu1_estimee, mu2_estimee, lamda_estimee, prob_de_pass
print(Model_based_politic)
print(count)
print(mu1_estimee)
print(mu2_estimee)
print(lamda_estimee)
```

4.7 Conclusions :

Ce projet a été très enrichissant et nous a permis de réviser l'ensemble du cours sur l'apprentissage par renforcement et d'approfondir nos connaissances dans ce domaine. Certains aspects du projet étaient assez difficiles, par exemple le traitement de tous les cas limites lorsque $Q1 = 0$ ou $Q1 = 20$. Mais ces défis ont été surmontés avec succès et nous avons apprécié de travailler sur ce projet dans l'ensemble.