
Commande optimale et apprentissage par renforcement

UNE APPROCHE UNIFIÉE POUR LES PROBLÈMES DE PRISES DE DÉCISIONS SÉQUENTIELLES

préparé par

Pr. Alexandre GIRARD



Dernière mise à jour le 7 août 2024

Préface

Ces notes présentent les diverses approches pour prendre des décisions intelligentes sous un cadre théorique unifié basé sur le principe de la programmation dynamique. Elle vise d'abord à établir les liens entre les approches issues du domaine de l'ingénierie (la science des asservissements et la commande optimale) et les approches issues des sciences informatiques (recherche opérationnelle et l'apprentissage par renforcement) qui ont en fait les même bases mathématiques. Ces notes visent principalement à donner à un lecteur issue du domaine de l'ingénierie les bases pour comprendre et utiliser les approches numériques issues des sciences informatiques.



Capsule vidéo
Série de capsules vidéos associées
<https://youtube.com/playlist?list=PL6adNeJOA8UtNs1NQfAHAzcjHcQixBSnu>

Sources externes utiles :

- Livre de programmation dynamique et commande optimale [Bertsekas, 2017]
- Livre d'introduction à l'apprentissage par renforcement [Sutton and Barto, 2018]
- Note de cours *Underactuated Robotics* [Tedrake, 2023]
- Vidéos d'introduction à l'apprentissage par renforcement [Silver, 2015]

Table des matières

1	Introduction	4
1.1	Introduction	4
1.2	La prise de décision en séquence	4
1.2.1	Un comportement défini par une politique à concevoir	4
1.2.2	Un objectif formulé avec une fonction scalaire cumulative	5
1.3	Tour d'horizon	7
1.3.1	Le problème de commande optimale	7
1.3.2	Le problème d'apprentissage par renforcement	7
2	Programmation dynamique	9
2.1	Formulation du problème	9
2.1.1	Dynamique	9
2.1.2	Politique (Loi de commande)	9
2.1.3	Fonction objectif (coût ou récompense)	10
2.1.4	Contraintes	10
2.1.5	Coût-àvenir	10
2.1.6	Solution optimale	11
2.1.7	Terminologie	12
2.2	Principe d'optimalité	14
2.3	Programmation dynamique excente	14
2.4	Variations sur un thème de Bellman	17
2.5	Forces et limites de la programmation dynamique	18
2.6	Programmation dynamique approximée	18
3	Commande stochastique	19
3.1	Dynamique stochastique	19
3.2	Optimisation de l'espérance	21
3.3	Formulation minimax (commande robuste)	21
3.4	Observations Partielles	22
3.4.1	Espace croyance (draft!)	22
4	Modèles d'évolution	24
4.1	Équations différentielles (temps continus)	24
4.1.1	Conversion en temps discret	25
4.2	Équations de différence	25
4.3	Graphes (états discrets déterministes)	25
4.3.1	Discrétisation de variables continues	26
4.4	Chaînes de Markov (états discrets stochastiques)	26
5	Équation de Bellman	27
5.1	Horizon de temps infini	27
5.2	Équation de Bellman	27
5.3	Équation de Hamilton–Jacobi–Bellman	28

6 Solutions Analytiques	29
6.1 LQR à temps discret	29
7 Algorithmes de planification	30
7.1 Algorithme d'itération de valeurs	30
7.2 Algorithme d'itération de politique	30
8 Algorithmes d'apprentissage	31
8.1 TD-Learning	31
8.2 Q-Learning	31
8.3 Sarsa	31
8.4 Function approximation	31
A Outils mathématiques	32
A.1 Ensembles	32
A.2 Probabilités	32
A.2.1 Probabilité pour une variable discrète	32
A.2.2 Probabilité pour une variable continue	33
A.2.3 Espérance	33
A.2.4 Probabilité jointe et conditionnelle	34
A.2.5 Loi de Bayes	34
A.3 Opérations	34
A.3.1 Minimum/maximum	34
B Approximation de fonctions	35
C Exercices	36
C.1 Formulation du problème	36
C.1.1 Fonction coût vs récompense	36
C.1.2 Politique fonction du temps	36
C.1.3 Politique stochastique	36
C.1.4 Fonction de coût pour un pendule	37
C.2 Programmation dynamique exacte	38
C.2.1 Navigation optimale dans un graphe	38
C.2.2 Loi de commande pour une suspension active	39
C.3 Commande stochastique	39
C.3.1 Loi de commande pour une suspension active II	40
C.3.2 Commande stochastique pour une diva à l'opéra	41
C.4 Commande robuste	42
C.4.1 Commande minimax pour tic-tac-toe	42
C.5 Solutions analytiques	43
C.5.1 Solution LQR par programmation dynamique	43
C.6 Algorithmes de planification	45
C.6.1 Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur	45
C.7 Algorithmes d'apprentissage	47
C.7.1 <i>Q-learning</i> pour une navigation optimale	47
C.7.2 Apprentissage d'une fonction $Q(x, u)$ approximée	48

Chapitre 1

Introduction

1.1 Introduction

L'apprentissage par renforcement est un domaine traitant du développement d'algorithme capable d'apprendre automatiquement des fonctions nommées politiques, qui déterminent pour un agent les actions appropriées en fonction d'observations. Le domaine de la commande optimale traite aussi de méthodes pour résoudre ce même problème, mais typiquement dans un contexte plus structuré où plus d'information est disponible sur l'environnement (ex : équations du mouvement). De façon général, l'apprentissage par renforcement comprend des outils très génériques mais qui offrent souvent peu de garanties, alors que la commande optimale a développé des outils avec des garanties mais pour des situations plus spécifiques et en utilisant souvent des approximations. Un fondement commun est la science de la programmation dynamique qui offre un cadre très général pour traiter les problèmes de prise de décisions en séquence après avoir observé l'état d'un système. Le principe peut être utilisé autant pour analyser un système asservis classique, comme contrôler un bras robot en choisissant la tension appliquée aux moteurs basé sur une observation de sa position, que pour des problèmes probabiliste dans un contexte de finance, comme choisir quand acheter ou vendre une action en observant l'évolution de son prix, ou bien un problème d'intelligence artificielle comme choisir la pièce à déplacer lors d'une partie d'échec en observant la position des pièces sur l'échiquier.



Capsule vidéo

Introduction

<https://youtu.be/1ThWOUmkVyY?si=wWIa0-0YpvR-vYbL>

1.2 La prise de décision en séquence

L'élément central qui unit les problèmes de prise de décision en séquence est que l'objectif est d'influencer l'évolution d'un système dynamique, qu'on appellera aussi l'environnement, grâce à un agent qui choisit continuellement des actions basées sur des observations de l'environnement. On est donc en présence d'un système dynamique en boucle fermée qui évolue dans le temps, comme illustré à la figure 1.1.

1.2.1 Un comportement défini par une politique à concevoir

L'objectif est de concevoir ou choisir une fonction π appelé la politique, qui définit le comportement de l'agent. La politique est la carte qui détermine l'action choisie en fonction de l'état de l'environnement observé :

$$\underbrace{u}_{\text{action}} = \pi(\underbrace{x}_{\text{observation}}) \quad (1.1)$$

où on note u la variable qui représente l'action et x la variable qui représente l'état de l'environnement que l'agent observe. Dans un contexte d'asservissement le terme utilisé pour la fonction π serait la *loi de*

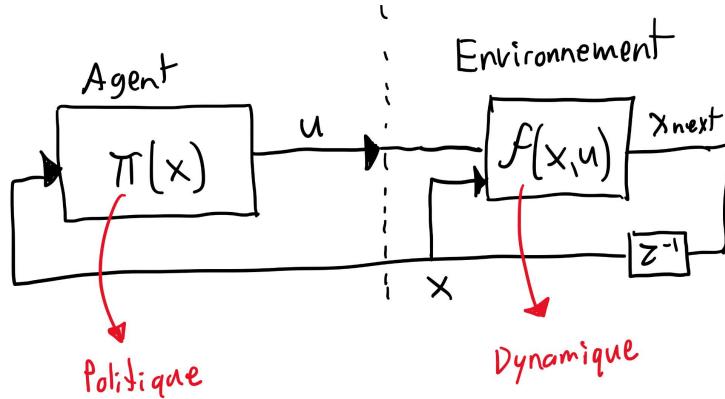


FIGURE 1.1 – Dynamique en boucle fermée avec un agent

commande. La forme de la fonction π peut aller d'une équation analytique, d'un réseau de neurone à un tableau de données en mémoire (look-up table), etc.

1.2.2 Un objectif formulé avec une fonction scalaire cumulative

Pour définir le comportement désiré du système dynamique, l'objectif sera exprimé mathématiquement comme une fonction additive qui dépend de la trajectoire du système et les actions utilisées. Typiquement, dans le domaine de la commande on formule l'objectif comme minimiser une fonction coût et dans le domaine de l'apprentissage par renforcement, on formule l'objectif comme maximiser une fonction récompense. Les deux formulations sont équivalentes et interchangeables (il suffit de changer le signe). Je vais dans ces notes utiliser par défaut la formulation d'une fonction coût qu'on désire minimiser. La fonction sera noté :

$$\underbrace{J}_{\text{Coût cumulatif}} = \underbrace{\sum}_{\text{Somme de coûts instantanés}} g(x, u) \quad (1.2)$$

où J est le coût cumulatif à minimiser et $g(x, u)$ est un coût instantané pour une étape. La forme cumulative de la fonction coût est centrale pour utiliser le principe de la programmation dynamique, mais ce n'est pas vraiment restrictif comme définition, tous les problèmes peuvent être reformulé sous cette forme. Lorsque que notre agent prend les meilleures décisions possibles en fonction de l'objectif on dira que la politique est optimale au sens qu'elle minimise la valeur de la fonction de coût.

Apprentissage machine vs. apprentissage par renforcement Une différence fondamentale par rapport aux autres problèmes d'apprentissage machine, c'est qu'on a pas d'exemples de solution pour apprendre (si on aurait des exemples de bonnes actions à exécuter en fonction d'observation on serait dans une branche appelée apprentissage par imitation, une forme d'apprentissage supervisé). L'agent doit plutôt explorer et apprendre par essai-erreur. Un autre aspect fondamentalement différent, est que les actions ont des conséquences long terme, on ne peut pas évaluer si une action est bonne ou non sans regarder l'évolution long-terme du système dynamique.

Exemple 1. Loi de commande pour un robot

Un exemple d'asservissement classique serait un bras robotique où l'action u déterminée par la politique correspond à un vecteur de couples à appliquer dans les moteurs électriques. Cette action sera calculée en fonction de l'état actuel du robot, donc ici un vecteur de positions et vitesses de ses diverses articulations. L'objectif serait formulé comme la minimisation de l'erreur de position du robot par rapport à une

position cible et potentiellement d'une pénalité pour utiliser beaucoup d'énergie. Typiquement notre solution de politique serait ici une équation analytique.

Exemple 2. Navigation d'un véhicule

Un exemple de prise de décision à plus haut niveau serait de choisir un trajet sur une carte. La loi de commande déterminerait ici quelle direction prendre en fonction de la position actuelle sur la carte. L'objectif d'atteindre la destination le plus rapidement possible pourrait être formuler comme la minimisation du temps écoulé avant d'atteindre celle-ci. La politique (qui serait une solution globale) pourrait être sous la forme d'une table de correspondance (look-up table) où est en mémoire la direction optimale à prendre pour chaque intersection sur laquelle on peut se trouver sur la carte.

Exemple 3. Achats d'une action

Un exemple dans un tout autre contexte serait pour un algorithme d'investissement. L'action de la loi de commande serait ici d'acheter ou non une action en fonction d'une observation de son prix. L'objectif pourrait ici être formuler comme la maximisation des gains financiers. La politique serait ici un seuil de prix, qui pourrait varier en fonction du temps, en dessous duquel l'agent décide d'acheter l'action.

1.3 Tour d'horizon

Une politique optimale va satisfaire une relation appelé l'équation de Bellman, qui peut prendre plusieurs formes selon comment est formulé l'objectif et la méthodologie utilisée pour décrire l'évolution du système. C'est une condition qui garantit l'optimalité de la prise de décision en fonction de la fonction coût qui définit l'objectif. D'un certain point de vue toutes les méthodes de commandes et d'apprentissage par renforcement ont comme objectif de trouver une solution (approximée) à cette équation.

$$J^*(x) = \min_u E [g(x, u) + J^*(f(x, u))]$$

Tâche définie par un coût/récompense

Évolution définie par des équations/échantillons

FIGURE 1.2 – Équation de Bellman, qui caractérise si une politique est optimale

On va voir par exemple, que lorsqu'on modélise l'évolution d'un système dans le domaine continu, comme c'est le cas généralement pour concevoir des asservissements, l'équation peut être réduite à la solution linéaire quadratique (LQR) avec certaines hypothèses. Aussi, l'intelligence artificielle d'un jeu comme les échecs peut aussi être basée sur cette équation lorsqu'on considère des états et actions discrètes. Finalement, en apprentissage par renforcement, le modèle d'évolution qui est utilisé pour approximer cette équation est stochastique et basé sur des échantillons qu'on obtient en observant le système évolué dans le temps.

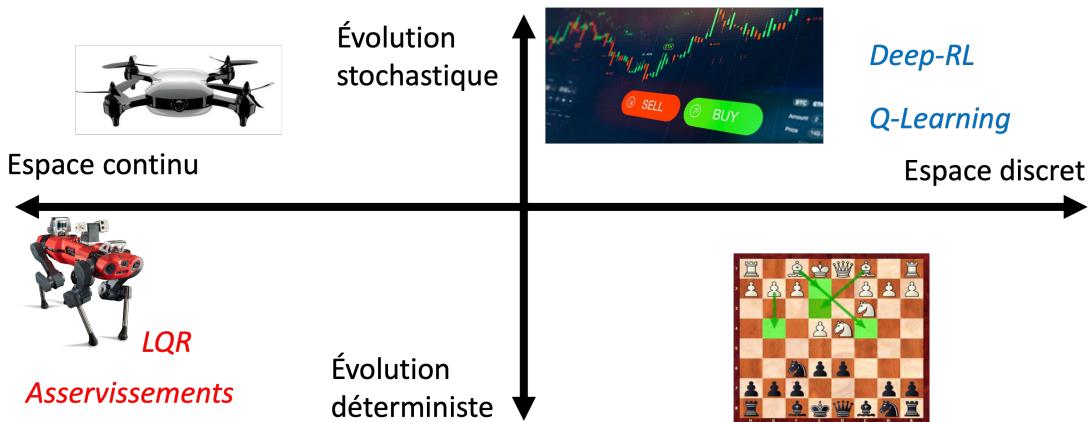


FIGURE 1.3 – Différentes contextes de prise de décision en séquence

1.3.1 Le problème de commande optimale

Dans un problème typique de commande optimale, on va synthétiser d'avance la politique à l'aide d'un modèle de la dynamique et la fonction coût, voir Figure 1.4. Dans la littérature sur l'apprentissage par renforcement, parfois ce problème est nommé *model-based reinforcement learning* ou *planning by dynamic programming*.

1.3.2 Le problème d'apprentissage par renforcement

Dans un problème typique d'apprentissage par renforcement, on assume ne pas connaître initialement la fonction coût ni la dynamique de l'environnement. Le modèle va être appris (plus ou moins directement selon la méthode) en observant les interactions entre l'agent et l'environnement, et la politique va être mise à jour en conséquence, voir Figure 1.5. Dans le domaine de la commande des variantes de ce problème sont nommées la commande adaptative ou l'identification de système.

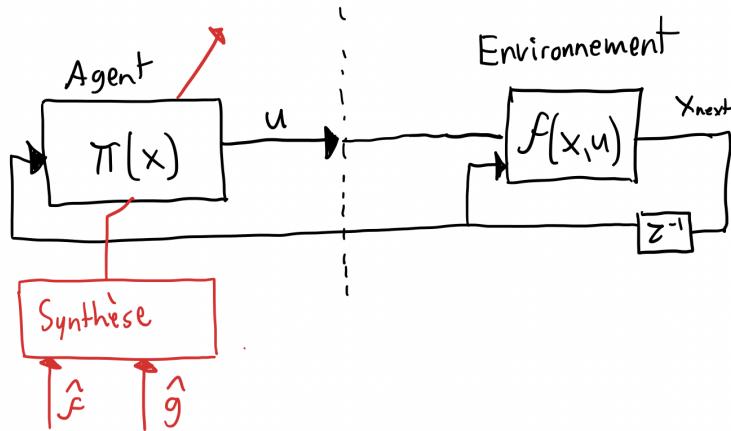


FIGURE 1.4 – Apperçu du problème de commande optimale

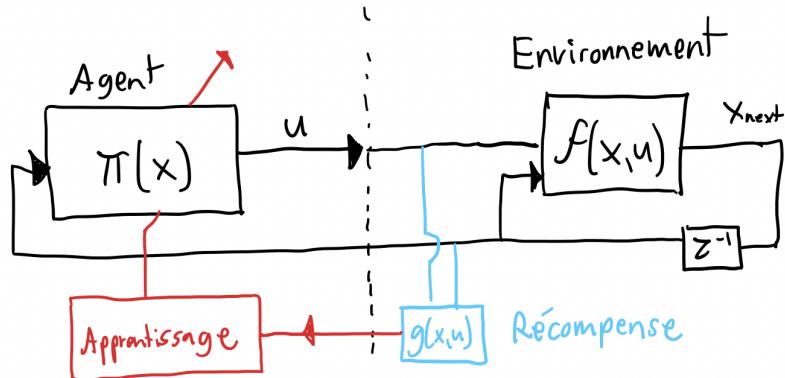


FIGURE 1.5 – Apperçu du problème d'apprentissage par renforcement

Exploration vs. exploitation

Un sous problème qui apparaît dans ce contexte est de balancer l'exploration (pour potentiellement découvrir une meilleure façon d'optimiser le coût), versus simplement exploiter la meilleure solution découverte à ce jour.

Chapitre 2

Programmation dynamique

Dans ce chapitre, les fondements mathématiques de la science de la prise de décisions séquentielles sont introduites. La formulation mathématique de base, qui sera utilisée pour introduire les principes et les différentes approches de solution, est une approche en temps discret où on va considérer qu'on veux optimiser une politique pour un nombre fini de N d'étapes futurs. De plus, on va débuter avec l'hypothèse que l'agent observe directement l'état de l'environnement.

Note sur l'ordre de présentation des concepts Dans ces notes je présente d'abord les concepts et algorithmes dans le contexte d'une évolution déterministe et d'un horizon de temps fini. On pourrait considérer que c'est un petit détour, plusieurs autres ouvrages présentent directement les concepts dans le contexte de chaînes de Markov (évolution probabiliste) et d'un horizon de temps infini. Toutefois, je crois que les concepts de base sont mieux introduits ainsi, et de plus, avec cette formulation les liens sont plus direct à faire avec les concepts de la science des asservissements. On va donc débuter avec une représentation mathématique basée sur des équations de différence déterministes, et introduire l'aspect probabiliste dans un deuxième temps. On traitera plus tard les situations où l'horizon de temps est infini (Chapitre 5), l'évolution est stochastique (Chapitre 3) et quand l'agent a accès à une observation partielle ou bruité de l'état (Chapitre 3.4).

2.1 Formulation du problème

Voici les ingrédients de base pour analyser mathématiquement un problème de prise de décision en séquence :

2.1.1 Dynamique

L'évolution dynamique du système est représentée par une équation de différence :

$$x_{k+1} = f_k(x_k, u_k) \quad k = 0, 1, \dots, N - 1 \quad (2.1)$$

où x est l'état du système, k un indice représentant le temps ou l'étape, et u une variable représentant les actions que l'agent peut prendre. C'est la représentation de comment l'environnement évolue.

2.1.2 Politique (Loi de commande)

La politique est une fonction noté π , qui dictent l'action u à prendre lorsque l'état du système x est observé. De façon général on peut avoir une politique spécifique pour chaque étape k .

$$u_k = \pi_k(x_k) \quad (2.2)$$

Un politique qui change en fonction de l'étape, peut être interprétée comme une politique qui varie dans le temps. On va noter π sans indices l'ensemble des fonctions politiques pour toutes les étapes :

$$\pi = \{\pi_0, \dots, \pi_{N-1}\} \quad (2.3)$$

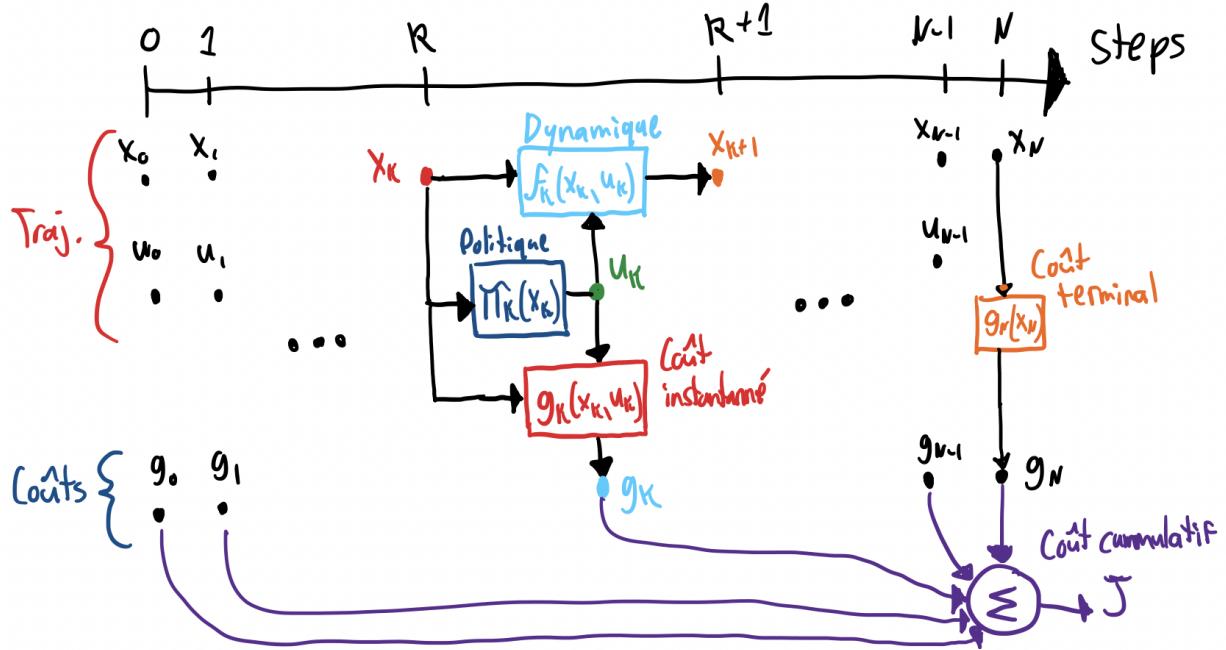


FIGURE 2.1 – Évolution du système en boucle fermée étape par étape.

2.1.3 Fonction objectif (coût ou récompense)

La fonction objectif, dépend des états et action exécutées sur une trajectoire, est définie par :

$$J(\underbrace{x_0, \dots, x_N, u_0, \dots, u_{N-1}}_{\text{Trajectoire}}) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N) \quad (2.4)$$

où N est l'horizon qui représente ici un nombre d'étape, g_k la fonction de coût instantané à l'étape k et g_N une fonction coût terminale qui dépend de l'état final x_N .

2.1.4 Contraintes

Il est souvent nécessaire pour bien caractériser l'objectif, d'inclure des contraintes dures dans le formulation sur problème, on va donc considérer qu'à une étape k , il y a un certain ensemble X_k d'états permis et un ensemble U_k d'action possible, qui peut dépendre de l'état actuel :

$$x_k \in X_k \quad u_k \in U_k(x_k) \quad (2.5)$$

On va appeler une politique admissible si elle respecte les contraintes, et l'ensemble des politiques admissibles est noté :

$$\pi \in \Pi \quad (2.6)$$

2.1.5 Coût-àvenir

Un concept important est celui du coût-àvenir, noté $J_\pi(x)$, qui représente le coût cumulatif total associé à débuter à l'état x et exécuter la politique π sur un horizon N :

$$J_\pi(x) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N) \quad \text{avec } x_0 = x \quad \text{et } x_{k+1} = f_k(x_k, \pi_k(x_k)) \quad (2.7)$$

Notez ici que le coût-àvenir est une fonction seulement de l'état actuel, c'est le coût prévu de la trajectoire future, i.e. le coût à venir.

Important ! Prenez le temps de prendre un bon café et de bien comprendre la définition du coût-àvenir. C'est une notion fondamentale sur lequel le principe de la programmation dynamique est bâtit. De plus, une grande catégorie d'algorithmes d'apprentissage par renforcement ont comme principe de base d'essayer d'approximer cette fonction avec un réseau de neurones.

2.1.6 Solution optimale

La politique optimale est définie par celle qui minimise le coût-àvenir. On va noter J^* la fonction coût-àvenir optimale et π^* la politique optimale.

$$J^*(x) = \min_{\pi \in \Pi} J_\pi(x) \quad (2.8)$$

$$\pi^* = \arg \min_{\pi \in \Pi} J_\pi(x) \quad (2.9)$$

(2.10)

2.1.7 Terminologie

Variable	Termes	Définition
$f_k(x_k, u_k)$	Dynamique, Système, Environnement, Processus, <i>Plant</i> ,	Équations qui définit l'évolution du système, i.e. de l'environnement de l'agent.
$u = \pi_k(x)$	Loi de commande, contrôleur, politique de l'agent, <i>policy</i>	Fonction qui définit la décision de l'agent comme une fonction de l'observation de l'état.
x	État, <i>State</i>	Variable qui représente toute l'information pour prédire l'évolution future du système.
u	Action, décision, entrée du système, <i>control input</i>	Variable qui représente la décision de l'agent.
k	index	Entier correspondant à l'étape actuelle (i.e. souvent représentant le temps discréte).
$U_k(x)$	Contraintes, <i>control set</i>	Ensemble représentant les actions qui sont possible lorsque l'état du système est x à l'étape k
$J(x_0, \dots, x_N)$	Fonction de coût cumulative, (opposé de la) fonction de récompense, <i>value function</i>	Fonction scalaire qui représente à quel point une trajectoire est bonne en fonction de l'objectif.
$J_\pi(x)$	Coût à venir, <i>cost-to-go</i>	Fonction scalaire qui représente la prédition de coût cummulative d'une trajectoire qui débute à x en utilisant la loi c
$J^*(x)$	Coût à venir optimal	Fonction scalaire qui représente la prédition de coût cummulative d'une trajectoire qui débute à x si toutes les actions dans le futur sont optimales
$g_k(x_k, u_k)$	Coût instantanée	Fonction scalaire qui définit le coût instantané à l'étape k
$g_N(x_N)$	Coût terminal	Fonction scalaire qui définit le coût final en fonction de l'état terminal.

Exemple 1. Pendule Simple en temps minimum

Le concept de coût-àvenir ce visualise bien avec un problème de temps minimal. Supposons qu'on s'intéresse à positionner un pendule en appliquant un couple à la base et que notre critère c'est de ce rendre à la position cible le plus vite possible. La fonction coût-àvenir $J_\pi(x)$ représenterait le temps-àvenir, i.e. le temps prévu avant d'atteindre la cible, lorsqu'on débute à l'état x . La figure 2.2, représente une temps-àvenir optimal calculé numériquement.

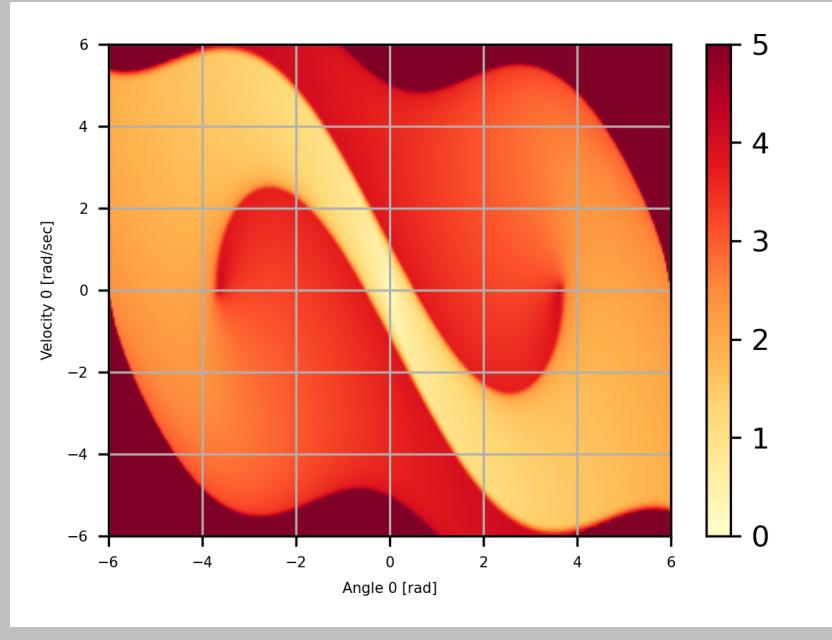


FIGURE 2.2 – Coût-àvenir optimal pour un pendule (temps-minimum)

2.2 Principe d'optimalité

Le principe d'optimalité formalise une notion qui peut sembler assez simple :

Si une trajectoire est optimale de x_0 à x_N en passant par x_i . Alors nécessairement la séquence de fin de cette trajectoire correspond aussi à la solution optimale d'une trajectoire qui débuterait à x_i pour aller à x_N .

$$[x_0, \dots, x_i, \dots, x_N] \quad (2.11)$$

$$[x_i, \dots, x_N] \quad (2.12)$$



Capsule vidéo

Le principe d'optimalité

<https://youtu.be/EMkpkMTg4U?si=eDITgpq50NhRD8-f>

Par exemple, disons qu'on a trouvé le chemin optimal entre Montréal et Québec, et que ce chemin passe par Drummondville. Alors, le principe d'optimalité dit que le chemin optimal entre Drummondville et Québec est nécessairement la séquence de fin du chemin optimal entre Montréal et Québec. Le principe de la programmation dynamique est d'exploiter ces séquences de queue dans une boucle récursive.

2.3 Programmation dynamique excente

L'algorithme de programmation dynamique permet de résoudre exactement le problème de décision en séquence formulé à la section 2.1. L'idée est de partir de la fin, et de calculer de le coût-àvenir en reculant étape par étape :

$$J_N^*(x_N) = g_N(x_N) \quad \forall x_N \in X_N \quad (2.13)$$

$$\vdots \quad (2.14)$$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^* \underbrace{(f_k(x_k, u_k))}_{x_{k+1}} \right] \quad \forall x_k \in X_k \quad (2.15)$$

$$\vdots \quad (2.16)$$

$$J_0^*(x_0) = \min_{u_0 \in U_0(x_0)} \left[g_0(x_0, u_0) + J_1^* \underbrace{(f_0(x_0, u_0))}_{x_1} \right] \quad \forall x_0 \in X_0 \quad (2.17)$$

La politique optimale est calculé simultanément en gardant en mémoire l'action qui minimise le coût-àvenir :

$$\pi_k^*(x_k) = \operatorname{argmin}_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^* \underbrace{(f_k(x_k, u_k))}_{x_{k+1}} \right] \quad \forall x_k \in X_k \quad (2.18)$$

Si on décortique, chaque expression dans l'opération *min* représente un coût instantané plus le coût-àvenir de se retrouver sur le prochain état à l'étape suivante, pour une option d'action. La politique optimale est de choisir l'action qui minimise cette expression et le coût-àvenir optimal est la valeur minimale :

$$\underbrace{J_k^*(x_k)}_{\text{Coût-àvenir optimal}} = \underbrace{\min_{u_k \in U_k(x_k)}}_{\text{Minimum possible}} \left[\underbrace{g_k(x_k, u_k)}_{\text{Coût instantané}} + \underbrace{J_{k+1}^* \underbrace{(f_k(x_k, u_k))}_{x_{k+1}}}_{\text{Coût-àvenir optimal à l'étape suivante}} \right] \underbrace{\text{Valeur } Q_k^*(x_k, u_k)}_{(2.19)}$$

La valeur de la parathèse, notée $Q_k^*(x_k, u_k)$ est souvent appelée la *valeur-Q* et est une quantité centrale en apprentissage par renforcement. Elle correspond au coût-àvenir de choisir une action u_k à l'état x_k . Un algorithme de base en apprentissage par renforcement ce nomme Q-Learning et a comme principe d'apprendre ces valeurs.



Capsule vidéo
Algorithme de programmation dynamique
<https://youtu.be/dfz9k3BGrH0?si=MBxJzpKPOUjnaERe>

J vs Q La valeur $J^*(x)$ est une fonction qui donne le coût-àvenir optimal pour un état donné en assumant qu'on choisi l'action optimal à partir de ce point. La valeur $Q^*(x, u)$ est une fonction qui donne le coût-àvenir optimal en assumant que l'action u à déjà été sélectionnée, et que par la suite les actions seront optimales.

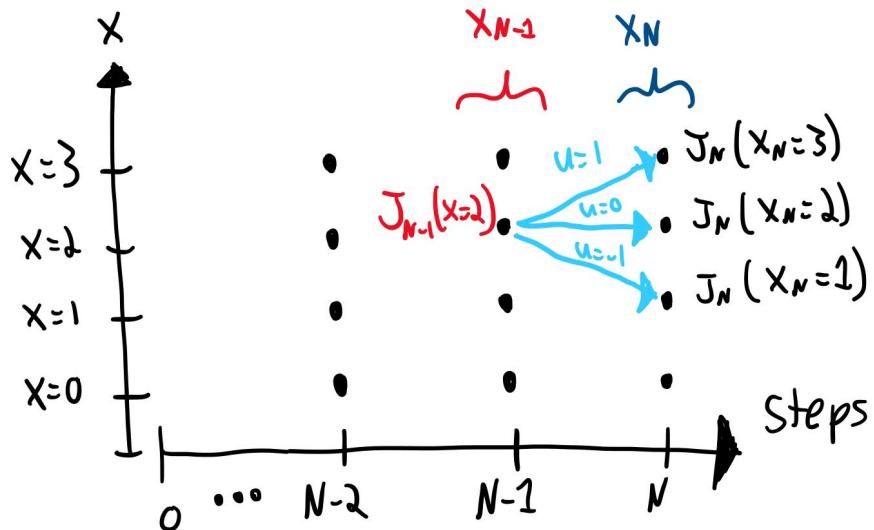


FIGURE 2.3 – Exemple de calcul de coût-àvenir pour l'état $x = 2$ à l'étape $k = N - 1$. Ici pour cet exemple on a quatres états discrets possibles et trois actions possibles. Comme illustré, on doit dabord avoir évalué le coût terminal de tout les états à l'étape $k = N$, ensuite on calcul de cout-àvenir de chaque action possible et on minimise.

Exemple 2. Navigation optimale



Capsule vidéo
Exemple de navigation optimale sur un graphe
<https://youtu.be/1GXUNWVgZOU?si=P4hDvWSWoau6nW4x>

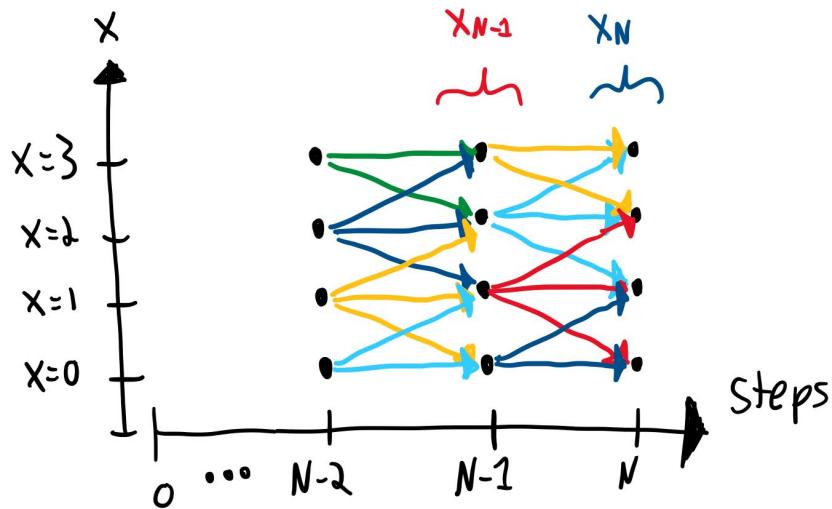


FIGURE 2.4 – L'algorithme de programmation dynamique, consiste en executant ce calcul pour chaque état à chaque étape en débutant par la fin. Ici chaque couleur représente une étape de minimisation pour un état. Il y a une valeur $Q(x, u)$ pour chaque état-action qui sont ici les arcs, et une valeurs $J(x)$ pour chaque état qui sont les noeuds.

Exemple 3. Chauffage optimale



Capsule vidéo

Exemple pour une politique de chauffage optimale

<https://youtu.be/QuXjiAzDENs?si=mQcKeWkjxUU-bBuM>

Exemple 4. Pendule inverse



Exercice de code

Démo de programmation dynamique

https://colab.research.google.com/drive/1mcExyc25P_0im4iU_wVEFnkzgw4UTVaq?usp=sharing

2.4 Variations sur un thème de Bellman

L'algorithme de programmation dynamique a plusieurs variantes, voici un petit tour d'horizon rapide :

Stochastique

Si l'environnement est stochastique on va rajouter un calcul de l'espérance pour optimiser une moyenne pondérée des coûts futur possible :

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{\mathbf{w}_k} \left[g_k(x_k, u_k, \mathbf{w}_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, \mathbf{w}_k)}_{x_{k+1}} \right) \right] \quad (2.20)$$

Robuste

Dans certaines situation, on peut préféré une formulation où l'espérance est remplacer par un *max*. C'est en fait la formulation *minimax* utilisée dans plusieurs algorithme de jeux comme les échecs :

$$J_k^*(x_k) = \min_{u_k} \max_{\mathbf{w}_k} \left[g_k(x_k, u_k, \mathbf{w}_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, \mathbf{w}_k)}_{x_{k+1}} \right) \right] \quad (2.21)$$

À horizon de temps infini

Dans la plupart des situations on va s'intéresser aux politiques optimales lorsque l'horizon de temps n'est pas limité et inclure un facteur qui priorise le futur proche vs. le futur lointain :

$$J^*(x) = \min_u \left[g(x, u) + \alpha J^* \left(\underbrace{f(x, u)}_{x_{k+1}} \right) \right] \quad (2.22)$$

Sans modèles (apprentissage par renforcement)

Si on n'a pas de modèle dynamique de l'environnement, on va préféré travailler avec des valeurs Q , représentant un coût-à-venir de choisir une action à un certain état :

$$Q^*(x, u) = g(x, u) + \min_{u_{k+1}} \left[Q^* \left(\underbrace{f(x, u)}_{x_{k+1}}, u_{k+1} \right) \right] \quad (2.23)$$

À temps continu

La programmation dynamique a un équivalent en temps continu, qui est en fait une équation différentielle partielle :

$$-\frac{\partial J^*(x, t)}{\partial t} = \min_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (2.24)$$

2.5 Forces et limites de la programmation dynamique

Forces

- **Formulation très flexible** : On peut inclure toute forme de fonction d'évolution, coût et contraintes : des non-linéarités dures, des variables discrètes et même stochastique.
- **Solution globale** : On trouve la solution globale au problème

Limites

- **Malédiction des dimensions** : La mise en oeuvre de l'algorithme de programmation dynamique exact est possible seulement pour des problèmes de petites dimensions.
- **Modèle du système** : Il faut connaître les équations de la dynamique du système.

2.6 Programmation dynamique approximée

À venir !

$$u^* = \arg \min_u \mathbb{E} \left[g(x, u, w) + J_{k+1}^*(x_{k+1}) \right]$$

↙ *discretisation des actions*
 ↙ *Monte carlo*
 ↙ *Calcul déterministe*
 ↘ *discretisation
doggégration*
 ↘ *Approximation hors-ligne
(deep reinforcement learning)*
 ↘ *Recherche en-ligne
(Rollout, MPC, etc.)*

FIGURE 2.5 – Différentes stratégies pour approximer la programmation dynamique exacte.



Capsule vidéo
Méthodes approximatives
<https://youtu.be/jWP9yiIL7gY?si=jLFLnHSRSpxuxS1I>

Chapitre 3

Commande stochastique

"The true Logic for this world is the calculus of probabilities"
– James Clerk Maxwell

3.1 Dynamique stochastique

La formulation du problème de décisions séquentielles peut être légèrement modifiée pour représenter une évolution stochastique. Une représentation possible d'une évolution stochastique, est de considérer que la fonction dynamique a une entrée additionnelle qui est une variable aléatoire w_k , que l'on peut interpréter comme une perturbation :

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad (3.1)$$

ou la variable w_k appartient à un ensemble $W_k(x)$:

$$w_k \in W_k(x) \quad (3.2)$$

Un modèle dynamique de l'environnement inclut donc les valeurs possibles de la perturbation w et la probabilité associée à chacune de ces valeurs. Ce modèle a la forme de probabilités (possiblement conditionnelles à l'état, l'action et l'étape actuelle) si w_k prend des valeurs discrète, ou une fonction de densité de probabilité

$$P(w_k|x_k, u_k, k) \quad \text{ou} \quad p(w_k|x_k, u_k, k) \quad (3.3)$$

Note Si vos notions de probabilité sont endormies, l'annexe A.2 présente une synthèse des notions de probabilités importantes pour cette section.



Capsule vidéo
Commande stochastique
<https://youtu.be/wvwrxsCbGwU?si=cqnH8BDjGGBLmx15>

Ce modèle d'évolution stochastique est une représentation possible de ce qu'on appelle une *Chaîne de Markov*. En effet, il est possible de convertir les équations (3.1) et (3.3) en probabilités de transitions, la représentation habituelle d'une chaîne de Markov :

$$P(x_{k+1}|x_k, u_k, k) \quad (3.4)$$

puisque :

$$P(x_{k+1} = f_k(x_k, u_k, w_k)|x_k, u_k, k) = P(w_k|x_k, u_k, k) \quad (3.5)$$

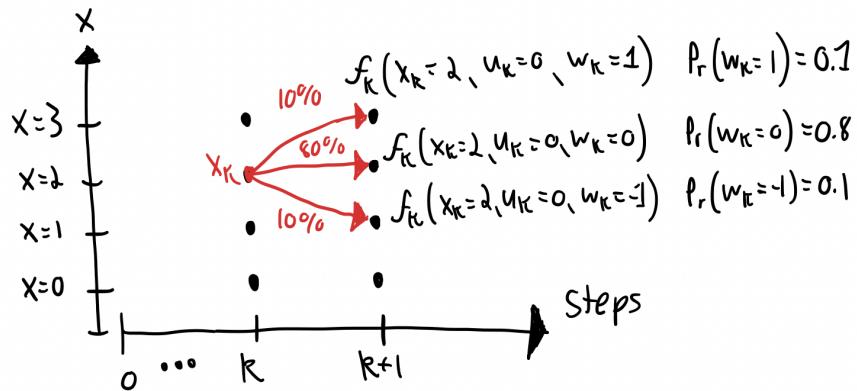


FIGURE 3.1 – Évolution stochastique avec des domaines discrets

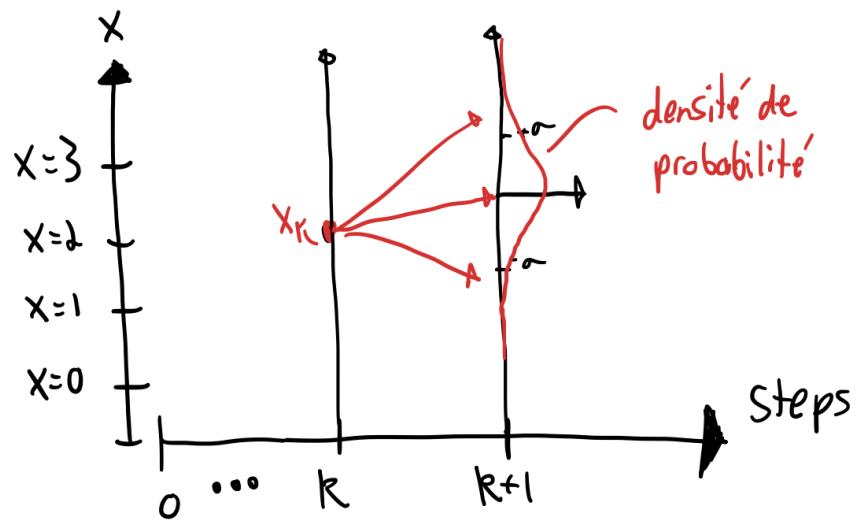


FIGURE 3.2 – Évolution stochastique avec des domaines continus

C'est un choix de modélisation de définir l'environnement directement par des probabilités de transition (équation (3.4)) ou bien une équation dynamique (équation (3.1)) plus des perturbations probabilistes (équation (3.3)).

Propriété de Markov Les fondements théoriques de la programmation dynamique sont basés sur l'hypothèse que l'état observé x a la propriété de Markov, c'est à dire que le l'état x est choisi de sorte à contenir toute l'information nécessaire pour prédire l'évolution future du système. Cette définition est consistante avec la définition d'un vecteur d'état utilisé dans le domaine de la dynamique et la commande. Formellement, dans un contexte probabiliste cette propriété est équivalente à dire que la probabilité de transitionner sur un état futur x_{k+1} , conditionnelle à l'état x_k et l'action u_k , est la même que la probabilité conditionnelle à tout l'historique du système :

$$P(x_{k+1} | x_k, u_k, k) = P(x_{k+1} | x_k, u_k, k, \underbrace{x_{k-1}, u_{k-1}, x_{k-2}, u_{k-2}, \dots, x_1, u_1, x_0, u_0}_{\text{historique}}) \quad (3.6)$$

Autrement dit, tout l'information possible sur l'environnement est compris dans l'état, il n'y a pas de variables cachées qui peuvent influencer le futur.

3.2 Optimisation de l'espérance

La formulation la plus standard pour un problème de décision séquentielles dans un contexte stochastique, qu'on appelle dans la littérature un processus de décision de Markov (MDP), est de chercher à minimiser l'espérance du coûts-àvenir. On modifie donc la définition de l'équation (2.7), pour rajouter un calcul de l'espérance par rapport à toutes les valeurs possibles que peuvent prendre les perturbations :

$$J_\pi(x) = \mathbb{E}_{w_0, w_1, \dots, w_{N-1}} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad \text{avec } x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k), w_k) \quad (3.7)$$

Ce qui représente une moyenne des coût-àvenir possible, pondérés en fonction de leurs probabilités d'occurrence. Le problème à résoudre devient alors de trouver la politique qui minimise cette définition stochastique du coût-àvenir donnée par l'équation (3.7). Par chance, on peut réutiliser l'approche récursive de programmation dynamique qui débute par la fin, il suffit de modifier l'algorithme de programmation dynamique exacte pour ajouté un calcul d'espérance sur le coût-àvenir de chaque option d'action :

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \right) \right] \quad (3.8)$$

$$u_k^*(x_k) = \operatorname{argmin}_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \right) \right] \quad (3.9)$$

3.3 Formulation minimax (commande robuste)

Une autre formulation possible, associé aux concepts de commande robuste, est de plutôt vouloir minimiser le pire scénario possible, i.e. minimiser le coût- à-venir avec la formualtion suivante :

$$J_\pi(x) = \max_{w_0, w_1, \dots, w_{N-1}} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad \text{avec } x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k), w_k) \quad (3.10)$$

Similairement, l'algorithme de programmation dynamique est modifié en incluant une maximisation (remplaçant l'espérance) :

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \quad (3.11)$$

$$u_k^*(x_k) = \operatorname{argmin}_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \quad (3.12)$$

Cette formulation est utilisée typiquement pour les intelligences artificielles de jeux comme les échecs. En effet, dans ce contexte les actions de l'adversaire peuvent être vues comme des perturbations, et puisqu'on assume que l'adversaire va choisir des actions pour nuire à l'autre jouer, il est plus logique d'assumer que la perturbation va nous nuire le plus possible.

3.4 Observations Partielles

Si l'observation de l'agent n'est pas l'état complet de l'environnement, ou est bruité, le problème prend une dimension de complexité supplémentaire. On peut modéliser cette situation avec une fonction observation, pour représenter le lien entre les états et l'observation :

$$y_k = h_k(x_k, u_k, v_k) \quad (3.13)$$

où v_k est une variable aléatoire représentant du bruit, voir Figure 3.3.

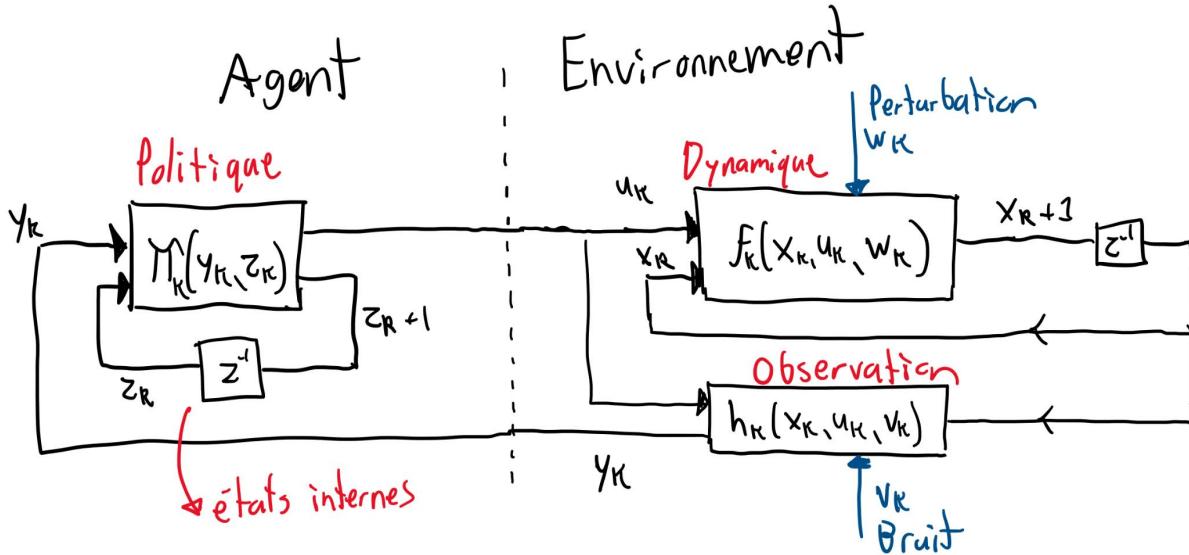


FIGURE 3.3 – Formulation mathématique incluant une fonction observation (utile pour représenter quand l'agent n'observe pas les états directement) et politique qui contient ses propres états et sa dynamique.

Le problème est appelé dans la littérature processus de décision markovien partiellement observable (POMDP), lorsque les états sont discrets. Dans la littérature sur la commande, on nomme souvent les observations les sorties du système.

3.4.1 Espace croyance (draft !)

Le problème générique peut être structuré en remplaçant l'état x (qu'on ne connaît pas avec certitude) par un espace de croyance (*belief state*) qui représente la distribution de probabilité de se trouver sur un état x

basé sur les observations antérieures y_k :

$$Pr(x_k = x_i | u_k, y_k, \dots, u_1, y_1, u_0, y_0) \quad (3.14)$$

On va qualifier de **statistique suffisante** une représentation b de cet espace, si elle rencontre la propriété suivante :

$$P(x_{k+1} | b_k, u_k, k) = P(x_{k+1} | b_k, u_k, k, \underbrace{y_{k-1}, u_{k-1}, y_{k-2}, u_{k-2}, \dots, y_1, u_1, y_0, u_0}_{\text{historique}}) \quad (3.15)$$

Autrement dit, si elle encode toute l'information possible sur l'évolution possible du système dynamique. La bonne nouvelle c'est qu'on peut convertir le problème original (POMDP) en un problème régulier (MDP) en considérant que l'état du système est la représentation de l'espace de croyance b . On peut utiliser les outils habituels et la politique optimale peut être alors exprimée comme une carte statique à partir de cet espace croyance :

$$u_k = \pi_k^*(b_k) \quad (3.16)$$

Il est toutefois à noter, le processus de mettre à jour l'espace croyance en fonction de nouvelles observations est un processus dynamique qui doit être mis en oeuvre par l'agent. Jusqu'à maintenant on assumait pouvoir observer directement l'état de l'environnement et la politique optimale était toujours une fonction statique entre les observations et les actions. Avec des observations partielles, ce n'est plus le cas, la politique optimale va avoir sa propre dynamique et ses états internes, qui est associée avec le processus de mise à jour de la distribution de probabilité de l'espace de croyance.

La mise à jour de l'espace de croyance, peut être vue théoriquement comme implémenter la règle de Bayes (voir Section A.2.5). Je dit théoriquement car en pratique c'est un problème très dur généralement insoluble exactement même sur des problèmes très simples. Il y a toutefois des outils pour des situations particulières, par exemple le filtre de Kalman est une solution à ce problème lorsque le système est linéaire et avec du bruit gaussien. Donc théoriquement la mise à jour de la croyance basée sur des nouvelles observations est :

$$P(x_{k+1} | y_k) = \frac{P(y_k | x_{k+1}) P(x_{k+1})}{P(y_k)} \quad (3.17)$$

Détails à venir !

Chapitre 4

Modèles d'évolution

Dans ce chapitre on va faire des liens entre les différences représentations possible de l'environnement, des équations différentielles qu'on utilise quand on modélise un robot basé sur des principes physiques, jusqu'aux tenseurs de probabilités utilisé dans le contexte de processus de décision de Markov.

4.1 Équations différentielles (temps continus)

Lorsqu'on modélise le mouvement d'un système physique, sur lequel nos actions sont des forces, on va naturellement travailler en temps continu. En effet, lorsqu'on applique les lois de Newton et de conservation sur des systèmes mécaniques, on va typiquement obtenir des équations différentielles d'ordre deux de la forme :

$$\underbrace{M(q)\ddot{q}}_{m\vec{a}} = \underbrace{\sum f(q, \dot{q}, u, w, t)}_{\vec{f}} \quad (4.1)$$

La représentation d'état pour ce système est alors un vecteur qui comprend des variables positions et vitesses :

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \quad (4.2)$$

et on peut retrouver une équation différentielle d'ordre un sous la forme :

$$\frac{d}{dt} \begin{bmatrix} \dot{q} \\ q \end{bmatrix} = \begin{bmatrix} M(q)^{-1} \sum \dot{f}(q, \dot{q}, u, w, t) \\ \dot{q} \end{bmatrix} \quad (4.3)$$

Ce qui est la forme d'un modèle d'état en temps continu :

$$\dot{x} = f(x, u, w, t) \quad (4.4)$$

Dans le domaine de la commande, on travaille très souvent avec une approximation linéaire d'un modèle dynamique sous cette forme :

$$\dot{x} = A(t)x + B(t)u + w \quad (4.5)$$

qu'on appelle LTI qu'il n'y pas pas de dépendence directe au temps :

$$\dot{x} = Ax + Bu + w \quad (4.6)$$

Les fonction de transfert, dans le domaine de Laplace, peuvent être convertit sous cette forme et vice-versa exactement sans approximation.

4.1.1 Conversion en temps discret

Ensuite, il est possible de convertir le modèle d'état en temps continu et en temps discret en intégrant le modèle continu sur un certain pas de temps. L'approximation la plus simple pour faire cette conversion est l'intégration d'Euler, ou on aurait :

$$x_{k+1} \approx x_k + \dot{x}_k \Delta t = \underbrace{x_k + f(x_k, u_k, w_k, t_k) \Delta t}_{f_k(x_k, u_k, w_k)} \quad (4.7)$$

Pour retrouver la forme d'évolution de base utilisée dans ces notes.

Note Il y a des schémas d'intégration plus complexes, par exemple Runge-Kutta, qui sont plus précis pour une pas de temps donné Δt .

4.2 Équations de différence

La formulation de base utilisé dans ces notes, est une équation de différence, qui défini l'état à l'étape suivante basée sur l'état actuel et une action choisie :

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad k = 0, 1, \dots, N - 1 \quad (4.8)$$

C'est une formulation qu'on obtient naturellement pour des environnements artificielles comme des jeux ou il y a une notion de tour, comme les échecs. On arrive aussi à cette formulation lorsqu'on modélise des asservissement et qu'on se place du point de vu d'un micro-contrôleur qui prend des décisions dans une boucle qui s'exécute en temps fini. Par exemple, une fonction de transfert utilisant la transformée en Z prend directement cette forme lorsqu'on utilise la représentation d'état. Dans le domaine des asservissements, on travaille souvent avec une approximation linéaire de ce type d'équation que l'on note :

$$x_{k+1} = A_k x_k + B_k u_k + w_k \quad k = 0, 1, \dots, N - 1 \quad (4.9)$$

4.3 Graphes (états discrets déterministes)

Pour les systèmes où les états et actions sont discret, il est possible de représenter les états comme des noeuds sur un graphe et les actions possible comme des arcs qui nous mène vers un autre état. Cette section présente le cas déterministe. Par exemple pour un jeux d'échec, il y a un nombre fini de positions pour une pièce, et un nombre fini d'action possible, et rien n'est stochastique. On peut donc utiliser le concept d'un graphe où les noeuds sont des états discrets possibles et les arcs des actions possibles. Dans ce contexte on peut simplifier la notation :

$$x_k \Leftrightarrow i \text{ index du noeud de départ} \quad (4.10)$$

$$u_k \Leftrightarrow j \text{ index de la destination de l'arc d'action} \quad (4.11)$$

$$x_{k+1} \Leftrightarrow j \text{ index du noeud d'arrivé} \quad (4.12)$$

$$g_k(x_k, u_k) \Leftrightarrow a_{ij}^k \text{ longueur de l'arc } ij \quad (4.13)$$

L'algorithme de programmation dynamique peut alors être écrit comme :

$$J_k^*(i) = \min_{j \in U(i)} [a_{ij}^k + J_{k+1}^*(j)] \quad (4.14)$$

$$\pi_k^*(i) = \operatorname{argmin}_{j \in U(i)} [a_{ij}^k + J_{k+1}^*(j)] \quad (4.15)$$

Pour le cas discret, le coût-àvenir J peut être représenté par un vecteur où chaque élément i est le coût-à-venir de l'état i

4.3.1 Discrétisation de variables continues

4.4 Chaînes de Markov (états discrets stochastiques)

Lorsque les états sont discret mais avec des transitions probabilistes, la formulation est appelée chaîne de Markov dans la littérature, et le problème de concevoir une politique est appelé un processus de décision de Markov.

On peut encore utiliser le concept de graphe pour illustrer le système, mais dans ce cas les arcs sont associées à des probabilités de transitions. On va définir la dynamique du système par des probabilités notées p_{ij} de transiter de l'état i à l'état j qui peuvent dépendre de l'action u et de l'étape actuelle k dans le cas le plus générique.

$$p_{ij}^k(u) = P(X_{k+1} = j \mid X_k = i, u, k) \quad (4.16)$$



Capsule vidéo
Programmation dynamique pour un MDP
https://youtu.be/xHoLePda478?si=5kJxeVALBuu0n_JW

$$x_k \Leftrightarrow i \text{ index du noeud de départ} \quad (4.17)$$

$$u_k \Leftrightarrow u_k \quad (4.18)$$

$$x_{k+1} \Leftrightarrow j \text{ index du noeud d'arrivé} \quad (4.19)$$

$$f_k(x_k, u_k, w_k) \Leftrightarrow p_{ij}^k(u) \text{ probabilités de transitionner} \quad (4.20)$$

$$g_k(x_k, u_k, w_k) \Leftrightarrow a_{ij}^k \text{ longueur de l'arc } ij \quad (4.21)$$

$$J_k^*(i) = \min_{u \in U(i)} \sum_j \left[p_{ij}^k(u) [a_{ij}^k + J_{k+1}^*(j)] \right] \quad (4.22)$$

$$\pi_k^*(i) = \operatorname{argmin}_{u \in U(i)} \sum_j \left[p_{ij}^k(u) [a_{ij}^k + J_{k+1}^*(j)] \right] \quad (4.23)$$

À venir !

Chapitre 5

Équation de Bellman

5.1 Horizon de temps infini

Dans un grand nombres de problèmes, on s'intéresse à concevoir une politique qui va bien performer en continu, et non pas sur un horizon de temps fini comme c'était le cas dans les sections précédentes. Une façon de définir ce problème est de formuler le coût-à-venir à optimiser comme la limite suivante :

$$J = \lim_{N \rightarrow \infty} \mathbb{E} \left[\sum_{k=0}^N \alpha^k g_k \right] \quad (5.1)$$

ou on introduit un facteur d'escompte $0 \leq \alpha \leq 1$ qui représente un biais pour moins prendre en considération les coûts/récompenses loin dans le futur. D'un côté c'est une façon de mieux définir le problème, mais d'un autre, la présence de ce facteur est aussi un peu nécessaire pour avoir des bons fondements mathématiques et en pratique des algorithmes qui convergent. En effet, le fait d'avoir des solutions où le coût peut diverger vers l'infini est un problème.

Note Le facteur d'escompte peut aussi être interprété comme une probabilité $(1 - \alpha)$ d'atteindre un état terminal sans coûts.



Capsule vidéo

Horizon infini

<https://youtu.be/WbpSBaChigQ?si=AtAWuNlo2xDQyznT>

5.2 Équation de Bellman

À venir !



Capsule vidéo

Équation de Bellman

<https://youtu.be/18KrNlHHT3E?si=F100xS1UC0KATgCs>



Capsule vidéo

Variantes de l'équation de Bellman

https://youtu.be/98IOSI_jWyY?si=skah-1-PRdZibSPT

5.3 Équation de Hamilton–Jacobi–Bellman

$$0 = \min_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (5.2)$$

Chapitre 6

Solutions Analytiques

6.1 LQR à temps discret



Capsule vidéo

LQR temps discret

<https://youtu.be/gg0IYkQwXWs?si=tgL0d6LhGVq5KAal>

Dans le contexte d'un système dynamique linéaire à états/actions continus représenté par :

$$\underline{x}_{k+1} = f_k(\underline{x}_k, \underline{u}_k, \underline{w}_k) = A_k \underline{x}_k + B_k \underline{u}_k + \underline{w}_k \quad (6.1)$$

où \underline{x}_k et \underline{w}_k sont des vecteurs de dimension n et \underline{w}_k un vecteur de dimension m . Le vecteur \underline{w}_k représente des perturbations aléatoires, indépendantes de l'état actuel, de l'action actuelle et de tous les états passés, et avec des distributions centrées autour de zéro :

$$\mathbb{E}[\underline{w}_k] = \underline{0} \quad (6.2)$$

Si on cherche donc à minimiser l'espérance du coût-àvenir :

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} \left(\underbrace{\underline{x}_k^T Q_k \underline{x}_k + \underline{u}_k^T R_k \underline{u}_k}_{g_k(\underline{x}_k, \underline{u}_k, \underline{w}_k)} \right) + \underbrace{\underline{x}_N^T Q_N \underline{x}_N}_{g_N(\underline{x}_N)} \right] \quad (6.3)$$

où les matrices Q_k et R_k sont symétriques et définies positives :

$$Q_k \geq 0 \quad R_k > 0 \quad (6.4)$$

En appliquant l'algorithme de programmation dynamique (pour l'étape $N \rightarrow N - 1$ ou une étape générique $k + 1 \rightarrow k$), on trouve que : 1) le coût-àvenir d'un état \underline{x}_k a la forme quadratique suivante :

$$J_k^*(\underline{x}_k) = \underline{x}_k^T S_k \underline{x}_k + c \quad (6.5)$$

où S_k est une matrice symétrique qui caractérise le coût-àvenir à l'état \underline{x}_k et c est une constante qui ne dépend pas de l'état actuel. 2) la loi de commande optimale a la forme linéaire suivante :

$$\underline{u}_k^* = c_k^*(\underline{x}_k) = -K_k \underline{x}_k \quad (6.6)$$

où K_k est la matrice de gain égale à

$$K_k = [R_k + B_k^T S_{k+1} B_k]^{-1} B_k S_{k+1} A_k \quad (6.7)$$

3) la matrice S_k dans les équations précédentes peut être calculée en partant du coût final à $k = N$ et en remontant dans le temps avec la récursion suivante :

$$S_k = Q_k + A_k^T \left(S_{k+1} - S_{k+1}^T B_k^T [R_k + B_k^T S_{k+1} B_k]^{-1} B_k S_{k+1} \right) A_k \quad (6.8)$$

Chapitre 7

Algorithmes de planification

7.1 Algorithme d'itération de valeurs



Capsule vidéo

Iteration de valeurs

<https://youtu.be/SpWKHB4GupU?si=Ewb6MwRTTE0lero0>

7.2 Algorithme d'itération de politique



Capsule vidéo

Iteration de politique

<https://youtu.be/5b1o8808e44?si=5r8Wm3s0jHHCtXe5>

Chapitre 8

Algorithmes d'apprentissage

8.1 TD-Learning

8.2 Q-Learning



Capsule vidéo

Q-Learning

<https://youtu.be/eyUcnKOpwsE?si=45HiCP0bZoUF4wow>

8.3 Sarsa

8.4 Function approximation



Capsule vidéo

RL avec approximation

https://youtu.be/CGbdEDGsJnU?si=BDLgkYyqqyOR_RYZ



Capsule vidéo

Approximation de fonctions

<https://youtu.be/p8BiszV8apQ?si=Tj0lqSOQZwZmtn9u>

Annexe A

Outils mathématiques

A.1 Ensembles

On note qu'une variable x peut prendre des valeurs dans un ensemble de valeurs possible X comme :

$$x \in X \quad (\text{A.1})$$

Pour des variables discrètes on peut définir un ensemble directement avec une liste d'éléments :

$$X = \{1, 2, 3, 4, 5\} \quad (\text{A.2})$$

Pour des variables continues, on peut définir des intervalles, incluant les bornes :

$$X = [0, 10] \quad \Rightarrow \text{ex : } 0 \in X, 2 \in X, 12 \notin X \quad (\text{A.3})$$

ou excluant les bornes :

$$X = (0, 10) \quad \Rightarrow \text{ex : } 0 \notin X, 2 \in X, 10 \notin X \quad (\text{A.4})$$

ou bien avec une condition :

$$X = \{x \in \mathbb{R} | 0 \leq x \leq 10\} = [0, 10] \quad (\text{A.5})$$

qu'on peut lire, l'ensemble des nombres réels qui sont plus grand ou égal à zéro et plus petit et égal à dix.

A.2 Probabilités

A.2.1 Probabilité pour une variable discrète

On note la probabilité qu'une variable aléatoire X prenne la valeur x :

$$P(X = x) \in [0, 1] \quad (\text{A.6})$$

Pour avoir une notation plus compacte on va parfois écrire seulement :

$$p_i = P(X = x_i) \quad (\text{A.7})$$

la probabilité qu'une variable aléatoire discrète prenne le i -ème valeur possible dans l'ensmeble. Pour un modèle probabiliste bien défini, la somme des probabilités doit être égale à 1 :

$$\sum_i p_i = 1 \quad (\text{A.8})$$

Exemple 1.

Par exemple pour un dé à six faces :

$$X \in \{1, 2, 3, 4, 5, 6\} \quad (\text{A.9})$$

et si le dé est bien équilibré :

$$P(X = 1) = 1/6 \quad (\text{A.10})$$

$$P(X = 2) = 1/6 \quad (\text{A.11})$$

$$P(X = 3) = 1/6 \quad (\text{A.12})$$

$$P(X = 4) = 1/6 \quad (\text{A.13})$$

$$P(X = 5) = 1/6 \quad (\text{A.14})$$

$$P(X = 6) = 1/6 \quad (\text{A.15})$$

A.2.2 Probabilité pour une variable continue

La probabilité pour une variable continue est définie sur un interval :

$$P(a < x < b) = \int_a^b p(x)dx \quad (\text{A.16})$$

ou $p(x)$ est une fonction de densité de probabilité, tel que :

$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad (\text{A.17})$$

Exemple 2.

La fonction de densité de probabilité la plus connue, la fameuse courbe normale ou la Gaussienne, est définie par :

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(\frac{x-\mu}{2\sigma})^2} \quad (\text{A.18})$$

ou μ est le centre de la distribution et σ l'écart type.

A.2.3 Espérance

Le concept de l'espérance, est une moyenne des valeurs d'un variable aléatoire pondérée par la probabilité d'occurrence. La définition exacte est :

$$\mathbb{E}[x] = \sum p_i x_i \quad \text{or} \quad \mathbb{E}[x] = \int x p(x)dx \quad (\text{A.19})$$

Voici quelques propriété utile de l'opérateur espérance :

$$\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y] \quad (\text{A.20})$$

$$\mathbb{E}[ax] = a \mathbb{E}[x] \quad (\text{A.21})$$

$$\mathbb{E}[xy] \neq \mathbb{E}[x] \mathbb{E}[y] \quad (\text{A.22})$$

A.2.4 Probabilité jointe et conditionnelle

Une probabilité conditionnelle est une probabilité qui est une fonction d'une autre variable. On note :

$$P(B = b | A = a) \quad (\text{A.23})$$

la probabilité que la variable B prenne la valeur b , sachant que la variable A a pris la valeur a . La probabilité jointe des deux événements, i.e. que deux choses aléatoires arrivent en même temps, est notée :

$$P(B = b, A = a) \quad (\text{A.24})$$

En gros, la barre verticale veux dire "sachant que" et la virgule veux dire "et". Une probabilité jointe peut être calculée avec des probabilités conditionnelles :

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A) \quad (\text{A.25})$$

A.2.5 Loi de Bayes

On peut réarranger l'équation (A.25) pour obtenir la loi de Bayes :

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (\text{A.26})$$

qui est centrale pour les problèmes de décision avec observations partielles. Typiquement on cherche à connaître la probabilité d'une cause B sachant qu'un symptôme A a été observé.

A.3 Opérations

A.3.1 Minimum/maximum

Annexe B

Approximation de fonctions

À venir !

Annexe C

Exercices

C.1 Formulation du problème

C.1.1 Fonction coût vs récompense

Vous voulez résoudre un problème que vous avez initialement formulé comme minimiser la fonction suivante :

$$g(x, u) = x^2 + u^2 \quad (\text{C.1})$$

mais vous voulez utiliser un algorithme qui maximise une fonction récompense, quelle fonction récompense vous devez passer à l'algorithme pour qu'il résoudre votre problème ?

C.1.2 Politique fonction du temps

Identifier une situation ou la politique optimale (selon votre bon sens, pas besoin de faire de calculs..) devrait dépendre directement du temps ou de l'étape actuelle et décrivez la.

C.1.3 Politique stochastique

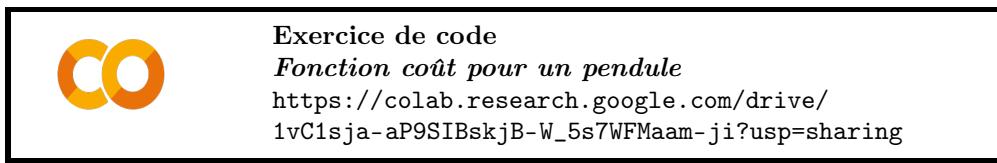
Est-ce qu'il pourrait y avoir un avantage à une politique stochastique ? Si oui dans quelle situation ? Réfléchissez à la question, décrivez votre réflexion et donner un exemple ou contre-exemple.

C.1.4 Fonction de coût pour un pendule

Compétences à développer :

- Compréhension des paramètres d'une fonction de coût quadratique
- Compréhension de la forme générique de coût additif $J = \int_0^{t_f} g(x, u, t) dt + h(x_f, t_f)$

Pour ce numéro du devoir, vous devrez utiliser le code disponible au lien ici :



Note : Vous pouvez travailler en-ligne directement dans colab ou travaillez directement sur votre ordinateur en téléchargeant la librairie <https://github.com/SherbyRobotics/pyro>.

Pour chacune des situations suivantes :

- a) Situation de référence : exécuter le code avec la fonction coût quadratique par défaut.
- b) Ajuster les valeurs dans la matrice Q pour pénaliser plus l'erreur en position du pendule.
- c) Modifiez la fonction $g(x, u, t)$ et $h(x, t)$ pour obtenir une solution qui correspond au temps minimal.
- d) [Optionnel] Testez et explorez d'autres variantes de fonction coût.

analysez :

- 1) la figure de coût-àvenir J^* calculée pour tout les états (qui correspond au coût minimal qui va être encouru à partir de cet état si les actions optimales sont prises).
- 2) la loi de commande générée (couple en fonction de l'angle et la vitesse).
- 3) la trajectoire pour le système lorsque le pendule débute à partir de la position en bas.

et interprétez les résultats (i.e. notez les changements et tentez d'expliquer ce qui peut les expliquer.)

Note : Pour plusieurs raisons l'algorithme peut avoir de la difficulté à converger pour certaines fonctions de coût. Essayez des changements plus mineurs si c'est le cas. Le but ici n'est pas de vous faire travailler pour ajuster les paramètres de convergence (un sujet pas encore abordé).

C.2 Programmation dynamique exacte

C.2.1 Navigation optimale dans un graphe

Compétences à développer :

- Programmation dynamique pour un problème avec des états et actions discrètes.
- Algorithmes pour déterminer un chemin le plus court dans un graphe

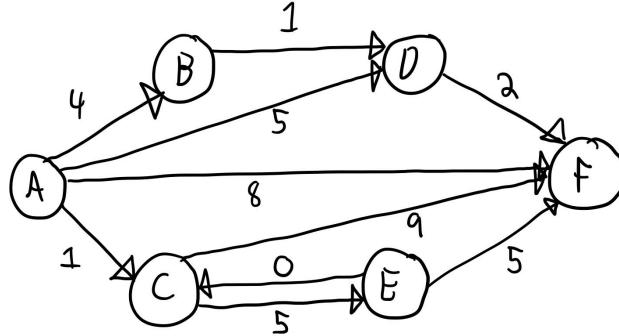


FIGURE C.1 – Graphique qui représente des chemins possibles pour aller vers la position F

Appliquez l'algorithme de programmation dynamique exacte :

$$J_k^*(\underline{x}_k) = \min_{\underline{u}_k} \left[g_k(\underline{x}_k, \underline{u}_k) + J_{k+1}^* \underbrace{\left(f_k(\underline{x}_k, \underline{u}_k) \right)}_{\underline{x}_{k+1}} \right] \quad (\text{C.2})$$

$$u_k^*(\underline{x}_k) = \operatorname{argmin}_{\underline{u}_k} \left[g_k(\underline{x}_k, \underline{u}_k) + J_{k+1}^* \underbrace{\left(f_k(\underline{x}_k, \underline{u}_k) \right)}_{\underline{x}_{k+1}} \right] \quad (\text{C.3})$$

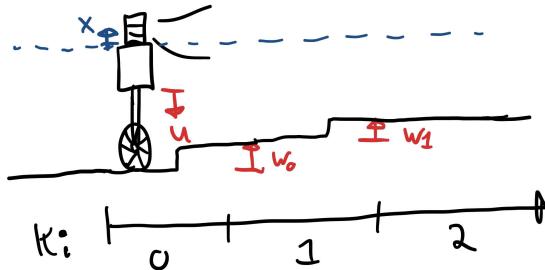
pour résoudre les actions optimales (choix de l'arc à suivre) pour se rendre à l'état cible (Noeud F) à partir de l'état actuel (les Noeuds $x_k \in [A, B, C, D, E]$) et de l'index de temps actuel k . Le coût de chaque option de chemin est représenté par le chiffre indiqué pour chaque arc sur le graphique ci-dessus. Considérez une fonction de coût sur un horizon de 5 pas de temps ($N=5$) et calculez les actions optimales pour les index de temps $k = [0, 1, 2, 3, 4]$. Considérez que le coût final est infini si on ne termine pas sur le Noeud F à $k = 5$. Le tableau suivant peut vous aider pour synthétiser les résultats :

$k =$	0	1	2	3	4	$N = 5$
$J^*(A) =$						
$u^*(A) =$						
$J^*(B) =$						
$u^*(B) =$						
$J^*(C) =$						
$u^*(C) =$						
$J^*(D) =$						
$u^*(D) =$						
$J^*(E) =$						
$u^*(E) =$						
$J^*(F) =$						

C.2.2 Loi de commande pour une suspension active

Compétences à développer :

- Application de la programmation dynamique pour un problème avec une dynamique linéaire et un coût quadratique
- Programmation dynamique exacte déterministe



Un robot équipé d'une suspension active roule sur un terrain accidenté. À chaque pas de temps sa suspension peut être ajustée d'une hauteur u_k . On désire calculer une loi de commande optimale pour deux pas de temps ($N=2$). L'évolution de la hauteur totale du robot par rapport à une hauteur désiré (x_k) est donnée par :

$$\text{Dynamique : } x_1 = x_0 + u_0 + w_0 \quad (\text{C.4})$$

$$x_2 = x_1 + u_1 + w_1 \quad (\text{C.5})$$

où les variables w_k sont des variations de hauteur du terrain connues d'avance car le robot a effectué un balayage LIDAR du chemin. On cherche ici à minimiser la hauteur finale du robot x_k à $N = 2$, mais aussi la variation de hauteur à chaque instant car ces changements brusques perturbent les instruments. La fonction coût à optimiser est définie comme il suit :

$$\text{Coûts : } g_0(x_0, u_0, w_0) = (u_0 + w_0)^2 \quad (\text{C.6})$$

$$g_1(x_1, u_1, w_1) = (u_1 + w_1)^2 \quad (\text{C.7})$$

$$g_2(x_2) = x_2^2 \quad (\text{C.8})$$

et on cherche à minimiser la somme de ces coûts additifs :

$$J = \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \quad (\text{C.9})$$

Utilisez l'algorithme de programmation dynamique exacte pour calculer les lois de commande optimales comme des fonctions de l'état actuel x_k mais aussi des mesures w_k qui sont des valeurs connues. Le tableau suivant peut vous aider pour synthétiser les résultats :

$k =$	0	1	$N = 2$
$J^*(x_k, w_k) =$			
$u^*(x_k, w_k) =$			

C.3 Commande stochastique

C.3.1 Loi de commande pour une suspension active II

Compétences à développer :

- Programmation dynamique exacte stochastique
- Calcul de l'espérance d'une variable aléatoire

Ici on reprend le même problème précédent, mais en considérant maintenant que les irrégularités du terrain w_k sont des perturbations inconnues. Supposons que nous avons des informations probabilistes sur les valeurs probables que ces perturbations peuvent prendre. Plus précisément que les perturbations ont 50% de chance de prendre la valeur 1 et 50% de chance d'être égale à zéro :

$$P(w_0 = 1) = 0.5 \quad (\text{C.10})$$

$$P(w_0 = 0) = 0.5 \quad (\text{C.11})$$

$$P(w_1 = 1) = 0.5 \quad (\text{C.12})$$

$$P(w_1 = 0) = 0.5 \quad (\text{C.13})$$

Recalculer les fonctions de commande optimales qui vont ici seulement dépendre de l'état actuel x_k , basé sur la programmation dynamique stochastique. On cherche donc ici à minimiser l'espérance du coûts-àvenir :

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad (\text{C.14})$$

avec l'algorithme de programmation dynamique suivant :

$$J_k^*(x_k) = \min_{\underline{u}_k} \mathbb{E}_{\underline{w}_k} \left[g_k(x_k, \underline{u}_k) + J_{k+1}^* \underbrace{(f_k(x_k, \underline{u}_k))}_{\underline{x}_{k+1}} \right] \quad (\text{C.15})$$

$$u_k^*(x_k) = \operatorname{argmin}_{\underline{u}_k} \mathbb{E}_{\underline{w}_k} \left[g_k(x_k, \underline{u}_k) + J_{k+1}^* \underbrace{(f_k(x_k, \underline{u}_k))}_{\underline{x}_{k+1}} \right] \quad (\text{C.16})$$

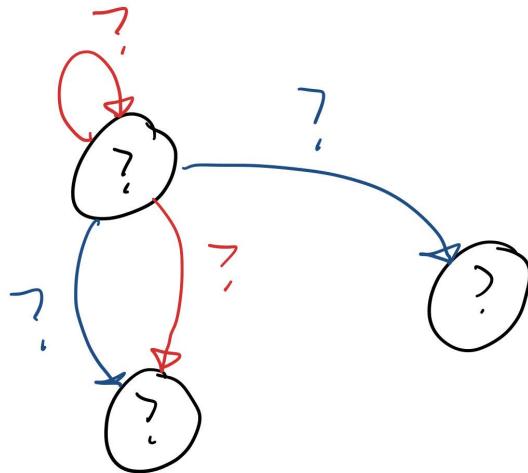
Le tableau suivant peut vous aider pour synthétiser les résultats :

$k =$	0	1	$N = 2$
$J^*(x_k) =$			
$u^*(x_k) =$			

C.3.2 Commande stochastique pour une diva à l'opéra

Compétences à développer :

- Modélisation d'un problème de type chaînes de Markov
- Programmation dynamique stochastique



Une diva est en résidence à votre casino pour effectuer des spectacles à tous les soirs. N représentations sont encore prévus à l'horaire. Lorsque la diva est satisfaite de sa performance (ce qui se produit aléatoirement avec une probabilité p_s) elle accepte de chanter le soir suivant. Toutefois lorsqu'elle n'est pas satisfaite, la seule façon de la convaincre de chanter le soir suivant est de lui acheter un cadeau qui coûte x dollar, une stratégie qui fonctionne avec une probabilité de p_c . Lorsque la stratégie du cadeau ne fonctionne pas (et aussi lorsqu'on ne lui offre pas de cadeau) la diva insatisfaite refuse de chanter le soir suivant et reste insatisfaite. Lorsque la diva refuse de chanter le casino perd y dollars par spectacle annulé. On considère qu'on peut offrir un cadeau à la diva pour tenter de la convaincre de chanter à nouveau une fois par jour.

1) Définissez le problèmes sous la structure d'une chaîne de markov :

1. Déterminez les états possibles
2. Déterminez les actions possibles
3. Illustriez le processus graphiquement avec des noeuds (états) et les transitions possibles
4. Déterminez les probabilités de transitions
5. Déterminez les coûts associés aux transitions.

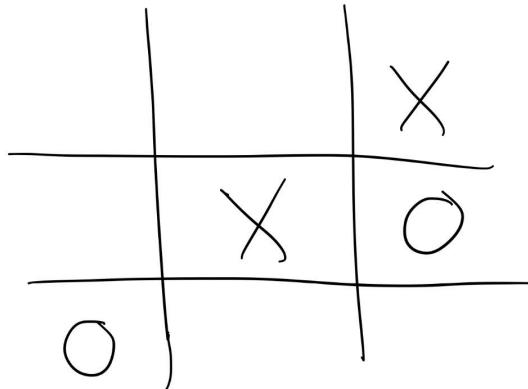
2) Calculer le coût-àvenir et la décision optimale en fonction de l'humeur de la diva pour chaque jour si $p_s = 0.5$, $p_c = 0.5$, $x = 10000$, $y = 15000$ et $N = 4$.

C.4 Commande robuste

C.4.1 Commande minimax pour tic-tac-toe

Compétences à développer :

- Algorithme minimax



Considérez l'état d'une partie de tic-tac-toe ci-dessus où c'est le tour des Xs à jouer. Si on considère une fonction de récompense qui consiste en seulement une valeur terminal de +1 lors d'une victoire des Xs, -1 lorsque d'une victoire des Os et 0 pour une nulle, démontrez en utilisant le principe de la programmation dynamique minimax que la valeur optimale de la récompense à venir pour cet état de jeux est de +1, i.e. la victoire est garantie pour les Xs. Décrivez une stratégie optimale qui garantie la victoire.

Note : Il n'est pas nécessaire d'évaluer toutes les branches de possibilités futures, dès qu'une action mène à un coût $J = +1$ par exemple on peut déterminer que le maximum est nécessairement +1 sans calculer les autres actions.

$$J_k^*(x_k) = \max_X \min_O \left[g_k + J_{k+1} = \begin{cases} +1 & \text{pour une position gagnante pour X} \\ 0 & \text{pour une grille pleine sans vainqueur} \\ -1 & \text{pour une position gagnante pour O} \\ J_{k+1}^*(x_{k+1}) & \text{pour une position non terminale} \end{cases} \right] \quad (\text{C.17})$$

C.5 Solutions analytiques

C.5.1 Solution LQR par programmation dynamique

Compétences à développer :

- Manipulation d'équations matricielles
- Manipulation d'équations impliquant le calcul de l'espérance
- Compréhension de la solution LQR à temps discret

Dans le contexte d'un système dynamique linéaire à états/actions continus représenté par :

$$\underline{x}_{k+1} = f_k(\underline{x}_k, \underline{u}_k, \underline{w}_k) = A_k \underline{x}_k + B_k \underline{u}_k + \underline{w}_k \quad (\text{C.18})$$

où \underline{x}_k et \underline{w}_k sont des vecteurs de dimension n et \underline{w}_k un vecteur de dimension m . Le vecteur \underline{w}_k représente des perturbations aléatoires, indépendantes de l'état actuel, de l'action actuelle et de tous les états passés, et avec des distributions centrées autour de zéro :

$$\mathbb{E}[\underline{w}_k] = 0 \quad (\text{C.19})$$

On cherche donc à minimiser l'espérance du coût-à-venir :

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} \left(\underbrace{\underline{x}_k^T Q_k \underline{x}_k + \underline{u}_k^T R_k \underline{u}_k}_{g_k(\underline{x}_k, \underline{u}_k, \underline{w}_k)} \right) + \underbrace{\underline{x}_N^T Q_N \underline{x}_N}_{g_N(\underline{x}_N)} \right] \quad (\text{C.20})$$

où les matrices Q_k et R_k sont symétriques et définies positives :

$$Q_k \geq 0 \quad R_k > 0 \quad (\text{C.21})$$

En appliquant l'algorithme de programmation dynamique (pour l'étape $N \rightarrow N - 1$ ou une étape générique $k + 1 \rightarrow k$). Démontrez que : 1) le coût-à-venir d'un état \underline{x}_k a la forme quadratique suivante :

$$J_k^*(\underline{x}_k) = \underline{x}_k^T S_k \underline{x}_k + c \quad (\text{C.22})$$

où S_k est une matrice symétrique qui caractérise le coût-à-venir à l'état \underline{x}_k et c est une constante qui ne dépend pas de l'état actuel. 2) la loi de commande optimale a la forme linéaire suivante :

$$\underline{u}_k^* = c_k^*(\underline{x}_k) = -K_k \underline{x}_k \quad (\text{C.23})$$

où K_k est la matrice de gain égale à

$$K_k = [R_k + B_k^T S_{k+1} B_k]^{-1} B_k S_{k+1} A_k \quad (\text{C.24})$$

3) la matrice S_k dans les équations précédentes peut être calculée en partant du coût final à $k = N$ et en remontant dans le temps avec la récursion suivante :

$$S_k = Q_k + A_k^T \left(S_{k+1} - S_{k+1}^T B_k^T [R_k + B_k^T S_{k+1} B_k]^{-1} B_k S_{k+1} \right) A_k \quad (\text{C.25})$$

Notes sur la dérivation d'un scalaire par un vecteur

Lorqu'on a une fonction scalaire $y = f(\underline{x})$ avec plusieurs entrée regroupée dans un vecteur colonne $\underline{x} \in \mathbb{R}^n$, par convention si on dérive cette fonction par rapport à un vecteur colonne \underline{x} de dimension $n \times 1$, le résultat est un vecteur rangé $1 \times n$;

$$\underline{z} = \frac{\partial y}{\partial \underline{x}} = \left[\begin{array}{ccc} \frac{\partial y}{\partial x_1} & \dots & \frac{\partial y}{\partial x_n} \end{array} \right] \Leftrightarrow z_i = \frac{\partial y}{\partial x_i} \quad (\text{C.26})$$

TABLE C.1 – Identités pour la dérivation d'une fonction scalaire par un vecteur

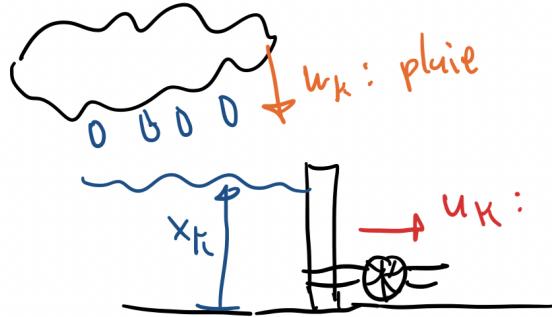
Fonction scalaire $y = f(\underline{x})$	Expression du gradient $\frac{\partial y}{\partial \underline{x}}$	Notes
$\underline{a}^T \underline{x} = \underline{x}^T \underline{a}$	\underline{a}^T	Si \underline{a} n'est pas une fonction de \underline{x}
$\underline{x}^T \underline{x}$	$2 \underline{x}^T$	
$\underline{x}^T A \underline{x}$	$\underline{x}^T (A + A^T)$	Si A n'est pas une fonction de \underline{x}
$\underline{x}^T A \underline{x}$	$2 \underline{x}^T A$	Si A est symétrique Si A n'est pas une fonction de \underline{x}

C.6 Algorithmes de planification

C.6.1 Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur

Compétences à développer :

- Chaînes de Markov
- Programmation dynamique stochastique
- Optimisation sur un horizon de temps infini
- Algorithme d'itération de valeur



Vous contrôlez les vannes de la turbine d'un barrage qui alimente une usine de production d'aluminium. Lorsque les turbines ne tournent pas à pleine capacité le déficit de production entraîne des pertes de revenus. Toutefois, la pluie se fait rare et il n'est pas possible de toujours faire fonctionner les turbines à pleine capacité, on désire donc optimiser la décision d'ouvrir totalement ou partiellement les vannes en fonction du niveau actuel d'un barrage. Supposons qu'on prend la décision une fois par jour sous la forme d'un volume u_k journalier qui passe dans la turbine et que l'évolution du niveau d'eau x_k dans le barrage est donné par :

$$x_{k+1} = \max [x_k - u_k + w_k, 4] \quad (\text{C.27})$$

où

$$x_k \in [0, 1, 2, 3, 4] \quad (\text{C.28})$$

$$u_k \in [0, 1, 2] \quad (\text{C.29})$$

$$u_k \leq x_k \quad (\text{C.30})$$

$$w_k = \begin{cases} 0 & P = 0.4 \\ 1 & P = 0.4 \\ 3 & P = 0.2 \end{cases} \quad (\text{C.31})$$

Le barrage a un volume maximal de 4 unités, l'eau excédante est perdue. On peut sélectionner un volume de turbine de 0, 1 ou 2 à conditions d'avoir la quantité d'eau suffisante. L'apport en eau w_k dépendant des précipitations aléatoires avec les probabilités données.

On cherche à minimiser les pertes financières donnés par :

$$g_k = \begin{cases} 0 & \text{si } u_k = 2 \\ 25 & \text{si } u_k = 1 \\ 100 & \text{si } u_k = 0 \end{cases} \quad \begin{array}{l} \text{Pleine production} \\ \text{Production essentielle seulement} \\ \text{Arrêt de la production} \end{array} \quad (\text{C.32})$$

sur un horizon de temps infini avec un facteur d'escompte (valeur actuelle des pertes futures) $\alpha = 0.9$. Donc de minimiser le coût-à-venir suivant :

$$J = \lim_{N \rightarrow \infty} \mathbb{E} \left[\sum_{k=0}^N \alpha^k g_k \right] \quad (\text{C.33})$$

Déterminer la loi de commande optimale en fonction de cet objectif :

$$u_k^* = c^*(x_k) = \begin{cases} ? & \text{si } x_k = 1 \\ ? & \text{si } x_k = 2 \\ ? & \text{si } x_k = 3 \\ ? & \text{si } x_k = 4 \end{cases} \quad (\text{C.34})$$

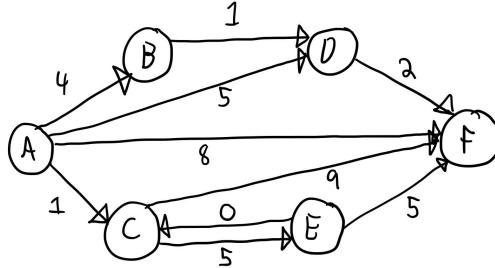
en utilisant l'algorithme d'itération de valeur (Value-iteration).

C.7 Algorithmes d'apprentissage

C.7.1 *Q-learning* pour une navigation optimale

Compétences à développer :

- Comprendre les bases de la méthode d'apprentissage par renforcement *Q-Learning*



Supposons qu'on ne connaît pas le coût des transitions du graphique ci-dessus mais que plusieurs trajectoires ont été effectuée expérimentalement et que le coût de chaque transition a été mesuré :

Épisode	Trajet	Coûts mesurés pour chaque transition
1	A,C,F	1,9
2	A,B,D,F	4,1,2
3	A,D,F	5,2
4	A,C,F	1,9
5	A,C,E,F	1,5,5
6	A,B,D,F	4,1,2
7	A,F	8
8	A,C,E,C,E,F	1,5,0,5,5
9	A,B,D,F	4,1,2
10	A,C,E,C,E,F	1,5,0,5,5

Utilisez ces données expérimentales pour faire des mises à jour des valeurs $Q(x, u)$ pour chaque transition effectuées. Comme le système est déterministe, vous pouvez prendre la version déterministe de l'algorithme de *Q-learning* (i.e. un taux d'apprentissage égal à un) :

$$Q(x, u) \leftarrow g_{\text{mesure}} + \min_{u_{k+1}} [Q(x_{k+1}, u_{k+1})] \quad (\text{C.35})$$

où x est le point de départ, $u = x_{k+1}$ est la destination et g_{mesure} le coût de la transition mesurée. Le tableau suivant peut vous aider pour synthétiser les résultats des itérations :

État(x)	Action (u)	Valeurs Q
A	B	
A	D	
A	F	
A	C	
B	D	
C	F	
C	E	
D	F	
E	C	
E	F	

À partir des valeurs $Q(x, u)$ calculées, calculez le coût-àvenir $J(x)$ et la loi de commande optimale $c(x)$. Comparez avec ce que vous aviez obtenus au devoir 1 lorsque le même problème avait été résolu par programmation dynamique.

C.7.2 Apprentissage d'une fonction $Q(x, u)$ approximée

Compétences à développer :

- Descente du gradient stochastique
- Apprentissage par renforcement avec des approximations de fonctions
- Solution LQR pour un horizon de temps infini

Pour le système avec la dynamique et la fonction de coût instantané suivante :

$$x_{k+1} = 0.5x_k + u_k \quad (\text{C.36})$$

$$g_k = x_k^2 + u_k^2 \quad (\text{C.37})$$

Génération d'une base de données

Générez une base de données avec environ 1000 données (x_k, u_k, g_k, x_{k+1}) avec une ou plusieurs séquences, avec des conditions initiales aléatoires, et avec des actions aléatoires u_k .

Apprentissage d'une fonction Q approximée

L'objectif est "d'apprendre" la fonction de coût-àvenir optimale (donc indirectement la loi de commande optimale) avec seulement les données en utilisant l'approximation polynomiale d'ordre 2 suivante :

$$\hat{Q}(x, u) \approx w_1 x^2 + w_2 u^2 + w_3 x u \quad (\text{C.38})$$

Utilisez les données générées à l'étape précédente et l'équation de mise à jour des paramètres suivante :

$$w_i^{new} = w_i^{old} + \eta \left[g_k + \min_{u_{k+1}} \hat{Q}(x_{k+1}, u_{k+1}) - \hat{Q}(x_k, u_k) \right] \frac{\partial \hat{Q}}{\partial w_i} \quad (\text{C.39})$$

pour estimer les paramètres (i.e. apprendre) w_1 , w_2 et w_3 .

Notes : Si les valeurs initiales pour x et les entrées u sont entre -1 et 1, avec un taux d'apprentissage autour de 1 vous devriez converger avec moins de 1000 itérations environ. L'étape de la minimisation peut se faire analytiquement ici. Vous pouvez travailler avec un chiffrier plutôt que du code si vous voulez pour 2.1 et 2.2

Comparaison avec la solution analytique LQR

Le système ci-dessus a une dynamique linéaire et un coût quadratique. Calculez le coût-àvenir optimal pour un horizon de temps infini de façon analytique et comparez à la fonction \hat{Q} obtenue numériquement par itérations.

Notes : La fonction $Q(x, u)$ analytique exacte pour la solution LQR correspond à la somme des termes à l'intérieur de la fonction min, il suffit de substituer la matrice S par la solution de l'équation de Riccati algébrique. Ici toutes les matrices sont 1x1 donc des scalaires.

Bibliographie

- [Bertsekas, 2017] Bertsekas, D. P. (2017). *Dynamic Programming and Optimal Control*. Athena Scientific, Nashua, NH, 4th edition edition.
- [Silver, 2015] Silver, D. (2015). RL Course.
- [Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning, second edition : An Introduction*. Bradford Books, Cambridge, Massachusetts, 2nd edition edition.
- [Tedrake, 2023] Tedrake, R. (2023). *Underactuated Robotics : Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. Course Notes for MIT 6.832.