
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 27 septembre 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	5
1.1	Introduction	5
1.2	La prise de décision en séquence	5
1.2.1	Un comportement défini par une politique à concevoir	5
1.2.2	Un objectif formulé avec une fonction scalaire cumulative	6
1.3	Tour d'horizon	8
1.3.1	Le problème canonique de commande optimale	8
1.3.2	Le problème canonique d'apprentissage par renforcement	9
1.3.3	Apprentissage par renforcement vs. apprentissage machine	10
1.3.4	Conséquences long terme	10
1.4	Exemples de mise en oeuvre	10
2	Programmation dynamique	11
2.1	Formulation du problème	11
2.1.1	Dynamique	11
2.1.2	Politique (Loi de commande)	11
2.1.3	Fonction objectif (coût ou récompense)	12
2.1.4	Contraintes	12
2.1.5	Coût-àvenir	13
2.1.6	Solution optimale	13
2.1.7	Terminologie	14
2.2	Principe d'optimalité	16
2.3	Programmation dynamique excate	16
2.4	Variations sur un thème de Bellman	19
2.5	Forces et limites de la programmation dynamique	20
2.6	Programmation dynamique approximée	20
3	Commande stochastique	21
3.1	Dynamique stochastique	21
3.2	Optimisation de l'espérance	23
3.3	Formulation minimax (commande robuste)	23
3.4	Observations Partielles	24
3.4.1	Espace croyance	25
3.5	Politique Stochastique	25
4	Modèles d'évolution	26
4.1	Équations différentielles (temps continus)	26
4.1.1	Conversion en temps discret	27
4.2	Équations de différence	27
4.3	Graphes (états discrets déterministes)	27
4.3.1	Discrétisation de variables continues	28
4.4	Chaînes de Markov (états discrets stochastiques)	28
4.4.1	Coût déterministe	28

4.4.2	Coût stochastique	29
5	Équations de Bellman	30
5.1	Horizon de temps infini	30
5.2	Équations de Bellman	31
5.2.1	Variantes pour un processus de décision de Markov	31
5.3	Coût-àvenir d'une politique	32
5.3.1	Variantes pour une processus de décision de Markov	32
5.3.2	Solution matricielle	32
5.4	Équation de Hamilton–Jacobi–Bellman	33
6	Solutions Analytiques	34
6.1	LQR à temps discret	34
6.1.1	Horizon infini	35
6.2	LQR à temps continu	35
6.2.1	Horizon infini	37
6.2.2	Stabilisation de trajectoires	37
7	Algorithmes de planification	39
7.1	Algorithme d'itération de valeurs	39
7.1.1	Conditions de convergence	39
7.1.2	Évaluation de politique	40
7.2	Algorithme d'itération de politique	40
8	Algorithmes d'apprentissage	41
8.1	Évaluation	41
8.1.1	Monte-carlo	41
8.1.2	TD-Learning	41
8.1.3	Sarsa	41
8.2	Optimisation	41
8.3	Q-Learning	41
8.4	Exploration	41
8.4.1	Exploitation vs. exploration	41
8.5	Approximation de fonctions	41
A	Outils mathématiques	42
A.1	Ensembles	42
A.2	Probabilités	42
A.2.1	Probabilité pour une variable discrète	43
A.2.2	Probabilité pour une variable continue	43
A.2.3	Espérance	44
A.2.4	Probabilité jointe et conditionnelle	44
A.2.5	Loi de Bayes	44
B	Approximation de fonctions	45
C	Exercices	46
C.1	Introduction et formulation du problème	46
C.1.1	Fonction coût vs récompense	46
C.1.2	Limites de la formulation coût/récompense cumulative	46
C.1.3	Formes possibles des politiques optimales	46
C.1.4	Learn to fly	46
C.1.5	Fonction de coût pour un pendule	47
C.2	Programmation dynamique exacte	48
C.2.1	Navigation optimale dans un graphe	48

C.2.2	Loi de commande pour une suspension active	49
C.2.3	Politique optimale pour un thermostat	50
C.2.4	Chemin le plus court dans un graphe	51
C.3	Commande stochastique	52
C.3.1	Loi de commande pour une suspension active II	52
C.3.2	Commande stochastique pour une diva à l'opéra	53
C.3.3	Stratégie optimale aux échecs	54
C.4	Commande robuste	55
C.4.1	Commande minimax pour tic-tac-toe	55
C.5	Solutions analytiques	56
C.5.1	Solution LQR par programmation dynamique	56
C.5.2	LQR en temps continu	58
C.5.3	Implémentation LQR	59
C.6	Algorithmes de planification	60
C.6.1	Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur	60
C.6.2	Algorithme d'itération de valeurs	61
C.6.3	Évaluation d'une politique	61
C.7	Algorithmes d'apprentissage	62
C.7.1	<i>Q-learning</i> pour une navigation optimale	62
C.7.2	Apprentissage d'une fonction $Q(x, u)$ approximée	63
C.7.3	<i>Q-Learning</i> à partir de l'algorithme DP	64
C.7.4	<i>Q-Learning</i> avec échantillonnage	64
C.8	Apprentissage par renforcement appliqué	65
C.8.1	Hands-on PPO	65

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 des 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 probabilistes 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/1ThWOUmkVY?si=wWIa0-0YpvR-vYbL>

1.2 La prise de décision en séquence

L'élément central qui unit les problèmes de prise de décisions 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*

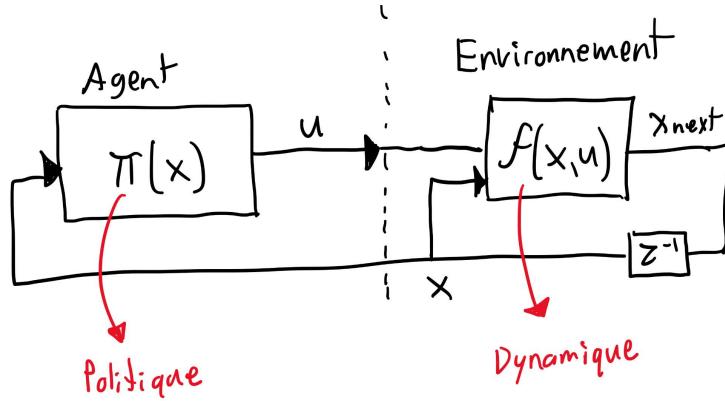


FIGURE 1.1 – Dynamique en boucle fermée avec un agent

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. 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.

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 méthode utilisée pour d'écrire l'évolution du système. C'est une condition qui garanti l'optimalité de la prise de décision en fonction de la fonction coût qui défini l'objectif. D'un certain point de vue toutes les méthodes de commandes et d'apprentissage par renforcement on 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 jeux 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. Une bonne partie de ces notes a donc comme objectif de présenter cette équation, dans les divers contextes, et des algorithmes pour la résoudre (souvent approximativement).

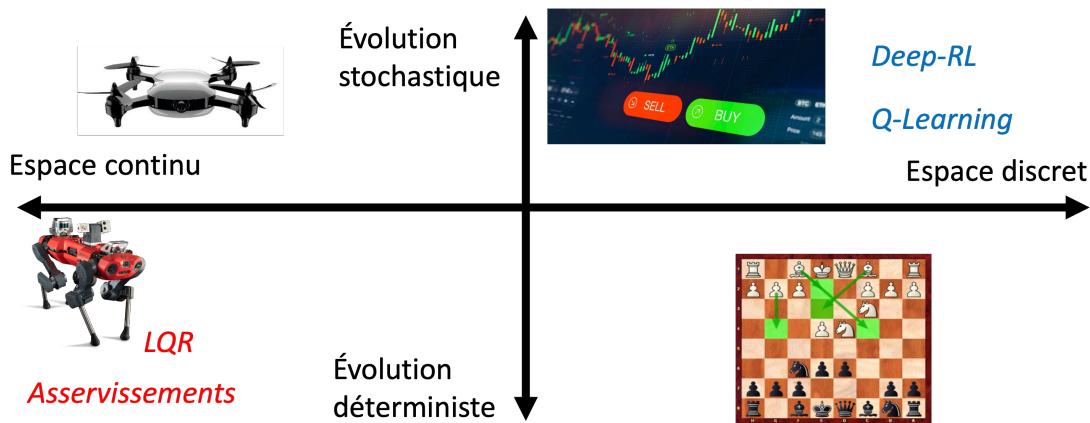


FIGURE 1.3 – Différentes contextes de prise de décision en séquence

1.3.1 Le problème canonique 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 de 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*.

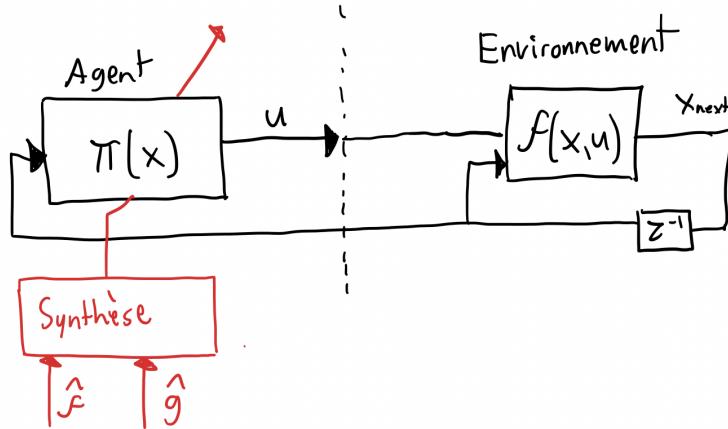


FIGURE 1.4 – Apperçu du problème de commande optimale

1.3.2 Le problème canonique 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, ce type de problème (lorsque utilise des données pour ajuster la politique ou identifier des paramètres d'un modèle) peut être nommé une loi de commande adaptative ou un problème d'identification de système.

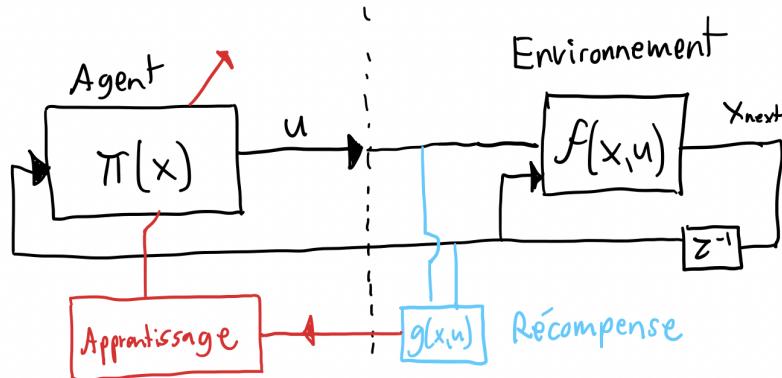


FIGURE 1.5 – Apperçu du problème d'apprentissage par renforcement



Capsule vidéo
Commande optimale vs. apprentissage par renforcement
<https://youtu.be/Zm4hGWqC5dI>

Exploration vs. exploitation

Un sous problème qui apparaît dans un contexte d'apprentissage par renforcement, 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.

1.3.3 Apprentissage par renforcement vs. apprentissage machine

L'apprentissage par renforcement, est une sous-catégorie d'apprentissage machine distincte de l'apprentissage supervisé ou l'apprentissage non-supervisé. Apprendre une politique par apprentissage supervisé nécessiterait d'avoir des démonstrations d'un expert qui donnerait les bonnes réponses d'action à faire selon les états. C'est en fait un domaine actif en robotique nommé *Behavior cloning*. Le problème d'apprentissage par renforcement est différent, on laisse l'agent apprendre par essai et erreur, et la forme de la supervision est fondamentalement différente : plutôt que lui dire les bonnes actions, on lui donne un signal de performance scalaire sous la forme de la fonction coût/récompense.

1.3.4 Conséquences long terme

Un aspect particulier du problème, qui distingue le domaine versus d'autres formes d'apprentissage machine ou d'optimisation, est que l'action du système peut avoir des conséquences long-termes qui ne sont pas immédiatement connues avant une longue évolution du système dynamique. On ne peut donc pas formuler le problème comme une optimisation statique, et le cœur du sujet tourne autour de méthode pour faire remonter l'information sur les conséquences futures d'une action.

1.4 Exemples de mise en oeuvre

Survolez les exemples suivants, analyser pour comprendre et tentez de faire des modifications pour voir si vous avez bien compris.



Exercice de code

Optimal control intro with a pendulum swing-up

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



Exercice de code

Drone learning to fly with PPO

https://colab.research.google.com/drive/1-EuRPiyf2Gv0n8p_17j1vvEIE188WT1m?usp=sharing



Exercice de code

DP and PPO for a pendulum swing-up

https://colab.research.google.com/drive/1umk613li2ts_Ny9icRL-SjFTHq-a5mAJ?usp=sharing

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 veut optimiser une politique pour un nombre fini de N d'étapes futures. 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 sur l'apprentissage par renforcement 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 différences 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 bruitée de l'état (Chapitre 3.4).

2.1 Formulation du problème

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

2.1.1 Dynamique

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

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

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. Certains algorithmes utilisent cette fonction, d'autres l'assument inconnue.

2.1.2 Politique (Loi de commande)

La politique est une fonction notée 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 .

$$u_k = \pi_k(x_k) \tag{2.2}$$

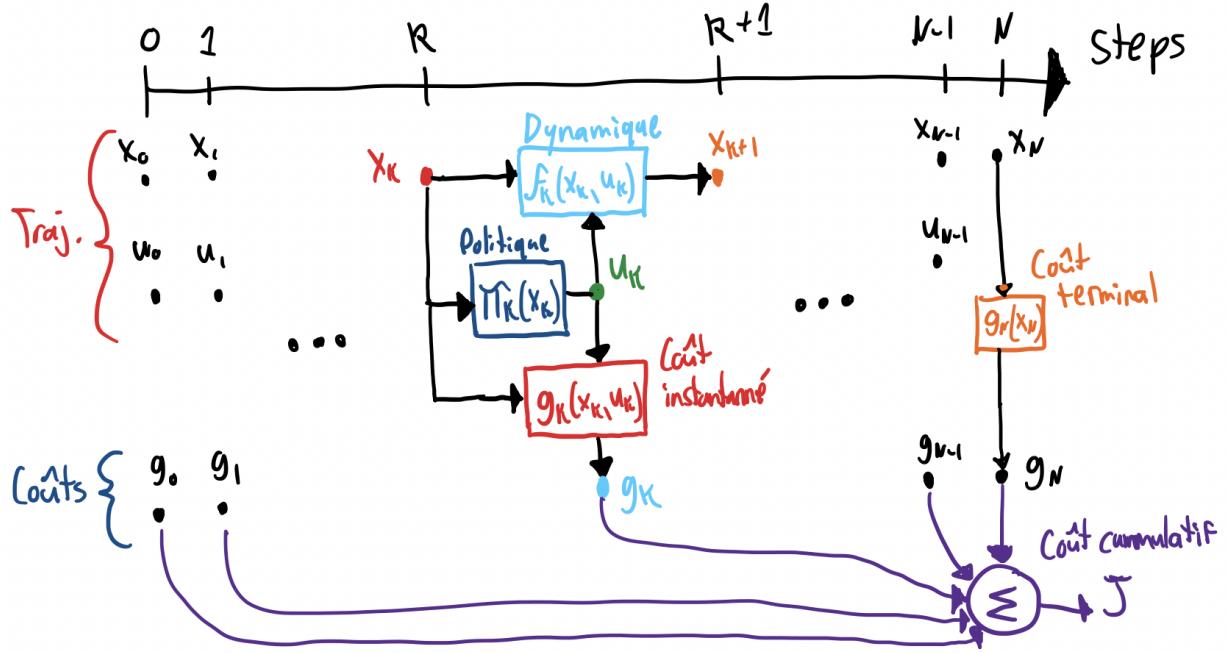


FIGURE 2.1 – Évolution du système en boucle fermée étape par étape.

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 :

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

2.1.3 Fonction objectif (coût ou récompense)

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

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

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 . Un art en apprentissage par renforcement est de bien représenté l'objectif avec les fonctions coûts g_k et g_N appropriées, plusieurs formulations sont théoriquement équivalentes, mais certaines variantes peuvent faciliter la tâche des algorithmes.

2.1.4 Contraintes

Il est souvent nécessaire pour bien caractériser l'objectif, d'inclure des contraintes dures dans le formulation d'un problème, on va donc considérer qu'à une étape k , il y a un certain ensemble \mathcal{X}_k d'états permis et un ensemble \mathcal{U}_k d'action possibles, qui peut dépendre de l'état actuel :

$$x_k \in \mathcal{X}_k \quad u_k \in \mathcal{U}_k(x_k) \quad (2.5)$$

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-à-venir

Un concept important est celui du coût-à-venir, 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 :

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

Notez ici que le coût-à-venir 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-à-venir. 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-à-venir. On va noter J^* la fonction coût-à-venir 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 discréteisé).
$\mathcal{U}_k(x)$	Contraintes, <i>control set</i>	Ensemble représentant les actions qui sont possible lorsque l'état du système est x à l'étape k
$g_k(x_k, u_k)$	Coût instantanée, récompense (problème de maximisation)	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.
$J(x_0, \dots, x_N)$	Fonction de coût cumulative, (opposé de la) fonction de récompense	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> , <i>value fonction</i>	Fonction scalaire qui représente la prédition de coût cummulative d'une trajectoire qui débute à x en utilisant la politique π
$J^*(x)$	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

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.

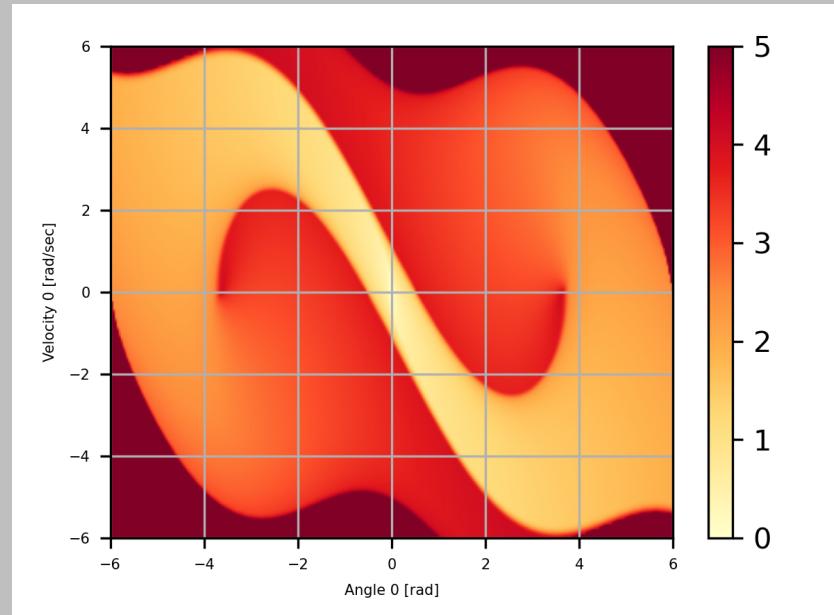


FIGURE 2.2 – Coût-àvenir optimal pour un pendule (temps-minimum)

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 le coût-àvenir en reculant étape par étape :

$$J_N^*(x_N) = g_N(x_N) \quad \forall x_N \in \mathcal{X}_N \quad (2.13)$$

$$\vdots \quad (2.14)$$

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

$$\vdots \quad (2.16)$$

$$J_0^*(x_0) = \min_{u_0 \in \mathcal{U}_0(x_0)} \left[g_0(x_0, u_0) + J_1^*(\underbrace{f_0(x_0, u_0)}_{x_1}) \right] \quad \forall x_0 \in \mathcal{X}_0 \quad (2.17)$$

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

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

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 :

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

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/d fz9k3BGrH0?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 optimale 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.

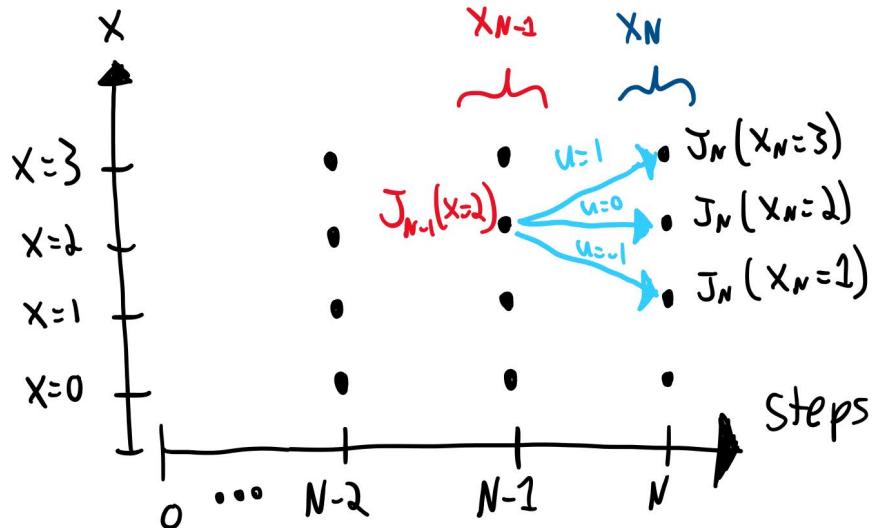


FIGURE 2.3 – Exemple de calcul de coût-àvenir pour l'état $x = 2$ à l'étape $k = N - 1$. Ici pour cet exemple on a 4 états discrets possibles et trois actions possibles. Comme illustré, on doit d'abord avoir évalué le coût terminal de tous les états à l'étape $k = N$, ensuite on calcule de coût-àvenir 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=P4hDvWSWoav6nW4x>

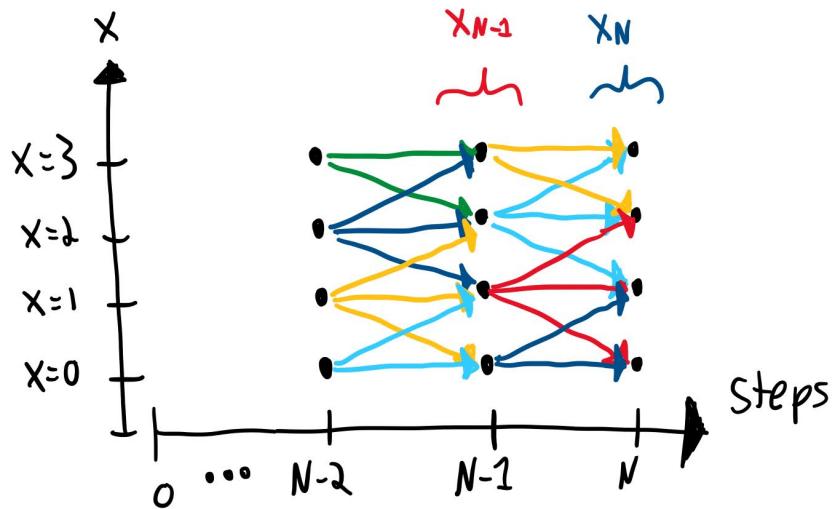


FIGURE 2.4 – L’algorithme de programmation dynamique consiste à exécuter ce calcul pour chaque état à chaque étape en débutant par la fin. Ici chaque couleur représente une étape de minimisation pour un état. Il y a une valeur $Q(x, u)$ pour chaque état-action qui sont ici les arcs, et une valeurs $J(x)$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

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

Robuste

Dans certaines situations, on peut préférer une formulation où l'espérance est remplacée par une opération de maximisation. C'est en fait la formulation *minimax* utilisée dans plusieurs algorithmes 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 travailler avec les valeurs Q , représentant un coût-àvenir 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 exacte 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

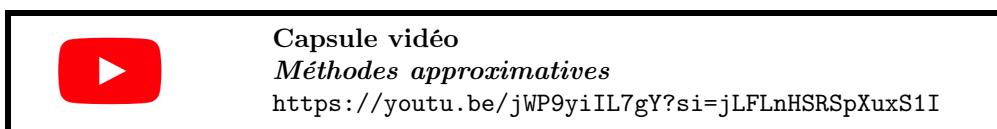
Comme illustré à la Figure 2.5, la plupart des outils visant à prendre des décisions intelligentes pour un agent, peuvent être vu comme une certaine forme de stratégie pour approximer un algorithme de programmation dynamique.

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

discretisation des actions
 Monte carlo
 Calcul déterministe

discretisation
 dégrégation
 Approximation hors-ligne
 (deep reinforcement learning)
 Recherche en-ligne
 (Rollout, MPC, etc.)

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



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 $\mathcal{W}_k(x)$:

$$w_k \in \mathcal{W}_k(x) \quad (3.2)$$

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 d'une fonction de densité de probabilité si la variable w_k est dans le domaine continu :

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

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 :

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

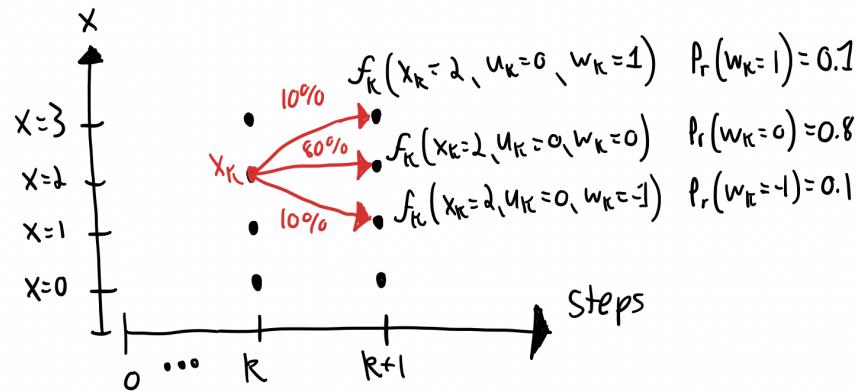


FIGURE 3.1 – Évolution stochastique avec des domaines discrets

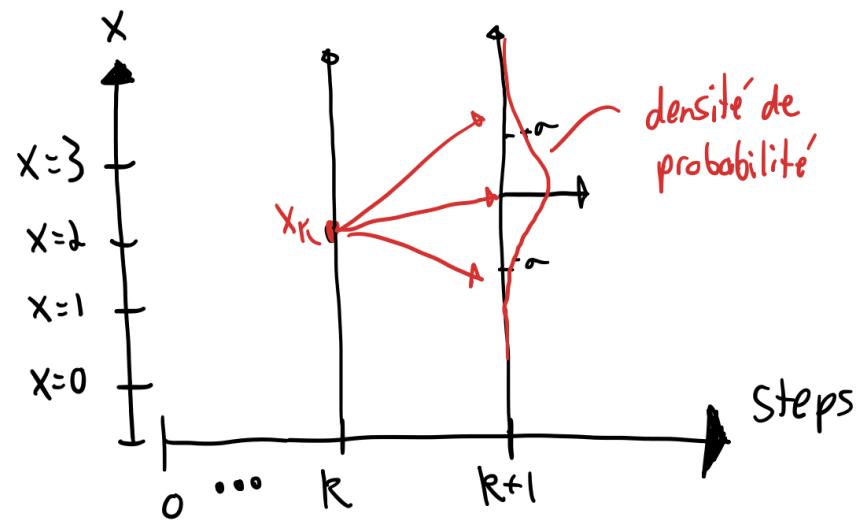


FIGURE 3.2 – Évolution stochastique avec des domaines continus

puisque :

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

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 :

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

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 décisions 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 coût-à-venir. 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-à-venir 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-à-venir 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 pour chaque option d'action :

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

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

3.3 Formulation minimax (commande robuste)

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

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

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

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 joueur, 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.

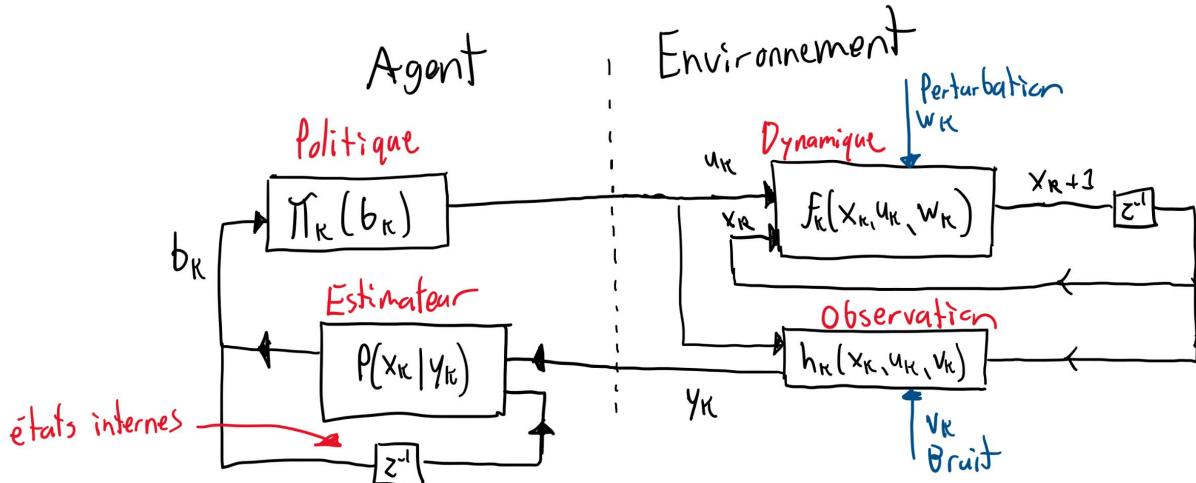


FIGURE 3.3 – Formulation mathématique incluant une fonction observation (utile pour représenter quand l'agent n'observe pas les états directement) et politique qui contient ses propres états et sa dynamique.

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.



Capsule vidéo
Observations partielles
<https://youtu.be/ZzT0L1Lqjio>

3.4.1 Espace croyance

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 :

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

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 que 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).

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

Je dit théoriquement car en pratique c'est généralement insoluble exactement même pour 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. On va généralement utiliser des méthodes approximées pour implémenter un estimateur, un exemple est le filtre de particules.

3.5 Politique Stochastique

Dans certaines situations, il peut avoir un avantage à avoir une politique qui est stochastique. Ça peut sembler contre-intuitif à première vue, et en fait si on regarde l'équation (3.9) il semblerait que agir optimalement doit être une fonction déterministe (sauf si plusieurs actions minimiseraient le cout-à-venir, alors choisir aléatoirement dans ce sous-ensemble serait aussi optimal). En effet, si on observe directement l'état d'un système la politique optimale est déterministe. Toutefois, lorsqu'on a de l'information partielle (soit on observe pas directement les états, soit on ne connaît pas la dynamique du système) il peut avoir un avantage à agir stochastiquement.

Lorsqu'on est dans un contexte de politique stochastique, on va réutiliser π mais pour indiquer la fonction de probabilité :

$$\pi_k(u|x) = P(u_k = u | x_k = x) \quad (3.18)$$

un petit abus de notation pour éviter d'introduire un nouveau symbole.

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 utilisés dans le contexte de processus de décision de Markov.



Capsule vidéo

Types de modèles d'évolution

https://youtu.be/D_HLuoPrD4w?si=CH3TcFrPsE6CH1DB

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 :

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

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 (*linear time invariant*) quand il n'y pas pas de dépendance directe au temps :

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

Aussi, une fonction de transfert dans le domaine de Laplace, peut 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 vers un modèle 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, i.e. une équation de différence.

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 utilisée dans ces notes, est une équation de différence, qui définit l'état à l'étape suivante basé 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 artificiels comme des jeux où il y a une notion de tour (comme les échecs). On arrive aussi à cette formulation lorsqu'on modélise des asservissements et qu'on se place du point de vue 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 les actions sont discrets, il est possible de représenter les états comme des noeuds sur un graphe et les actions possibles comme des arcs qui nous mène vers un autre état. Cette section présente le cas déterministe. Par exemple pour un jeu d'échec, il y a un nombre fini de positions pour une pièce, et un nombre fini d'actions possibles, et rien n'est stochastique. On peut donc utiliser le concept d'un graphe où les noeuds sont des états discrets possibles (numéroté avec un index s et les arcs des actions possibles (qu'on identifiera directement avec l'index de la destination de l'arc).

Notation : On va utiliser une notation spécifique pour les systèmes discrets, pour s'aligner partiellement avec la littérature en apprentissage par renforcement basée sur [Sutton and Barto, 2018]. Globalement, on va garder la notation alignée avec les ouvrages de programmation dynamique et commande optimale comme [Bertsekas, 2017] pour les valeurs réelles comme le coût instantané g et le coût à venir J , mais on va utiliser les variables s , s' , a et r pour référer aux indexées identifiant un état actuel, un état futur, une action et un coût (récompense). **Attention !** Certaines parties antérieures du matériel (comme les capsules vidéos) utilisent la notation i et j plutôt que s et s' .

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

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

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

$$J_k^*(x_k) \Leftrightarrow J_k^*(s) \quad (4.13)$$

$$\pi_k^*(x_k) \Leftrightarrow \pi_k^*(s) \quad (4.14)$$

$$g_k(x_k, u_k) \Leftrightarrow g_k(s, s') \text{ longueur de l'arc } ss' \quad (4.15)$$

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

$$J_N^*(s) = g_N(s) \quad \forall s \quad (4.16)$$

$$\vdots \quad (4.17)$$

$$J_k^*(s) = \min_{s' \in \mathcal{U}(s)} [g_k(s, s') + J_{k+1}^*(s')] \quad \forall s \quad (4.18)$$

$$\pi_k^*(s) = \operatorname{argmin}_{s' \in \mathcal{U}(s)} [g_k(s, s') + J_{k+1}^*(s')] \quad \forall s \quad (4.19)$$

Pour le cas discret, les fonctions peuvent être représentées par des vecteurs. Le coût-àvenir J peut être représenté par un vecteur où chaque élément $J(s)$ est le coût-àvenir de l'état s , la politique par un vecteur d'index où chaque élément $\pi(s)$ est un index de l'action optimale.

4.3.1 Discréétisation de variables continues

À venir !

4.4 Chaînes de Markov (états discrets stochastiques)

Lorsque les états sont discrets 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* (MDP). 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.

$$x_k \Leftrightarrow s \text{ index du noeud de départ} \quad (4.20)$$

$$u_k \Leftrightarrow a \text{ index de l'action} \quad (4.21)$$

$$x_{k+1} \Leftrightarrow s' \text{ index du noeud d'arrivé} \quad (4.22)$$

$$f_k(x_k, u_k, w_k) \Leftrightarrow p_k(s'|s, a) \text{ probabilités de transitionner vers } s' \quad (4.23)$$

$$g_k(x_k, u_k, w_k) \Leftrightarrow p_k(r|s, a) \text{ probabilités d'obtenir un coût instantané correspondant à l'index } r \quad (4.24)$$

4.4.1 Coût déterministe

Dans un premier temps, supposons que le coût (ou son espérance) est pleinement déterminé pour une transition de s vers s' avec l'action a . Dans ce cas, le problème est pleinement déterminé par les probabilités de transition vers s' :

$$p_k(s'|s, a) = P(x_{k+1} = s' | x_k = s, u_k = a) \quad (4.25)$$

et une carte des coûts donnée par

$$g_k = g_k(s, s', a) \quad (4.26)$$

qu'on peut encore voir comme les longueurs des arcs sur le graphe, mais avec comme différence avec le cas déterministe qu'il y a des arcs pour chaque action a . L'algorithme de programmation dynamique peut alors prendre la forme suivante, où l'opérateur de l'espérance peut être remplacé par une somme avec les probabilités :

$$J_k^*(s) = \min_{a \in \mathcal{U}(s)} \sum_{s'} p_k(s'|s, a) [g_k(s, s', a) + J_{k+1}^*(s')] \quad (4.27)$$

$$\pi_k^*(s) = \operatorname{argmin}_{a \in \mathcal{U}(s)} \sum_{s'} p_k(s'|s, a) [g_k(s, s', a) + J_{k+1}^*(s')] \quad (4.28)$$

Concrètement, le modèle de l'environnement dans cette situation peut être encodé comme un tenseur d'ordre 3 (si les probabilités sont constantes d'une étape à l'autre), ou chaque élément $p(s'|s, a)$ représente une probabilité de transition.



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

Le concept de valeur Q , représentant le coût-àvenir d'exécuter une action, peut alors aussi être défini par une somme :

$$Q(s, a) = \sum_{s'} p_k(s'|s, a) [g_k(s, s', a) + J_{k+1}^*(s')] \quad (4.29)$$

4.4.2 Coût stochastique

Pour la définition la plus large, on va considérer que le coût instantané est aussi probabiliste, mais peut prendre un nombre fini de valeurs $g(r) \in \mathbb{R}$ ou $r \in \mathbb{N}$ est l'index de la valeur de coût instantané. On peut alors définir le système par des probabilités de transiter de l'état s à l'état s' avec un coût $g(r)$ qui peuvent dépendre de l'action a et de l'étape actuelle k dans le cas le plus générique :

$$p_k(s', r|s, a) = P(x_{k+1} = s', g_k = g_r | x_k = s, u_k = a) \quad (4.30)$$

avec

$$x_k \Leftrightarrow s \text{ index du noeud de départ} \quad (4.31)$$

$$u_k \Leftrightarrow a \text{ index de l'action} \quad (4.32)$$

$$x_{k+1} \Leftrightarrow s' \text{ index du noeud d'arrivé} \quad (4.33)$$

$$f_k(x_k, u_k, w_k) \Leftrightarrow p_k(s', r|s, a) \text{ probabilités de transitionner vers } s' \text{ avec un coût } g_k(r) \quad (4.34)$$

$$g_k(x_k, u_k, w_k) \Leftrightarrow g_k(r) \text{ liste de coût possibles} \quad (4.35)$$

L'algorithme de programmation dynamique prend alors la forme suivante :

$$J_k^*(s) = \min_{a \in \mathcal{U}(s)} \sum_{s'} \sum_r p_k(s', r|s, a) [g_k(r) + J_{k+1}^*(s')] \quad (4.36)$$

$$\pi_k^*(s) = \operatorname{argmin}_{a \in \mathcal{U}(s)} \sum_{s'} \sum_r p_k(s', r|s, a) [g_k(r) + J_{k+1}^*(s')] \quad (4.37)$$

Notation du livre de Sutton et Barto On retrouve ici exactement la forme des équations qui est utilisée dans le livre séminal de Sutton et Barto [Sutton and Barto, 2018], la différence étant seulement la notation de coût g_k et coût-àvenir J vs. Sutton et Barto utilisent une récompense r et une valeur v . De plus, le livre de Sutton et Barto utilise directement le cas où l'horizon de temps est infini et rien n'est dépendant de l'étape actuelle, la situation spécifique qu'on explore au chapitre suivant.

Chapitre 5

Équations 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 dans un environnement statique, et non pas sur un horizon de temps fini comme c'était le cas dans les sections précédentes. L'environnement statique veux dire ici que la dynamique et la fonction coût sont les mêmes pour toutes les étapes :

$$f_k(x_k, u_k, w_k) \Rightarrow f(x, u, w) \quad (5.1)$$

$$g_k(x_k, u_k, w_k) \Rightarrow g(x, u, w) \quad (5.2)$$

Dans cette situation, une façon de définir ce problème est de formuler le coût à optimiser comme la limite suivante :

$$J^\pi(x) = \lim_{N \rightarrow \infty} \mathbb{E} \left[\sum_{k=0}^N \alpha^k g(x_k, u_k, w_k) \right] \quad \text{avec } x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k), w_k) \quad (5.3)$$

ou on introduit un facteur d'escampte $0 \leq \alpha \leq 1$ qui représente un biais pour moins prendre en considération les coûts/récompenses loins dans le futur. Le facteur d'escampte peut aussi être interprété comme une probabilité $(1 - \alpha)$ d'atteindre un état terminal sans coûts.

Dans ce contexte, sous certaines conditions, la limite existe et la politique optimale est aussi statique :

$$u_k = \pi_k(x_k) \Rightarrow u = \pi(x) \quad (5.4)$$

Un façon de voir les choses est que, pour un état actuel, le futur est équivalent peut importe l'étape actuelle, donc les décisions optimales n'ont pas de raisons d'être conditionnelles à l'étape actuelle.



Capsule vidéo

Horizon infini

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

Existence de la solution Un problème va être mal défini si il y a une possibilité que le coût/récompense diverge lorsque $N \rightarrow \infty$, concrètement cela implique aussi que les algorithmes d'apprentissages divergeraient eux aussi en tentant de résoudre le problème. Pour s'assurer que le problème est bien défini, la méthode la plus simple est de définir $\alpha < 1$, une autre est de garantir que le système va atteindre un état terminal avec un coût fini, sinon il existe aussi des formulations alternatives de coût moyen, voir [Bertsekas, 2017] pour les détails.

5.2 Équations de Bellman

La solution aux problèmes de commande optimale sur un horizon de temps infini, tel que décrit à la section précédente, est caractérisée par l'équation de Bellman :

$$J^*(x) = \min_u \mathbb{E}_w \left[g(x, u, w) + \alpha J^*(\underbrace{f(x, u, w)}_{x_{k+1}}) \right] \quad \forall x \quad (5.5)$$

$$\pi^*(x) = \operatorname{argmin}_u \mathbb{E}_w \left[g(x, u, w) + \alpha J^*(\underbrace{f(x, u, w)}_{x_{k+1}}) \right] \quad \forall x \quad (5.6)$$

qui est nécessaire et suffisante pour confirmer l'optimalité d'un coût-àvenir et d'une politique. Il est à noter, que les équations de Bellman, sont en fait tout simplement les itérations de l'algorithme de programmation dynamique mais sans les indexés ! En effet, plus on se projette sur un horizon de temps lointain, plus nos itérations de programmation dynamique vont converger vers une solution statique et l'équation de Bellman représente l'équilibre qui serait atteint.



Capsule vidéo
Équation de Bellman
<https://youtu.be/18KrN1HHT3E?si=F100xS1UC0KATgCs>



Capsule vidéo
Éléments de l'équation de Bellman
<https://youtu.be/18KrN1HHT3E?si=2ccHURCR-f8hHRUR>

La solution aux équations de Bellman est deux fonctions, J^* et π^* , qui peuvent être représentées par des vecteurs lorsque l'espace d'état \mathcal{X} est discret. Lorsque l'espace d'état est continu, il est possible de trouver des solutions exactes seulement dans des cas particuliers, discutés à la section 6. Lorsque l'espace d'état est discret, trouvez la solution implique de résoudre $N = |\mathcal{X}|$ (nombre d'états) équations non-linéaires (5.5), et il existe des algorithmes (présentés à la section 7) qui sont tractables lorsque N est relativement petit. Lorsque trouver des solutions exactes est impossible, on va plutôt tenter de trouver des approximations de fonction

Note : La solution J^* est unique, mais pas la politique π^* . En effet, deux actions peuvent être équivalentes et minimiser le coût, par exemple passer à gauche ou à droite d'un poteau bien centré sur notre chemin.

5.2.1 Variantes pour un processus de décision de Markov

Il existe plusieurs version des équations de Bellman, selon les hypothèses et le type de modèle.



Capsule vidéo
Variantes de l'équation de Bellman
https://youtu.be/98I0SI_jWyY?si=skah-1-PRdZibSPT

La forme la plus générique, pour un processus de décision de Markov, tel que décrit à la section 4.4, est

$$J^*(s) = \min_a \sum_{s'} \sum_r p(s', r | s, a) [g(r) + J^*(s')] \quad (5.7)$$

$$\pi^*(s) = \operatorname{argmin}_a \sum_{s'} \sum_r p(s', r | s, a) [g(r) + J^*(s')] \quad (5.8)$$

Si on suppose que le coût est déterminé par une transition, alors on peut écrire :

$$J^*(s) = \min_a \sum_{s'} p(s'|s, a) [g(s, s', a) + J^*(s')] \quad (5.9)$$

$$\pi^*(s) = \operatorname{argmin}_a \sum_{s'} p(s'|s, a) [g(s, s', a) + J^*(s')] \quad (5.10)$$

, et si l'évolution est déterministe, alors :

$$J^*(s) = \min_{s'} [g(s, s') + J^*(s')] \quad (5.11)$$

$$\pi^*(s) = \operatorname{argmin}_{s'} [g(s, s') + J^*(s')] \quad (5.12)$$

Aussi, on peut réécrire l'équation (5.10), fonction des facteurs Q :

$$Q^*(s, a) = \sum_{s'} p(s'|s, a) [g(s, s', a) + \min_{a'} Q^*(s', a')] \quad (5.13)$$

$$\pi^*(s) = \operatorname{argmin}_a Q^*(s, a) \quad (5.14)$$

5.3 Coût-àvenir d'une politique

Supposons que l'on cherche à évaluer le coût-àvenir d'une politique spécifique arbitraire :

$$u = \pi(x) \quad (5.15)$$

l'équation de Bellman peut être modifiée pour la tâche. Dans ce cas, l'opération de minimisation est retirée, et le calcul est plutôt de seulement calculer l'espérance d'exécuter la politique π à l'état x .

$$J^\pi(x) = \mathbb{E} \left[g(x, \underbrace{\pi(x)}_u, w) + \alpha J^\pi(f(x, \underbrace{\pi(x)}_u, w)) \underbrace{\qquad\qquad\qquad}_{x_{k+1}} \right] \quad (5.16)$$

À noter, que dans le cas où la politique est stochastique, il faut calculer l'espérance selon les probabilités de choix actions en plus des probabilités de perturbations.

5.3.1 Variantes pour une processus de décision de Markov

Sur une chaîne de Markov, pour évaluer une politique spécifique caractérisée par une politique déterministe $a = \pi(s)$, on a la forme :

$$J^\pi(s) = \sum_{s'} p(s'|s, \pi(s)) [g(s, s', \pi(s)) + \alpha J^\pi(s')] \quad (5.17)$$

Si la politique est stochastique et caractérisée par une probabilité $\pi(a|s)$, alors le calcul de l'espérance doit être modifié et on aurait :

$$J^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [g(s, s', a) + \alpha J^\pi(s')] \quad (5.18)$$

5.3.2 Solution matricielle

Les équations (5.17) et (5.18) peuvent être ramenée à un gros système d'équation linéaire. En effet, chaque termes peuvent être représentés par des vecteurs, matrices ou tenseurs. La politique serait une matrice où chaque élément π_{as} représente la probabilité de choisir l'action a à l'état s , les probabilités de transitions

seraient groupée dans un tenseur d'ordre 3 où chaque élément $p_{s'as}$ serait la probabilité de transitionner à l'état s' à partir de l'état s avec l'action a et un autre tenseur d'ordre 3 avec chaque élément $g_{s'as}$ représentant le coût instantanné d'une transition.

On pourrait d'abord calculer un vecteur d'espérance du coup instantané :

$$r_s^\pi = \sum_a \pi_{as} \sum_{s'} p_{s'as} g_{s'as} \quad (5.19)$$

et une matrice de probabilité de transition pour la politique :

$$p_{s's}^\pi = \sum_a \pi_{as} \sum_{s'} p_{s'as} \quad (5.20)$$

On peut alors réécrire le système d'équation (5.18) sous la forme :

$$J^\pi = r^\pi + \alpha P^\pi J^\pi \quad (5.21)$$

et on peut isoler le vecteur de coup-àvenir :

$$J^\pi = [I - \alpha P^\pi]^{-1} r^\pi \quad (5.22)$$

5.4 Équation de Hamilton–Jacobi–Bellman

En temps continu, on retrouve des équations qui sont similaires, mais impliquant des dérivées partielles par rapport au temps. L'équation est habituellement présentée dans un contexte déterministe. Pour un horizon de temps fini l'équivalent de l'algorithme de programmation dynamique exacte est une équation différentielle partielle qu'on peut intégrer de la fin vers le début pour trouver le coût-àvenir $J^*(x, t)$ et la politique $\pi^*(x, t)$ avec :

$$-\frac{\partial J^*(x, t)}{\partial t} = \min_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (5.23)$$

$$\pi^*(x, t) = \operatorname{argmin}_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (5.24)$$

Ensuite, si l'horizon de temps est infini, l'équivalent de l'équation de Bellman est :

$$0 = \min_u \left[g(x, u) + \frac{\partial J^*(x)}{\partial x} \underbrace{f(x, u)}_{\dot{x}} \right] \quad (5.25)$$

$$\pi^*(x) = \operatorname{argmin}_u \left[g(x, u) + \frac{\partial J^*(x)}{\partial x} \underbrace{f(x, u)}_{\dot{x}} \right] \quad (5.26)$$

qui représente l'équilibre lorsque l'horizon de temps tend vers l'infini.

Chapitre 6

Solutions Analytiques

Ce chapitre présente les solutions analytiques à l'équation de Bellman, pour les situations particulières où la dynamique est une relation linéaire et le coût une relation quadratique, connue sous l'acronyme LQR pour *Linear Quadratic Regulator* dans le domaine de la commande.

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{u}_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 on trouve que :

1) le coût-àvenir d'un état \underline{x}_k a la forme quadratique suivante :

$$J_k^*(\underline{x}_k) = \underline{x}_k^T S_k \underline{x}_k + c_k \quad S_k = S_k^T > 0 \quad \forall k \quad (6.5)$$

où S_k est une matrice symétrique qui caractérise le coût-àvenir à l'état \underline{x}_k et c_k est une constante qui ne dépend pas de l'état actuel.

2) la loi de commande optimale a la forme linéaire suivante :

$$\pi_k^*(x_k) = -K_k x_k \quad (6.6)$$

où K_k est la matrice de gain égale à

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

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 :

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

Une équation nommée l'équation de Riccati en temps discret.

6.1.1 Horizon infini

Si le problème ne dépend pas directement de l'étape, i.e. toutes les matrices A , B , Q et R sont constantes, et qu'on cherche la solution optimale sur un horizon de temps infini, alors la politique optimale va aussi être une matrice de gain constant :

$$\pi^*(\underline{x}) = -K \underline{x} \quad (6.9)$$

où K est la matrice de gain égale à

$$K = [R + B^T S B]^{-1} B S A \quad (6.10)$$

avec S étant la solution de l'équation nommée l'équation de Riccati algébrique en temps discret (DARE) :

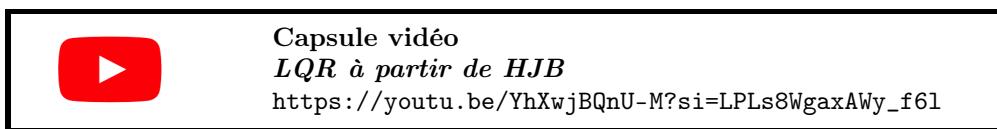
$$S = Q + A^T \left(S - S^T B^T [R + B^T S B]^{-1} B S \right) A \quad (6.11)$$

qui est en fait l'équation de Bellman pour ce cas particulier.

Existence de la solution Dans ce contexte, il n'est pas typique d'utiliser un facteur d'escompte α . L'existence d'un coût-àvenir fini peut être garantie par le critère de contrôlabilité du système, qui est une propriété qui se calcule basé sur la matrice A et B . Celle-ci garantie qu'il existe une séquence d'action possible pour amener le système dynamique sur un état cible arbitraire. Dans ce cas c'est relié au fait qu'il existe donc nécessairement une solution ou un coût-àvenir arrête de croître dans le temps si on amène l'état à l'origine $x = 0$ qui est un point sans coût.

6.2 LQR à temps continu

En temps continu, on trouve aussi une équation analytique à l'équation HJB (voir section 5.4), si la dynamique est linéaire et le coût quadratique.



Donc, plus précisément, dans le contexte d'un système dynamique à états et actions continues où la dynamique est linéaire :

$$\dot{\underline{x}} = A(t)\underline{x} + B(t)\underline{u} + \underline{w} \quad (6.12)$$

où \underline{x} et \underline{w} sont des vecteurs de dimension n et \underline{u} un vecteur de dimension m . Le vecteur \underline{w} 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 :

$$\mathbb{E} [\underline{w}] = 0 \quad (6.13)$$

De plus A et B sont des matrices caractérisant la dynamique, qui peuvent dépendre du temps. Si on cherche donc à minimiser l'espérance du coût-à-venir :

$$J = \mathbb{E} \left[\int_{t=0}^{t_f} \left(\underbrace{\underline{x}^T Q(t) \underline{x} + \underline{u}^T R(t) \underline{u}}_{g(\underline{x}, \underline{u}, t)} \right) + \underbrace{\underline{x}_f^T S_f \underline{x}_f}_{h(\underline{x}_f, t_f)} \right] \quad (6.14)$$

où les matrices $Q(t)$ et $R(t)$ sont symétriques et définies positives :

$$Q(t) \geq 0 \quad R(t) > 0 \quad \forall t \quad (6.15)$$

et peuvent déprendre du temps.

Note : La solution qu'on dérive ici est valide même si les matrice A , B , Q et R sont des fonctions du temps. On va toutefois omettre le (t) dans les équations suivantes pour alléger la présentation.

En partant du coût-à-venir optimal au temps terminal, qui est nécessairement directement la fonction de coût terminal définie par le problème :

$$J^*(x, t_f) = \underline{x}^T S_f \underline{x} \quad (6.16)$$

on peut retrouver la solution optimale en intégrant l'équation de HJB (voir section 5.4) qui peut être simplifiée :

$$-\frac{\partial J^*(x, t)}{\partial t} = \min_u \mathbb{E} \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, w, t))}_{\dot{x}} \right] \quad (6.17)$$

$$-\frac{\partial J^*(x, t)}{\partial t} = \min_u \mathbb{E} \left[\underline{x}^T Q \underline{x} + \underline{u}^T R \underline{u} + \frac{\partial J^*(x, t)}{\partial x} \underbrace{(A \underline{x} + B \underline{u} + \underline{w})}_{\dot{x}} \right] \quad (6.18)$$

Un résultat, **probablement le plus important de tout le domaine de la commande optimale !**, est que la fonction coût-à-venir optimale va avoir la forme quadratique suivante :

$$J^*(x, t) = \underline{x}^T S(t) \underline{x} \quad (6.19)$$

et la politique optimale va avoir la forme linéaire suivante :

$$\pi^*(x, t) = - \underbrace{[R^{-1} B^T S(t)]}_{K(t)} \underline{x} \quad (6.20)$$

Les deux solutions décrites ci-dessous sont fonctions d'une matrice variable dans le temps $S(t)$ qui peut être calculée en intégrant dans le temps l'équation différentielle suivante :

$$-\dot{S}(t) = S(t)A + A^T S(t) - S(t)BR^{-1}B^T S(t) + Q \quad (6.21)$$

qui est la condition pour que l'équation (6.18) soit respectée. Cette équation est connue sous le nom d'équation de Riccati différentiel en temps continu. On intègre cette équation à partir de la valeur au temps terminal qui correspond au coût terminal défini par le problème :

$$S(t_f) = S_f \quad (6.22)$$

6.2.1 Horizon infini

Si le problème ne dépend pas directement de l'étape, i.e. toutes les matrices A , B , Q et R sont constantes, et qu'on cherche la solution optimale sur un horizon de temps infini, alors la politique optimale va aussi être une matrice de gain constant :

$$\pi^*(\underline{x}) = -K^* \underline{x} \quad (6.23)$$

où K est la matrice de gain égale à

$$K^* = R^{-1} B^T S \quad (6.24)$$

avec S étant la solution de l'équation nommée l'équation de Riccati algébrique en temps continu (CARE) :

$$0 = SA + A^T - SBR^{-1}B^T S + Q \quad (6.25)$$

qui est en fait l'équation de Bellman pour ce cas particulier. Il existe des méthodes numériques efficaces pour résoudre cette équation matricielle quadratique. La solution existe si le système dynamique a une propriété qu'on appelle la contrôlabilité, reliée aux matrices A et B .

Note : Lorsqu'on parle d'un contrôleur LQR en général sans précision, on réfère généralement à la solution statique pour un horizon de temps infini.

6.2.2 Stabilisation de trajectoires

La solution LQR pour un horizon de temps fini, permet de résoudre des problèmes de commande non-linéaire en linéarisant autour d'une trajectoire temporelle. En effet, supposons qu'on désire concevoir une politique pour stabiliser un système dynamique non-linéaire :

$$\dot{\underline{x}} = f(\underline{x}, \underline{u}) \quad (6.26)$$

sur une trajectoire nominale :

$$\underline{x}_d(t) \quad \underline{u}_d(t) \quad \text{avec} \quad \dot{\underline{x}}_d = f(\underline{x}_d, \underline{u}_d) \quad (6.27)$$

La dynamique de l'erreur :

$$\underline{x}_e = \underline{x}_d - \underline{x} \quad (6.28)$$

est égale à

$$\dot{\underline{x}}_e = \dot{\underline{x}}_d - \dot{\underline{x}} \quad (6.29)$$

$$\dot{\underline{x}}_e = f(\underline{x}_d, \underline{u}_d) - f(\underline{x}, \underline{u}) \quad (6.30)$$

Et si on fait une expansion de Taylor pour approximer la dynamique autour de la trajectoire nominale, on trouve que la dynamique de l'erreur prend une forme linéaire mais dépendant du temps qui peut être résolu avec la solution LQR pour un horizon de temps fini.

$$\dot{\underline{x}}_e = f(\underline{x}_d, \underline{u}_d) - f(\underline{x}, \underline{u}) \quad (6.31)$$

$$\dot{\underline{x}}_e \approx f(\underline{x}_d, \underline{u}_d) - \left(f(\underline{x}_d, \underline{u}_d) + \left[\frac{\partial f(x, u)}{\partial x} \right]_{(\underline{x}_d, \underline{u}_d)} \Delta \underline{x} + \left[\frac{\partial f(x, u)}{\partial u} \right]_{(\underline{x}_d, \underline{u}_d)} \Delta \underline{u} + h.o.t. \right) \quad (6.32)$$

$$\dot{\underline{x}}_e \approx A(t) \Delta \underline{x} + B(t) \Delta \underline{u} \quad (6.33)$$

Donc pour un système non-linéaire, sur une trajectoire nominale spécifique, le problème de stabilisation d'une erreur relative à cette trajectoire peut être attaqué avec la solution LQR à horizon de temps fini, en

linéarisant la dynamique sur plusieurs points pour obtenir une approximation qui correspond à un système linéaire avec des matrices A et B qui varient dans le temps.

Exemple d'implémentation : Voir <https://github.com/SherbyRobotics/pyro/blob/master/pyro/control/lqr.py>

Chapitre 7

Algorithmes de planification

Ce chapitre présente des algorithmes pour trouver des solutions exactes aux équations de Bellman, lorsque les états et les actions ont des domaines discrets.

7.1 Algorithme d’itération de valeurs



Capsule vidéo
Iteration de valeurs
<https://youtu.be/SpWKHB4GupU?si=Ewb6MwRTTE0lero0>

L’algorithme d’itération de valeurs, consiste à initialiser arbitrairement un vecteur $J(x)$ de valeurs de coûts-àvenir pour chaque état x , et ensuite itérer avec l’opération de programmation dynamique suivante :

$$J(x) \leftarrow \min_u \mathbb{E}_w \left[g(x, u, w) + \alpha J(\underbrace{f(x, u, w)}_{x_{next}}) \right] \quad \forall x \quad (7.1)$$

jusqu’à la convergence des valeurs du vecteur $J(x)$ selon un certain seuil de tolérance. Le point fixe de ces itérations correspond à l’équation de Bellman et donc $J(x)$ converge vers le cout-àvenir optimal $J^*(x)$. Ensuite, la politique optimale peut être calculée avec :

$$\pi^*(x) = \operatorname{argmin}_u \mathbb{E}_w \left[g(x, u, w) + \alpha J^*(\underbrace{f(x, u, w)}_{x_{next}}) \right] \quad \forall x \quad (7.2)$$

On peut interpréter l’algorithme d’itération de valeur comme simplement utiliser l’algorithme de programmation dynamique exacte pour calculer les valeurs optimales en reculant dans le temps étape par étapes, jusqu’à atteindre un point au l’horizon de temps est très très grand et donc que la solution correspond approximativement à un horizon de temps infini. Il est toutefois à noter que la valeur d’initialisation, qui correspond à un coût terminal selon cette interprétation, n’a pas d’impact sur la valeur finale après convergence de l’algorithme.

7.1.1 Conditions de convergence

L’algorithme d’itération de valeur peut ne pas converger si il y a possibilité d’une situation où un coût peut tendre vers l’infini sur une trajectoire. Une première façon de garantir que ce n’est pas le cas est si la fonction de coût instantanée est bornée par une valeur finie (i.e. n’est jamais infini) et si le facteur d’escompte est inférieur à 1 :

$$g(x, u, w) < G \quad \text{et} \quad 0 < \alpha < 1 \quad (7.3)$$

Alternativement, si on veut utiliser un facteur d'escompte égale à un, une autre méthode pour garantir la convergence est de s'assurer que le système va inévitablement converger sur un état terminal avec un coût fini.

7.1.2 Évaluation de politique

On peut aussi utiliser cet algorithme pour évaluer le cout-àvenir d'une politique spécifique avec les itérations suivantes :

$$J(x) \Leftarrow \mathbb{E}_w \left[g(x, \pi(x), w) + \alpha J(\underbrace{f(x, \pi(x), w)}_{x_{next}}) \right] \quad \forall x \quad (7.4)$$

pour converger sur $J^\pi(x)$.

7.2 Algorithme d'itération de politique



Capsule vidéo
Iteration de politique

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

Un algorithme alternatif est l'algorithme d'itération de politique. L'idée est de débuter avec une politique arbitraire π . Ensuite on alterne entre une étape d'évaluation et une étape d'amélioration. L'étape d'évaluation consiste à faire un certain nombre d'itérations avec l'algorithme d'évaluation de politique :

$$J^\pi(x) \Leftarrow \mathbb{E}_w \left[g(x, \pi(x), w) + \alpha J^\pi(\underbrace{f(x, \pi(x), w)}_{x_{next}}) \right] \quad \forall x \quad (7.5)$$

Ensuite, on va améliorer la politique en choisissant l'action qui optimise le coût-àvenir estimé actuel :

$$\pi'(x) = \operatorname{argmin}_u \mathbb{E}_w \left[g(x, u, w) + \alpha J^\pi(\underbrace{f(x, u, w)}_{x_{next}}) \right] \quad \forall x \quad (7.6)$$

Ensuite on retourne à l'étape d'amélioration, ou bien l'algorithme termine si l'algorithme la politique est fixe à l'étape d'amélioration.

Chapitre 8

Algorithmes d'apprentissage

8.1 Évaluation

8.1.1 Monte-carlo

8.1.2 TD-Learning

8.1.3 Sarsa

8.2 Optimisation

8.3 Q-Learning



Capsule vidéo

Q-Learning

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

8.4 Exploration

8.4.1 Exploitation vs. exploration

n-armed bandit

À venir !

8.5 Approximation de fonctions



Capsule vidéo

RL avec approximation

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



Capsule vidéo

Approximation de fonctions

<https://youtu.be/p8BizsV8apQ?si=Tj0lqSOQZwZmtn9u>

Annexe A

Outils mathématiques

A.1 Ensembles

On note qu'une variable x peut prendre des valeurs dans un ensemble X de valeurs possibles avec le symbole \in ainsi :

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

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

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

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

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

ou excluant les bornes :

$$\mathcal{X} = (0, 10) \quad \Rightarrow \text{ex : } 0 \notin \mathcal{X}, 2 \in \mathcal{X}, 9.999999999 \in \mathcal{X} \quad (\text{A.4})$$

ou bien avec une condition :

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

qu'on peut lire, l'ensemble des nombres réels qui sont plus grands ou égaux à zéro et plus petits et égaux à dix.



Capsule vidéo
Ensembles - Notions de base
<https://https://youtu.be/PtQNvbycVLc>

A.2 Probabilités



Capsule vidéo
Probabilités - Notions de base
<https://youtu.be/a2jBknWV1Vw>

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'ensemble. 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 :

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

Voici quelques propriété utile de l'opérateur espérance :

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

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

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

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 évènements, i.e. que deux choses aléatoire arrive 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.

Annexe B

Approximation de fonctions

À venir !

Annexe C

Exercises

C.1 Introduction et 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 :

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

représentant un erreur de positionnement au carré plus une pénalité quadratique pour utiliser de l'énergie. Toutefois, vous voulez utiliser un algorithme programmé pour maximiser une fonction récompense, comment modifier votre fonction avant de la fournir à l'algorithme pour qu'il résoudre votre problème ? Prouvez que c'est équivalent.

C.1.2 Limites de la formulation coût/récompense cumulative

Selon vous est-ce que la formulation d'un problème comme minimiser/maximiser un coût/récompense additif, étape par étape est limitée ? i.e. est-ce que certains objectifs pourrait ne pas être formulé sous cette forme ? De plus, réfléchissez spécifiquement aux situations suivantes :

1. Imaginons un jeux de hasard où les récompenses serait multipliée à chaque tour, est-ce qu'on peut représenter le problème avec une forme cumulative ?
2. J'ai un problème multi-objectif, est-ce que je peux vraiment représenter mon problème avec une fonction coût scalaire ?

C.1.3 Formes possibles des politiques optimales

Identifier et décrivez une situation ou la politique optimale (selon votre bon sens, pas besoin de faire de calculs..) devrait :

1. Dépendre directement du temps ou de l'étape actuelle ?
2. Être stochastique ?

Donnez un exemple et un contre-exemple pour chaque situation si possible.

C.1.4 Learn to fly

Play with the following code example :



Exercice de code
Planar Drone with PPO
https://colab.research.google.com/drive/1-EuRPiyf2Gv0n8p_17j1vvEIE188WT1m?usp=sharing

1. How much training steps do you have to use to learn an adequate policy for the drone ?
2. Verify if the policy works well for various initial conditions.

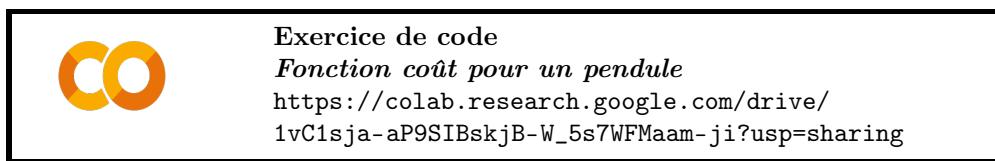
3. Look at the learned policy plot, do you understand the control law ? does it look familiar ?
4. Try changing the code to have a more aggressive flight, assume your drone can be as powerful as you like :)

C.1.5 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 tous 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

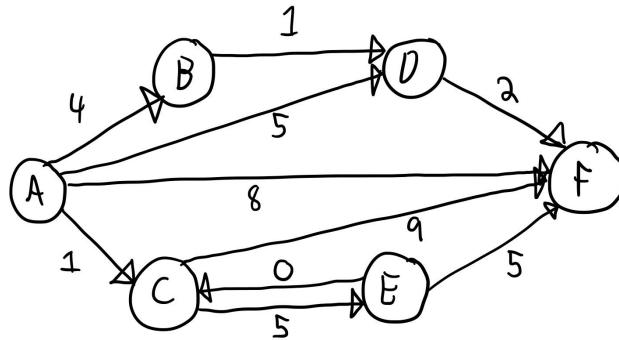


FIGURE C.1 – Graphique qui représente des chemins possibles pour aller vers la position *F*

Appliquez l'algorithme de programmation dynamique exacte :

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

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

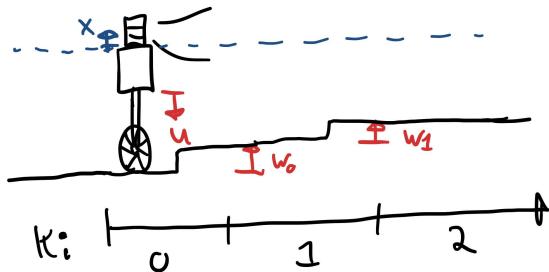
pour résoudre les actions optimales (choix de l'arc à suivre) pour se rendre à l'état cible (Noeud *F*) à partir de l'état actuel (les Noeuds $x_k \in [A, B, C, D, E]$) et de l'index de temps actuel k . Le coût de chaque option de chemin est représenté par le chiffre indiqué pour chaque arc sur le graphique 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	$N = 5$
$J^*(A) =$						
$\pi^*(A) =$						
$J^*(B) =$						
$\pi^*(B) =$						
$J^*(C) =$						
$\pi^*(C) =$						
$J^*(D) =$						
$\pi^*(D) =$						
$J^*(E) =$						
$\pi^*(E) =$						
$J^*(F) =$						

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 de coût à optimiser est définie comme :

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

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

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

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	$N = 2$
$J^*(x_k, w_k) =$			
$\pi^*(x_k, w_k) =$			

C.2.3 Politique optimale pour un thermostat

Compétences à développer :

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



Capsule vidéo
Politique optimale de chauffage (solution à ce problème (à quelques détails près))
<https://youtu.be/QwXjiAzDENs?si=9A9qe-DvUuZy5juw>

Nous désirons programmer un thermostat pour chauffer une maison. Imaginons une situation simplifiée suivante, vous arrivez à la maison à 17h (étape finale $N = 2$) et on cherche à optimiser la puissance de chauffage u_0 sélectionnée à 15h (étape initial $k = 0$) et u_1 sélectionnée à 16h (étape intermédiaire $k = 1$). L'évolution de la température x (relative à la température extérieure) de la maison est donné par les équations dynamiques suivantes (on utilise un pas de temps d'heure en heure) :

$$\text{Dynamique : } x_1 = ax_0 + u_0 \quad (\text{C.10})$$

$$x_2 = ax_1 + u_1 \quad (\text{C.11})$$

Supposons qu'il y a une tarification dynamique, qu'on veux sauver de l'argent, mais qu'on veut aussi avoir une température confortable à notre arrivée. On peut formuler la fonction de coût suivante :

$$\text{Coûts : } g_0(x_0, u_0) = r_0 u_0^2 \quad (\text{C.12})$$

$$g_1(x_1, u_1) = r_1 u_1^2 \quad (\text{C.13})$$

$$g_2(x_2) = q_2(x_2 - x_d)^2 \quad (\text{C.14})$$

On suppose que le coût de chauffage est proportionnel au carré de la puissance, et il y a possiblement un tarif à 15h et à 16h donnés par r_0 et r_1 . De plus, supposons que la valeur monétaire de notre confort à 17h est caractérisé par la variable q_2 qui multiplie le carré de l'écart de la température avec la température ambiante désirée x_d .

La politique à concevoir consiste en deux fonctions qui choisissent la puissance de chauffage en fonction de la température mesurée :

$$\text{Politiques } u_0 = \pi_0(x_0) \quad (\text{C.15})$$

$$u_1 = \pi_1(x_1) \quad (\text{C.16})$$

On cherche la politique optimale qui minimise le coût cumulatif suivant :

$$J(x_0, x_1, x_2, u_0, u_1) = g_0(x_0, u_0) + g_1(x_1, u_1) + g_2(x_2) \quad (\text{C.17})$$

Calculez le coût-àvenir et la politique optimale à l'étape $k = 0$ et à l'étape $k = 1$. Le tableau suivant peut aider à synthétiser vos résultats :

$k =$	0	1	$N = 2$
$J^*(x) =$			
$\pi^*(x) =$			

Ensuite, analysez les solutions des cas limites suivants :

1. $a = 0$ une maison sans aucune isolation (simplifiez avec $r_0 = r_1 = q_2 = 1$)
2. $a = 1$ une maison parfaitement isolée (simplifiez avec $r_0 = r_1 = q_2 = 1$)

C.2.4 Chemin le plus court 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

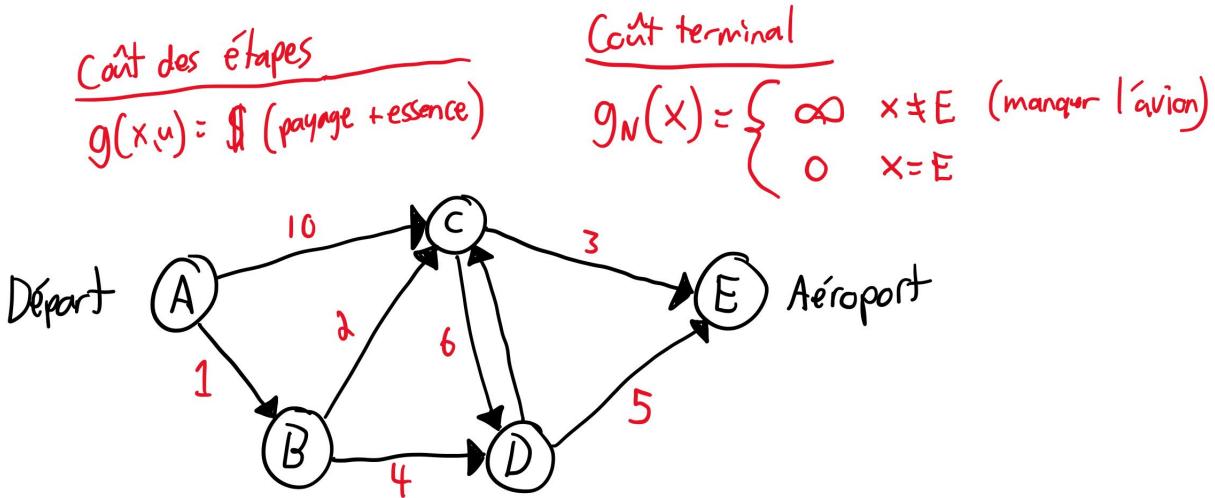


FIGURE C.2 – Problème d’optimisation d’un trajet vers l’aéroport

Appliquez l’algorithme de programmation dynamique exacte :

$$J_k^*(x_k) = \min_{u_k} \left[g_k(x_k, u_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k)}_{x_{k+1}} \right) \right] \quad (\text{C.18})$$

$$\pi_k^*(x_k) = \operatorname{argmin}_{u_k} \left[g_k(x_k, u_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k)}_{x_{k+1}} \right) \right] \quad (\text{C.19})$$

pour résoudre les actions optimales (choix de l’arc à suivre) pour se rendre à l’aéroport (Noeud E) à partir de l’état actuel (les Noeuds $x_k \in [A, B, C, D]$) 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 E à $k = 5$. Le tableau suivant peut vous aider pour synthétiser les résultats :

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

C.3 Commande stochastique

C.3.1 Loi de commande pour une suspension active II

Compétences à développer :

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

Ici on reprend le même problème qu'à l'exercice C.2.2, 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 :

$$P(w_0 = 1) = 0.5 \quad (\text{C.20})$$

$$P(w_0 = 0) = 0.5 \quad (\text{C.21})$$

$$P(w_1 = 1) = 0.5 \quad (\text{C.22})$$

$$P(w_1 = 0) = 0.5 \quad (\text{C.23})$$

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-à-venir :

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad (\text{C.24})$$

avec l'algorithme de programmation dynamique suivant :

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k) + J_{k+1}^* \underbrace{(f_k(x_k, u_k))}_{x_{k+1}} \right] \quad (\text{C.25})$$

$$\pi_k^*(x_k) = \operatorname{argmin}_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k) + J_{k+1}^* \underbrace{(f_k(x_k, u_k))}_{x_{k+1}} \right] \quad (\text{C.26})$$

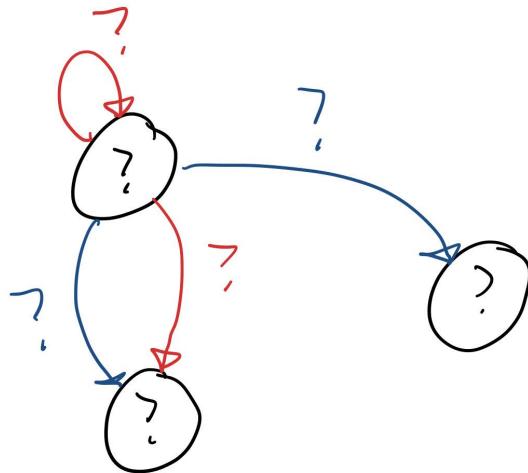
Le tableau suivant peut vous aider pour synthétiser les résultats :

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

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. Illustrer 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.3.3 Stratégie optimale aux échecs

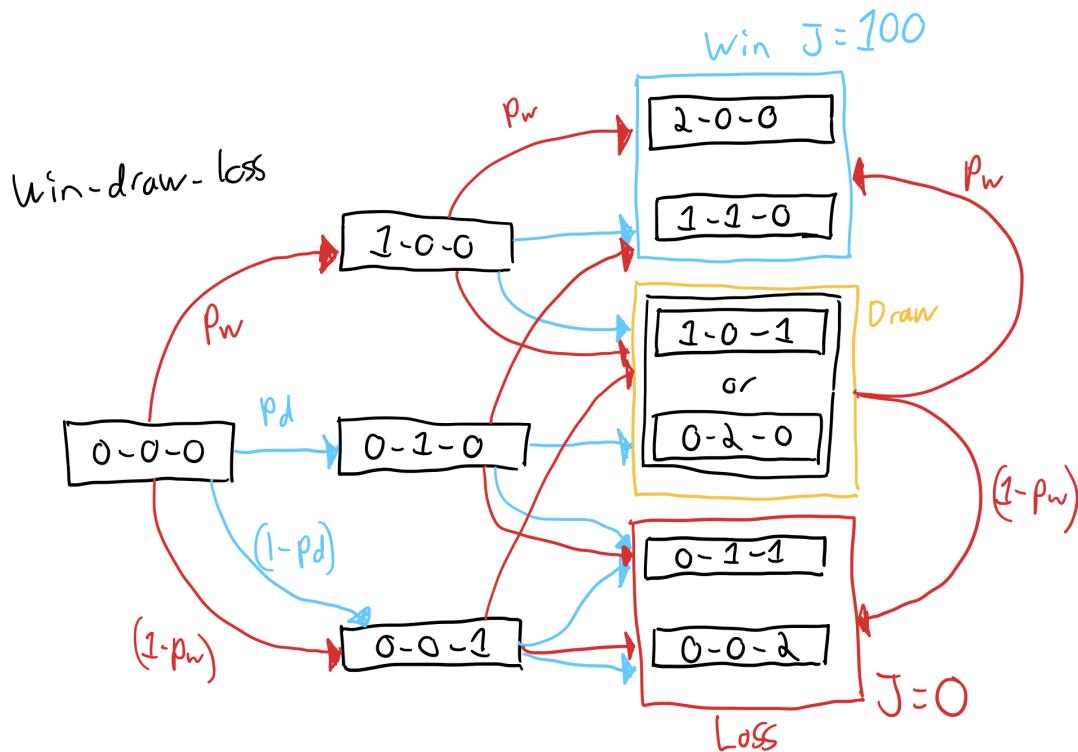
Note : Adapté d'un exemple dans le livre de D. Bertsekas

Vous allez jouer une série de deux parties d'échec, et vous voulez optimiser votre choix de stratégie à chaque partie, pour maximiser vos chances de gagner. Si après deux parties c'est l'égalité, alors on rejoue et le premier joueur qui gagne une partie gagne la série. Supposons que vous connaissez la performance de vos stratégies. Votre stratégie défensive mène à une partie nulle avec une probabilité p_d , à une défaite avec une probabilité $1 - p_d$ et jamais à une victoire. Votre stratégie agressive mène à une victoire avec une probabilité p_w , une défaite avec probabilité $1 - p_w$ et jamais à une partie nulle. Si $p_d = 0.99$ et $p_w = 0.49$, déterminez la probabilité de gagner la série avec les politiques suivantes :

1. Toujours utiliser la stratégie agressive
2. Toujours utiliser la stratégie défensive
3. Utiliser la stratégie agressive d'abord et la stratégie défensive pour la deuxième partie
4. Choisir la stratégie optimale en fonction de l'état

Note : Supposez que la stratégie agressive est toujours utilisée lors d'un bris d'égalité.

Est-ce que c'est possible d'avoir plus de 50% de chances de gagner la série même si individuellement votre probabilité de victoire est inférieur à 50% pour une partie ?

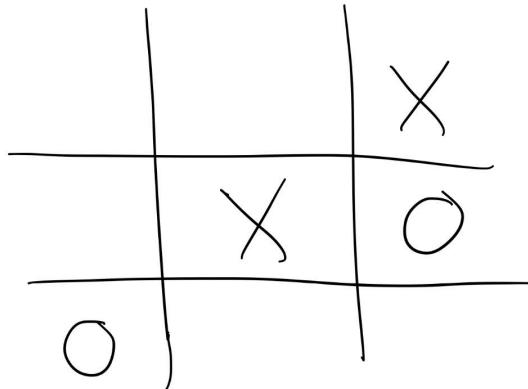


C.4 Commande robuste

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.

$$J_k^*(x_k) = \max_X \min_O \left[g_k + J_{k+1} = \begin{cases} +1 & \text{pour une position gagnante pour X} \\ 0 & \text{pour une grille pleine sans vainqueur} \\ -1 & \text{pour une position gagnante pour O} \\ J_{k+1}^*(x_{k+1}) & \text{pour une position non terminale} \end{cases} \right] \quad (\text{C.27})$$

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 :

$$\underline{x}_{k+1} = f_k(\underline{x}_k, \underline{u}_k, \underline{w}_k) = A_k \underline{x}_k + B_k \underline{u}_k + \underline{w}_k \quad (\text{C.28})$$

où \underline{x}_k et \underline{w}_k sont des vecteurs de dimension n et \underline{u}_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 :

$$\mathbb{E}[\underline{w}_k] = 0 \quad (\text{C.29})$$

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

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} \left(\underbrace{\underline{x}_k^T Q_k \underline{x}_k + \underline{u}_k^T R_k \underline{u}_k}_{g_k(\underline{x}_k, \underline{u}_k, \underline{w}_k)} \right) + \underbrace{\underline{x}_N^T Q_N \underline{x}_N}_{g_N(\underline{x}_N)} \right] \quad (\text{C.30})$$

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

$$Q_k \geq 0 \quad R_k > 0 \quad (\text{C.31})$$

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 :

$$J_k^*(\underline{x}_k) = \underline{x}_k^T S_k \underline{x}_k + c_k \quad S_k = S_k^T > 0 \quad \forall k \quad (\text{C.32})$$

où S_k est une matrice symétrique qui caractérise le coût-à-venir à l'état \underline{x}_k et c_k est une constante qui ne dépend pas de l'état actuel.

2) la loi de commande optimale a la forme linéaire suivante :

$$\underline{u}_k^* = c_k^*(\underline{x}_k) = -K_k \underline{x}_k \quad (\text{C.33})$$

où K_k est la matrice de gain égale à

$$K_k = [R_k + B_k^T S_{k+1} B_k]^{-1} B_k^T S_{k+1} A_k \quad (\text{C.34})$$

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 :

$$S_k = Q_k + A_k^T \left(S_{k+1} - S_{k+1} B_k [R_k + B_k^T S_{k+1} B_k]^{-1} B_k^T S_{k+1} \right) A_k \quad (\text{C.35})$$

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 $1 \times n$;

$$\underline{z} = \frac{\partial y}{\partial \underline{x}} = \left[\begin{array}{ccc} \frac{\partial y}{\partial x_1} & \dots & \frac{\partial y}{\partial x_n} \end{array} \right] \Leftrightarrow z_i = \frac{\partial y}{\partial x_i} \quad (\text{C.36})$$

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.5.2 LQR en temps continu

Pour le problème de LQR en temps continu présenté à la section 6.2, démontrez que la politique optimale a la forme :

$$\underline{u} = -\frac{1}{2} R^{-1} B^T \left[\frac{\partial J^*}{\partial \underline{x}} \right]^T \quad (\text{C.37})$$

Ensuite, basée sur l'hypothèse que la solution est

$$J^*(\underline{x}, t) = \underline{x}^T S(t) \underline{x} \quad (\text{C.38})$$

1. Calculez

$$\frac{\partial J^*}{\partial \underline{x}} = ? \quad (\text{C.39})$$

2. Calculez

$$\frac{\partial J^*}{\partial t} = ? \quad (\text{C.40})$$

3. Vérifiez que J est bien une solution à l'équation HJB, sous condition que la matrice $S(t)$ soit une solution à l'équation différentielle de Riccati.
4. Vérifiez si la solution change si on considère que l'évolution dynamique est stochastique et caractérisée par un bruit additif normal w , i.e. $\dot{x} = Ax + Bu + w$ avec $w \sim \mathcal{N}(0, \sigma^2)$

Solution à un exercice similaire :



Capsule vidéo
LQR à partir de HJB
https://youtu.be/YhXwjBQnU-M?si=LPLs8WgaxAWy_f61

C.5.3 Implémentation LQR

Basé sur le code source <https://github.com/SherbyRobotics/pyro/blob/master/pyro/control/lqr.py> et l'exemple suivant :



Exercice de code
LQR cart-pole démo
<https://colab.research.google.com/drive/1-qslmY5MRogkf8YL9TRAH1R2xLgfrwNH?usp=sharing>

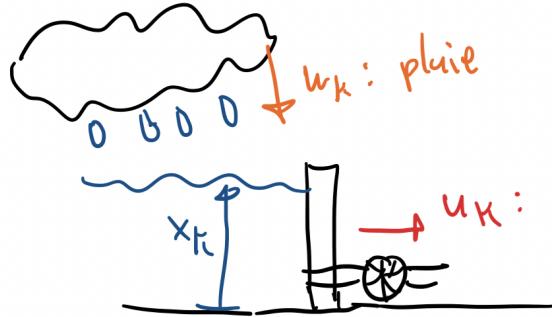
1. Analysez le code source qui implémente les solutions LQR
2. Exécutez le code de l'exemple et analysez le comportement de la politique LQR pour un horizon de temps infini
3. Simulez la trajectoire pour diverses conditions initiales
4. Exécutez le code de l'exemple et analysez le comportement de la politique LQR pour un horizon de temps fini
5. Simulez la trajectoire pour diverses conditions initiales
6. Dans le code source de la version à horizon fini, la vraie équation à intégrer pour la matrice S a été remplacée par une approximation. Quel problème peut survenir lors de cette intégration ?

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 :

$$x_{k+1} = \min [x_k - u_k + w_k, 4] \quad (\text{C.41})$$

où

$$x_k \in \{0, 1, 2, 3, 4\} \quad (\text{C.42})$$

$$u_k \in \{0, 1, 2\} \quad (\text{C.43})$$

$$u_k \leq x_k \quad (\text{C.44})$$

$$w_k = \begin{cases} 0 & P = 0.4 \\ 1 & P = 0.4 \\ 3 & P = 0.2 \end{cases} \quad (\text{C.45})$$

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 :

$$g_k = \begin{cases} 0 & \text{si } u_k = 2 & \text{Pleine production} \\ 25 & \text{si } u_k = 1 & \text{Production essentielle seulement} \\ 100 & \text{si } u_k = 0 & \text{Arrêt de la production} \end{cases} \quad (\text{C.46})$$

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 :

$$J = \lim_{N \rightarrow \infty} \mathbb{E} \left[\sum_{k=0}^N \alpha^k g_k \right] \quad (\text{C.47})$$

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

$$\pi^*(x) = \begin{cases} ? & \text{si } x = 1 \\ ? & \text{si } x = 2 \\ ? & \text{si } x = 3 \\ ? & \text{si } x = 4 \end{cases} \quad (\text{C.48})$$

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

C.6.2 Algorithme d'itération de valeurs

Basé sur la démonstration de base d'une mise en oeuvre de l'algorithme d'itération de valeur au lien ci-dessous :



Exercice de code

Positionnement d'une masse en temps minimal

https://colab.research.google.com/drive/1KQDGznPu_VtTo6Y3ZTkf1Ahr0av2zpAp?usp=sharing

1. Testez l'effet d'augmenter la valeur EPS qui détermine la zone terminale où on considère que la masse est arrivée sur la cible. Comment la solution optimale est affectée ?
2. Observez l'effet du nombre d'itération sur solution (ligne `dp.compute_steps(250)`). En particulier, analysez la solution lorsqu'on effectue seulement 25 itérations, expliquez ce qu'elle représente.
3. Il existe une solution analytique à ce problème, qu'elle est cette politique ? Validez avec l'équation de Bellman. (Bonus)

C.6.3 Évaluation d'une politique



Exercice de code

Évaluation d'une politique de positionnement

<https://colab.research.google.com/drive/1P2sj0o9T31-6rkTBHsK3ZFaGMntvBDzn?usp=sharing>

Le code ci-dessous est une amorce pour évaluer une politique de positionnement de pendule, avec une fonction de coût qui représente le temps pour atteindre la cible.

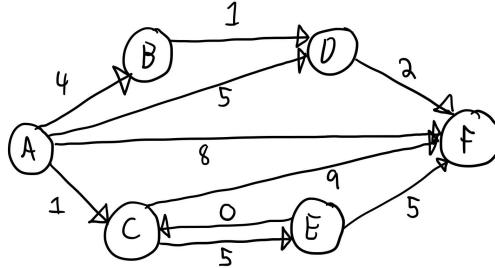
1. Comparez le coût-àvenir de cette politique avec la solution optimale (voir exercice C.6.2)
2. Comparez le coût-àvenir de plusieurs états initiaux, avec le coût d'une trajectoire simulée qui débute sur ces états initiaux.
3. Concevez une politique plus performante et validez numériquement avec l'algorithme.

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) :

$$Q(x, u) \leftarrow g_{\text{mesure}} + \min_{u_{k+1}} [Q(x_{k+1}, u_{k+1})] \quad (\text{C.49})$$

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 :

$$x_{k+1} = 0.5x_k + u_k \quad (\text{C.50})$$

$$g_k = x_k^2 + u_k^2 \quad (\text{C.51})$$

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 :

$$\hat{Q}(x, u) \approx w_1 x^2 + w_2 u^2 + w_3 x u \quad (\text{C.52})$$

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

$$w_i^{new} = w_i^{old} + \eta \left[g_k + \min_{u_{k+1}} \hat{Q}(x_{k+1}, u_{k+1}) - \hat{Q}(x_k, u_k) \right] \frac{\partial \hat{Q}}{\partial w_i} \quad (\text{C.53})$$

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.

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.

C.7.3 *Q-Learning* à partir de l'algorithme DP

Basé sur l'opération de programmation dynamique, qui donne la relation entre le coût-àvenir d'une étape J_k et la suivante J_{k+1} :

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^* \underbrace{(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (\text{C.54})$$

1) Décrivez cette même relation en fonction des valeurs Q_k et Q_{k+1} , qui sont définies ainsi :

$$Q_k^*(x_k, u_k) = \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + Q_{k+1}^* \underbrace{(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (\text{C.55})$$

C.7.4 *Q-Learning* avec échantillonnage

Supposons qu'on obtient des échantillons de valeurs Q basé sur des données de trajectoires :

$$q_i(x_k, u_k, x_{k+1}, u_{k+1}) = g_k(x_k, u_k) + \min_{u_k} Q_{k+1}^*(x_{k+1}, u_{k+1}) \quad (\text{C.56})$$

ou q_i est une variable aléatoire et on cherche à évaluer son espérance :

$$Q^* = \mathbb{E}[q_i] = \frac{1}{N} \sum_{i=1}^N q_i \quad (\text{C.57})$$

On peut utiliser l'algorithme de descente du gradient stochastique, qui a la forme générique suivante :

$$\text{Loss function : } \mathcal{L} = \mathbb{E}[y - \hat{y}(\theta)] = \frac{1}{N} \left[\sum_i \underbrace{y_i - \hat{y}_i(\theta)}_{e_i} \right] \quad (\text{C.58})$$

$$\text{Learning law : } \theta \Leftarrow \theta - \eta \frac{\partial e_i}{\partial \theta} \quad (\text{C.59})$$

pour estimer la valeur Q_k^* . Dans le contexte, l'algorithme du gradient stochastique cherche à minimiser la fonction perte suivante :

$$\text{Loss function : } \mathcal{L} = \mathbb{E}[q_i - \hat{Q}] = \frac{1}{N} \sum_i q_i - \hat{Q} \quad (\text{C.60})$$

Si on suppose que le paramètre estimé est directement la valeur approximée de \hat{Q} :

$$\hat{Q} = \theta \quad (\text{C.61})$$

Déterminez la loi d'apprentissage.

C.8 Apprentissage par renforcement appliqu 

C.8.1 Hands-on PPO

At the following link, you will find a baseline code with a reinforcement learning algorithme able to learn a swing-up policy for an inverted pendulum. However, the way the objective is defined the solution is basically a bang-bang policy. Try to modify the objective function and the algorithm parameter to learn a swing-up policy that use less energy by swinging back-and-worth.



Exercice de code

PPO for a pendulum swing-up

https://colab.research.google.com/drive/1umk613li2ts_Ny9icRL-SjFTHq-a5mAJ?usp=sharing

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.