Commande optimale et apprentissage par renforcement

Une approche unifiée pour les problèmes de prises de décisions séquentielles

préparé par

Pr. Alexandre Girard

Université de Sherbrooke

Dernière mise à jour le 10 novembre 2025

Préface

Ces notes présentent les diverses approches pour prendre des décisions intelligentes sous un cadre théorique unifié dont le pilier central est le **programmation dynamique**. L'objectif est de construire un pont entre deux mondes qui partagent les mêmes fondations mathématiques : le domaine de l'**ingénierie** (la science des asservissements et la commande optimale), où l'on dispose souvent d'un modèle mathématique précis du système, et le domaine des **sciences informatiques** (recherche opérationnelle et apprentissage par renforcement). Ces notes visent principalement à donner à un lecteur issu du domaine de l'ingénierie les bases pour comprendre et utiliser ces puissantes approches numériques.

Capsule vidéo
Série de capsules vidéos associés
`https://youtube.com/playlist?list=PL6adNeJ0A8UtNslNQfAHAzcjHcQixBSnu`

Sources externes utiles

- **Livre Fondamental - Commande Optimale :** *Dynamic Programming and Optimal Control* [Bertsekas, 2017]
- **Livre Fondamental - Apprentissage par Renforcement :** *Reinforcement Learning : An Introduction* [Sutton and Barto, 2018]
- **Notes de Cours Appliquées :** *Underactuated Robotics* [Tedrake, 2023]
- **Vidéos d'Introduction au RL :** Cours de David Silver [Silver, 2015]

Table des matières

Chapitre 1

Introduction

L'apprentissage par renforcement est un domaine qui développe des algorithmes capables d'apprendre automatiquement des stratégies, appelées politiques, qui déterminent pour un agent les actions à choisir en fonction de ses observations. Le domaine de la commande optimale propose également des méthodes pour résoudre ce même problème, mais dans un contexte plus structuré où un modèle de l'environnement, comme les équations du mouvement, est généralement connu. De façon générale, l'apprentissage par renforcement offre des outils très génériques qui apprennent par l'expérience, mais avec peu de garanties formelles. À l'inverse, la commande optimale fournit des outils assortis de garanties solides, mais souvent pour des situations plus spécifiques et en s'appuyant sur des approximations du modèle. Leur fondement commun est la science de la programmation dynamique, qui offre un cadre théorique très général pour traiter tout problème de prise de décision en séquence. Ce principe est si fondamental qu'il s'applique aussi bien à l'analyse d'un système asservi classique, comme le contrôle d'un bras robot ; qu'à des problèmes probabilistes en finance, comme la décision d'acheter ou de vendre une action ; ou encore à un problème d'intelligence artificielle, comme le choix de la pièce à déplacer aux échecs.

Capsule vidéo
Introduction
`https://youtu.be/1ThWOUnkVyY?si=wWIaO-OYpvR-vYbL`

1.1 La prise de décision en séquence

Au cœur de tout problème de prise de décision en séquence se trouve une interaction fondamentale entre un **agent** et son **environnement**. L'objectif de l'agent est d'influencer l'évolution de cet environnement, que l'on considère comme un **système dynamique**. Pour ce faire, l'agent choisit continuellement des **actions** en se basant sur les **observations** qu'il reçoit de l'environnement. Cette interaction constante — observer, décider, agir — crée un **système dynamique en boucle fermée** qui évolue dans le temps, tel qu'illustré à la figure 1.1.

1.1.1 Un comportement défini par une politique à concevoir

L'objectif principal est de concevoir une fonction, notée π et appelée la **politique**, qui définit le comportement de l'agent. La politique agit comme une règle de décision ou une "carte" qui détermine l'action à entreprendre en fonction de l'état observé de l'environnement :

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

où u représente l'action choisie et x représente l'état de l'environnement que l'agent observe. Dans le domaine de l'asservissement, le terme équivalent pour la fonction π est la *loi de commande*. La forme concrète de cette politique est très flexible et peut aller d'une simple équation analytique ou d'un tableau de données (*look-up table*) à un réseau de neurones complexe.

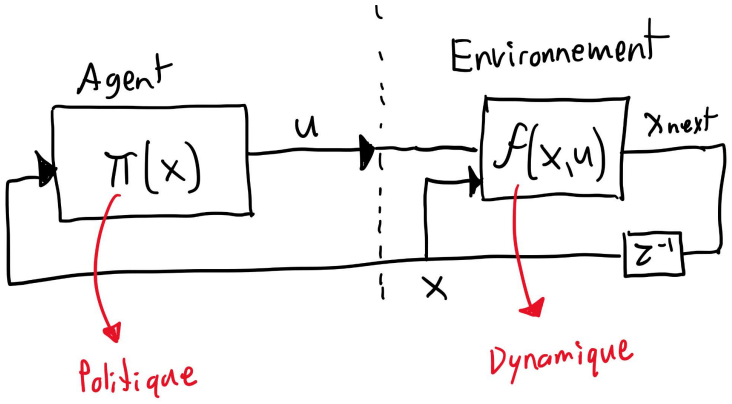


FIGURE 1.1 – Dynamique en boucle fermée avec un agent

1.1.2 Un objectif formulé avec une fonction scalaire cumulative

Pour définir le comportement désiré de l'agent, son objectif est formulé mathématiquement par une fonction additive qui cumule une valeur sur l'ensemble d'une trajectoire. Typiquement, en **commande optimale**, on cherche à minimiser une fonction de **coût**, tandis qu'en **apprentissage par renforcement**, on vise à maximiser une fonction de **récompense**. Ces deux formulations sont équivalentes — maximiser une récompense revient à minimiser son opposé — et nous utiliserons par défaut la minimisation de coût dans ces notes. Cette fonction objectif, le coût cumulatif, est notée J et se définit comme la somme des coûts instantanés $g(x, u)$ à chaque étape :

$$\underbrace{J}_{\text{Coût cumulatif}} = \sum \underbrace{g(x,u)}_{\text{Coût instantané}} \tag{1.2}$$

où $g(x, u)$ représente le coût ponctuel pour se trouver dans l'état x et avoir choisi l'action u. La forme cumulative de cette fonction est un prérequis essentiel pour appliquer le principe de la programmation dynamique (toutefois ce n'est pas restrictif les problèmes peuvent tous être reformuler sous cette forme, typiquement avec des états augmentés si nécessaire). Par conséquent, une politique est dite **optimale** lorsqu'elle guide l'agent à prendre la séquence de décisions qui minimise la valeur de ce coût cumulatif J.

Exemple 1. Loi de commande pour un robot

Considérons un exemple d'asservissement classique : le contrôle d'un bras robotique. On peut décomposer ce problème en plusieurs éléments clés :

- ***État*** *(x) : Un vecteur représentant l'état actuel du robot, qui inclut typiquement les positions et les vitesses de ses diverses articulations.*
- ***Action*** *(u) : Le vecteur de couples (forces de rotation) que la politique détermine qu'il faut appliquer aux moteurs électriques de chaque articulation.*
- ***Objectif*** *(J) : L'objectif est formulé comme la minimisation d'une fonction de coût. Ce coût combine généralement deux termes : l'erreur de position du robot par rapport à une cible, et une pénalité pour l'énergie consommée (l'amplitude des couples).*
- ***Politique*** *(π) : La politique $u = \pi(x)$, ou loi de commande, est la fonction qui calcule l'action à partir de l'état. Pour ce type de problème, la solution est souvent une équation analytique.*

Exemple 2. Navigation d'un véhicule

Un exemple de prise de décision à un plus haut niveau est le choix d'un trajet sur une carte routière. Les composantes de ce problème peuvent être définies comme suit :

- ***État** (x) : La position actuelle du véhicule sur la carte, par exemple à une intersection donnée.*
- ***Action** (u) : La direction à prendre à partir de la position actuelle (ex : "tourner à gauche", "continuer tout droit").*
- ***Objectif** (J) : Atteindre la destination le plus rapidement possible. Cet objectif est formulé comme la minimisation du temps de trajet total.*
- ***Politique** (π) : La politique est la solution globale qui, pour chaque intersection possible sur la carte (chaque état), spécifie la direction optimale à prendre. Elle peut être stockée sous la forme d'une table de correspondance (look-up table).*

Exemple 3. Achats d'une action

Dans un tout autre contexte, un algorithme d'investissement peut aussi être vu comme un agent prenant des décisions séquentielles :

- ***État** (x) : L'observation principale est le prix actuel de l'action. Un état plus complexe pourrait aussi inclure l'historique des prix ou d'autres indicateurs de marché.*
- ***Action** (u) : La décision à prendre à un instant donné. Dans ce cas simple, ce serait "Acheter" ou "Ne pas acheter".*
- ***Objectif** (J) : L'objectif est la maximisation des gains financiers. C'est un exemple typique de problème formulé en termes de maximisation de récompense.*
- ***Politique** (π) : La règle de décision. Elle pourrait être un simple seuil de prix qui, potentiellement, varie dans le temps : si le prix observé passe en dessous de ce seuil, l'agent décide d'acheter.*

1.2 Tour d'horizon

Une politique optimale doit satisfaire une relation fondamentale appelée l'**équation de Bellman**, qui peut prendre plusieurs formes selon la formulation de l'objectif et la méthode utilisée pour décrire l'évolution du système. Cette équation est la condition qui garantit l'optimalité d'une prise de décision par rapport à la fonction de coût choisie. D'un certain point de vue, toutes les méthodes de commande et d'apprentissage par renforcement ont donc pour objectif commun de trouver une solution, même approximative, à cette équation centrale.

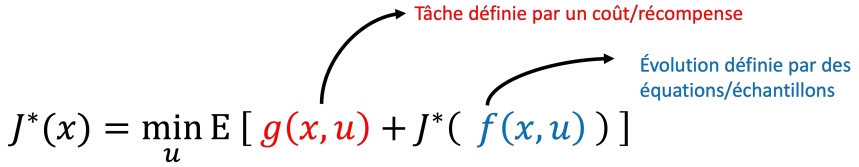


FIGURE 1.2 – Équation de Bellman, qui caractérise si une politique est optimale

L'équation de Bellman est un principe flexible qui s'adapte à de nombreux contextes. Par exemple, lorsqu'on modélise l'évolution d'un système dans un **domaine continu**, comme c'est le cas en asservissement, elle peut se réduire sous certaines hypothèses à la célèbre solution linéaire quadratique (LQR). À l'inverse, l'intelligence artificielle d'un jeu comme les échecs, qui évolue en **états et actions discrets**, se base aussi sur cette équation ; en considérant les actions de l'adversaire comme une perturbation, le principe mène directement à l'**algorithme minimax**. Finalement, en apprentissage par renforcement, où le modèle d'évolution est souvent **stochastique** et inconnu, on cherche à approximer une solution à l'équation de Bellman à partir

d'échantillons obtenus par observation. Une bonne partie de ces notes a donc pour objectif de présenter cette équation dans ces divers contextes, et d'explorer les algorithmes permettant de la résoudre.

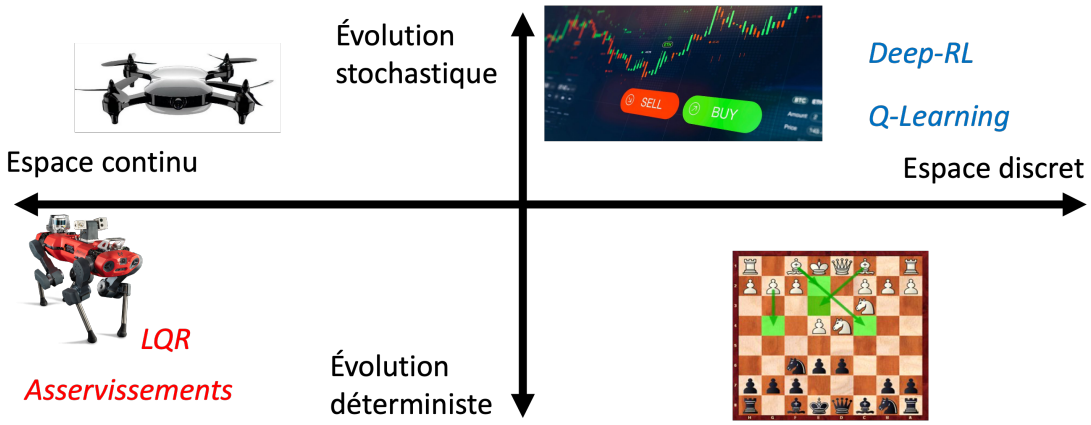


FIGURE 1.3 – Différentes contextes de prise de décision en séquence

1.2.1 Le problème canonique de commande optimale

Dans un problème typique de commande optimale, la politique est synthétisée entièrement *a priori*, c'est-à-dire avant que l'agent n'interagisse avec l'environnement. Comme l'illustre la figure 1.4, cette phase de synthèse (ou de planification) requiert la connaissance préalable de deux éléments : un **modèle de la dynamique** $\hat{f}$ du système et la **fonction de coût** $\hat{g}$ qui définit l'objectif. Dans la littérature sur l'apprentissage par renforcement, cette approche est parfois désignée par *model-based reinforcement learning* (apprentissage par renforcement basé sur un modèle).

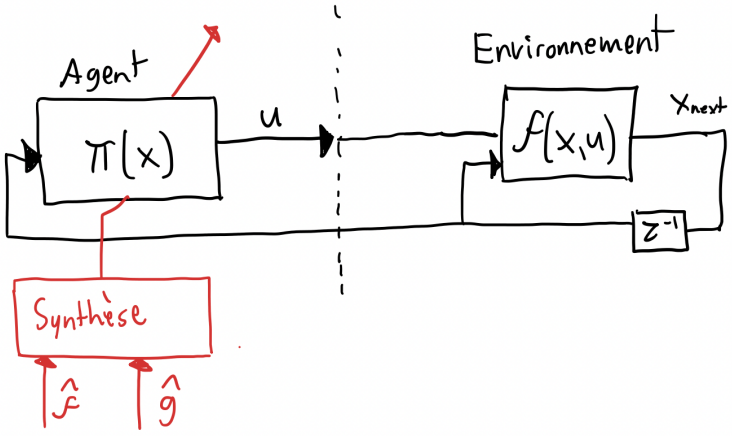


FIGURE 1.4 – Aperçu du problème de commande optimale

1.2.2 Le problème canonique d'apprentissage par renforcement

À l'opposé de la commande optimale, un problème d'apprentissage par renforcement suppose que la dynamique de l'environnement et la fonction de coût sont initialement **inconnues**. La politique ne peut donc pas être calculée *a priori*. Au lieu de cela, l'agent doit apprendre une politique optimale en interagissant directement avec l'environnement par un processus d'essai et erreur, comme le montre la figure 1.5. La politique est ainsi mise à jour progressivement à mesure que l'agent accumule de l'expérience. Dans le domaine de

la commande, une approche similaire où des données sont utilisées en ligne pour ajuster une politique ou identifier les paramètres d'un modèle est souvent appelée *commande adaptative* ou *identification de système*.

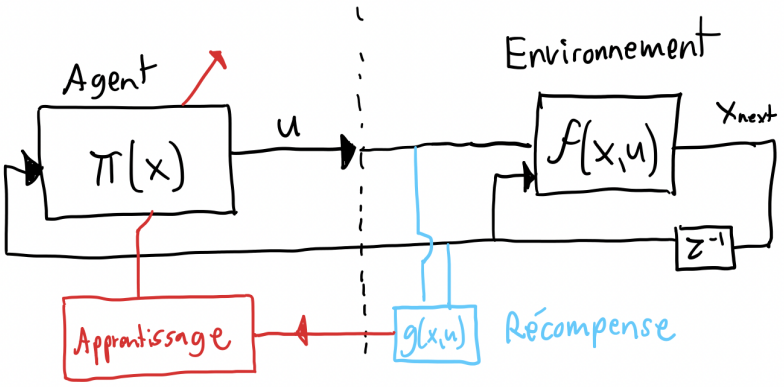


FIGURE 1.5 – Aperçu du problème d'apprentissage par renforcement

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

Exploration vs. exploitation

Puisque l'agent en apprentissage par renforcement ne connaît pas l'environnement *a priori*, il fait face à un dilemme fondamental : le compromis entre l'**exploitation** et l'**exploration**. L'exploitation consiste à appliquer la meilleure stratégie découverte jusqu'à présent pour maximiser les gains immédiats. L'exploration, à l'inverse, consiste à tenter de nouvelles actions — au risque d'obtenir un moins bon résultat à court terme — dans le but de découvrir une stratégie potentiellement encore plus performante pour l'avenir. La gestion de cet arbitrage est un des défis centraux de l'apprentissage par renforcement.

La place de l'apprentissage par renforcement dans l'apprentissage machine

Bien que l'apprentissage par renforcement soit une branche de l'apprentissage machine, il se distingue fondamentalement des branches classiques. soit l'apprentissage supervisées ou non supervisées. L'apprentissage **supervisé** requiert un expert pour fournir des "bonnes réponses", comme des paires état-action correctes à imiter dans le contexte d'une politique (une technique nommée *Behavioral Cloning*). L'apprentissage **non supervisé**, quant à lui, cherche à découvrir la structure cachée dans des données non étiquetées (ex : clustering) sans notion d'objectif ou d'actions. L'apprentissage par renforcement est tout autre : l'agent apprend par **essai et erreur** en interagissant avec un environnement, guidé uniquement par un **signal de performance scalaire** (un coût ou une récompense). Cette interaction introduit une complexité majeure qui le distingue des autres formes d'optimisation : la gestion des **conséquences à long terme**. Une action peut avoir des répercussions qui ne se manifestent que bien plus tard, et le cœur du sujet est de développer des méthodes pour attribuer correctement un résultat futur à une décision passée.

1.3 Exemples de mise en œuvre

La meilleure façon de solidifier la théorie est de la mettre en pratique. Les page *colab* suivantes sont conçus pour une exploration interactive. Il est recommandé de les parcourir, d'analyser le code pour en comprendre le fonctionnement, et de tenter d'y faire des modifications pour valider votre compréhension.

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-EuRPiyf2GvOn8p_17jlvvEIE188WT1m?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`

1.4 Résumé du Chapitre

Le chapitre a introduit le cadre général des problèmes de prise de décision séquentielle et présenté les deux principales approches pour les résoudre : la commande optimale et l'apprentissage par renforcement. Il été discuté que malgré leurs origines différentes, ces deux approches partagent un fondement théorique commun basé sur la programmation dynamique et l'équation de Bellman. Les termes principaux sont résumé au tableau 1.4.

Terme	Définition
Agent	L'entité qui perçoit son environnement et prend des décisions (ex : un robot, un algorithme d'investissement).
Environnement	Le "monde" extérieur avec lequel l'agent interagit et qu'il cherche à influencer ; il est souvent modélisé comme un système dynamique.
Action	Une décision ou une commande que l'agent applique à l'environnement.
Observation	L'information que l'agent reçoit de l'environnement à un instant donné pour prendre sa décision.
Politique	La stratégie, ou fonction, qui détermine quelle action (u) l'agent choisit dans un état donné (x). C'est la solution que l'on cherche à concevoir.
Loi de commande	Le terme équivalent à "politique" dans le domaine de l'ingénierie et de l'asservissement.
Coût / Récompense	Un signal numérique qui évalue la performance (coût ou récompense) immédiate d'une action lorsque l'environnement est dans un état donné.
Coût cumulatif	La somme des coûts (ou récompenses) sur une trajectoire complète. C'est la fonction objectif globale que l'on cherche à optimiser.
Politique optimale	La politique qui minimise le coût cumulatif (ou maximise la récompense cumulative) sur le long terme.
Commande optimale	Une famille d'approches où la politique est calculée *a priori* (hors ligne) en s'appuyant sur un modèle mathématique connu de l'environnement et de la fonction de coût.
Apprentissage par renforcement	Une famille d'approches où la politique est apprise par l'agent via un processus d'essai et erreur, en interagissant directement avec un environnement généralement inconnu.
Équation de Bellman	La relation mathématique fondamentale qui caractérise une politique optimale.
Exploration vs. Exploitation	Le dilemme central en apprentissage par renforcement : faut-il utiliser la meilleure stratégie connue (exploitation) ou en essayer de nouvelles pour potentiellement trouver mieux (exploration) ?
Conséquences à long terme	La difficulté majeure des problèmes de commande optimale ou d'apprentissage par renforcement, où l'impact d'une action peut n'être visible que bien plus tard.

Chapitre 2

Programmation dynamique

Ce chapitre introduit les fondements mathématiques de la science de la prise de décision séquentielle. Pour établir les principes de base, nous utiliserons une formulation qui repose sur trois hypothèses simplificatrices : nous travaillerons en **temps discret**, nous chercherons à optimiser une politique sur un **horizon fini** de N étapes, et nous supposerons que l'agent **observe directement** l'état complet 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 détour, car 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. De plus, avec cette formulation, les liens avec les concepts de la science de l'asservissement sont plus directs à faire. On traitera plus tard les situations où l'horizon de temps est infini (Chapitre 5), où l'évolution est stochastique (Chapitre 3), et où 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 décision en séquence :

2.1.1 Dynamique

La première composante est la **dynamique** du système, notée f_k, qui décrit la manière dont l'environnement évolue. Elle est représentée par une équation de différence qui lie l'état à une étape à l'état de l'étape suivante :

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

Dans cette équation, x_k est l'état du système à l'étape k et u_k est l'action choisie par l'agent à cette même étape. La fonction de dynamique f_k est donc le *modèle* de l'environnement. La connaissance de cette fonction est une distinction clé entre les différentes approches : certains algorithmes l'utilisent directement, tandis que d'autres supposent qu'elle est inconnue.

2.1.2 Politique (Loi de commande)

La deuxième composante est la **politique** (ou *loi de commande*), notée π. Elle représente la stratégie de l'agent : la fonction qui dicte quelle action u prendre lorsque l'état x est observé. Dans le contexte d'un problème à horizon fini, la décision optimale peut dépendre de l'étape k en cours. On peut donc définir une politique spécifique pour chaque étape :

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

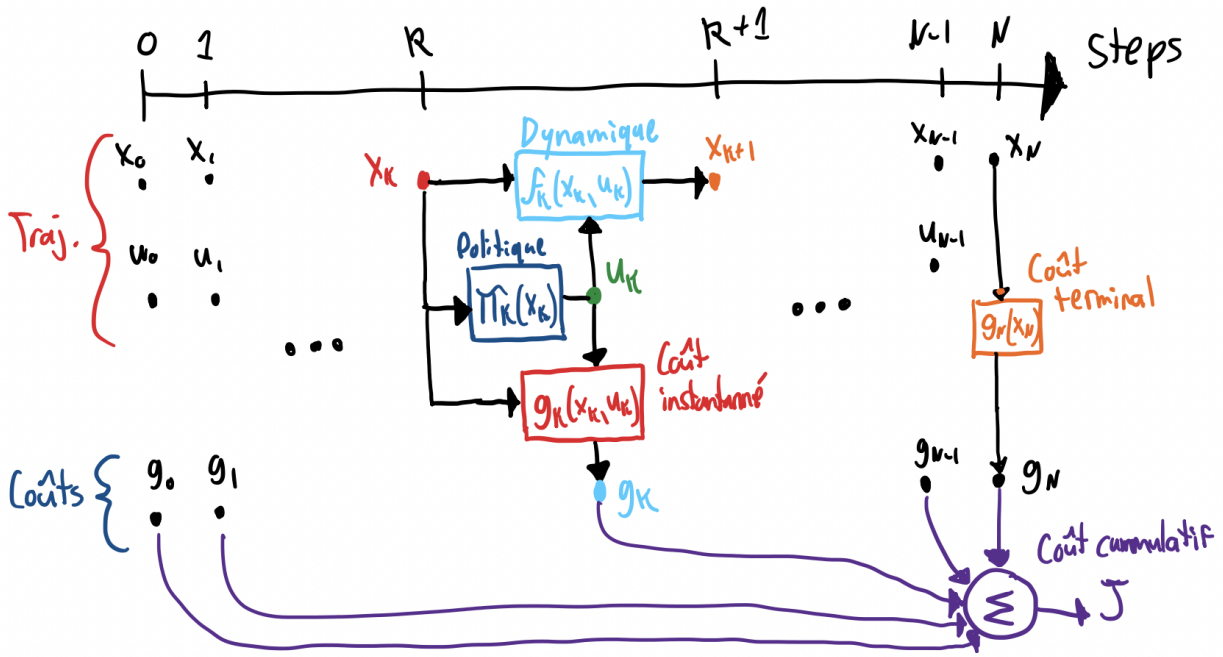


FIGURE 2.1 – Évolution du système en boucle fermée étape par étape.

Une politique qui change en fonction de l'étape est dite *non stationnaire* (ou dépendante du temps). La stratégie complète sur tout l'horizon, notée π sans indice, est alors l'ensemble de ces politiques spécifiques à chaque étape :

$$\pi = \{\pi_0, \dots, \pi_{N-1}\} \tag{2.3}$$

2.1.3 Fonction objectif (coût ou récompense)

La troisième composante est la **fonction objectif**, notée J. Son rôle est d'assigner une valeur numérique à une trajectoire complète afin d'évaluer sa performance par rapport à l'objectif visé. Elle est définie comme la somme des coûts encourus à chaque étape, plus un coût terminal :

$$J(\underbrace{x_0, \dots, x_N, u_0, \dots, u_{N-1}}_{\text{Trajectoire}}) = \underbrace{\sum_{k=0}^{N-1} g_k(x_k, u_k)}_{\text{Somme des coûts instantanés}} + \underbrace{g_N(x_N)}_{\text{Coût terminal}} \tag{2.4}$$

où $g_k(x_k, u_k)$ est le **coût instantané** à l'étape k, $g_N(x_N)$ est le **coût terminal** qui ne dépend que de l'état final x_N, et N est l'**horizon** (le nombre total d'étapes). La conception de ces fonctions de coût est un art en apprentissage par renforcement : bien que plusieurs formulations puissent être théoriquement équivalentes, leur structure peut grandement faciliter ou compliquer la tâche d'apprentissage de l'algorithme.

2.1.4 Coût-à-venir

Un concept fondamental est le **coût-à-venir** (ou *cost-to-go*), noté $J^\pi(x)$. Contrairement à la fonction objectif J qui évaluait une trajectoire complète, le coût-à-venir est une fonction de **prédiction**. Il représente le coût cumulatif total qu'on va accumuler en partant de l'état x et en suivant la politique π sur un horizon de N étapes :

$$J^\pi(x) = \sum_{k=0}^{N-1} g_k(x_k, \pi_k(x_k)) + g_N(x_N), \quad \text{avec} \quad \begin{cases} x_0 = x \\ x_{k+1} = f_k(x_k, \pi_k(x_k)) \end{cases} \tag{2.5}$$

La distinction clé est que $J^\pi(x)$ est une fonction uniquement de l'état initial x. La politique π et la dynamique f_k déterminent entièrement la trajectoire future, ce qui permet de résumer le coût de cette trajectoire en une seule valeur associée à son point de départ.

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 laquelle est bâti tout le principe de la programmation dynamique. De plus, elle est au cœur de l'apprentissage par renforcement, où une grande catégorie d'algorithmes a pour objectif principal d'apprendre une approximation de cette fonction, souvent à l'aide d'un réseau de neurones.

2.1.5 Contraintes

En plus de la fonction objectif, il est souvent nécessaire d'inclure des **contraintes** pour définir les limites opérationnelles ou les "règles du jeu" du problème. On considère généralement deux types de contraintes à chaque étape k :

$$x_k \in \mathcal{X}_k \qquad \text{et} \qquad u_k \in \mathcal{U}_k(x_k) \tag{2.6}$$

La première contrainte, $x_k \in \mathcal{X}_k$, signifie que l'état doit rester dans un ensemble d'états permis. La seconde, $u_k \in \mathcal{U}_k(x_k)$, signifie que l'action doit être choisie dans un ensemble d'actions possibles, qui peut lui-même dépendre de l'état actuel x_k. Une politique est dite **admissible** si elle respecte ces contraintes à chaque étape. L'ensemble de toutes les politiques admissibles est noté Π :

$$\pi \in \Pi \tag{2.7}$$

2.1.6 Solution optimale

L'objectif final est de trouver la meilleure politique parmi toutes celles qui sont admissibles dans l'ensemble Π. La **politique optimale**, notée π^*, est par définition celle qui minimise la fonction de coût-à-venir pour un état de départ x. La valeur de ce coût minimal est appelée le **coût-à-venir optimal**, noté $J^*(x)$. Mathématiquement, ces concepts sont définis comme suit :

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

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

L'équation (2.8) définit la valeur du meilleur coût possible, tandis que l'équation (2.9) identifie la politique qui permet d'atteindre cette valeur.

2.1.7 Terminologie

Symbole	Termes Associés	Définition
	Variables de Base	
k	Indice de temps, Étape	Entier correspondant à l'étape actuelle dans un processus discrétisé.
x_k	État, *State*	Variable qui représente toute l'information nécessaire pour prédire l'évolution future du système.
u_k	Action, Décision, Entrée de commande, *Control Input*	Variable qui représente la décision prise par l'agent.
	Fonctions Clés	
$f_k(x_k, u_k)$	Dynamique, Système, Environnement, *Plant*	Équation qui définit l'évolution de l'environnement de l'agent.
$g_k(x_k, u_k)$	Coût instantané, Récompense	Fonction scalaire qui définit le coût (ou la récompense) à l'étape k.
$g_N(x_N)$	Coût terminal	Fonction scalaire qui définit le coût final en fonction de l'état terminal.
$J(\dots)$	Coût cumulatif, Fonction objectif	Fonction scalaire qui évalue la qualité d'une trajectoire complète.
$J^\pi(x)$	Coût-à-venir, *Cost-to-go*, *Value Function*	Fonction qui prédit le coût cumulatif d'une trajectoire débutant à x en suivant la politique π.
$J^*(x)$	Coût-à-venir optimal, *Optimal Cost-to-go*	Fonction qui donne le coût cumulatif minimal possible pour une trajectoire débutant à x.
$\pi_k(x_k)$	Politique, Loi de commande, Contrôleur, *Policy*	Fonction qui définit la décision de l'agent comme une fonction de l'état observé.
π^*	Politique optimale	La politique qui minimise le coût cumulatif sur le long terme.
	Ensembles (Sets)	
$\mathcal{X}_k$	Ensemble d'états admissibles	Ensemble des états permis à l'étape k.
$\mathcal{U}_k(x_k)$	Ensemble d'actions admissibles, *Control Set*	Ensemble des actions possibles à l'étape k lorsque le système est dans l'état x_k.
Π	Ensemble des politiques admissibles	L'ensemble de toutes les politiques qui respectent les contraintes d'état et d'action.

Exemple 1. Pendule Simple en temps minimum

Le concept de coût-à-venir 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-à-venir $J^{\pi}(x)$ représenterait le temps-à-venir, i.e. le temps prévu avant d'atteindre la cible, lorsqu'on débute à l'état x. La figure 2.2, représente une temps-à-venir optimal calculé numériquement.

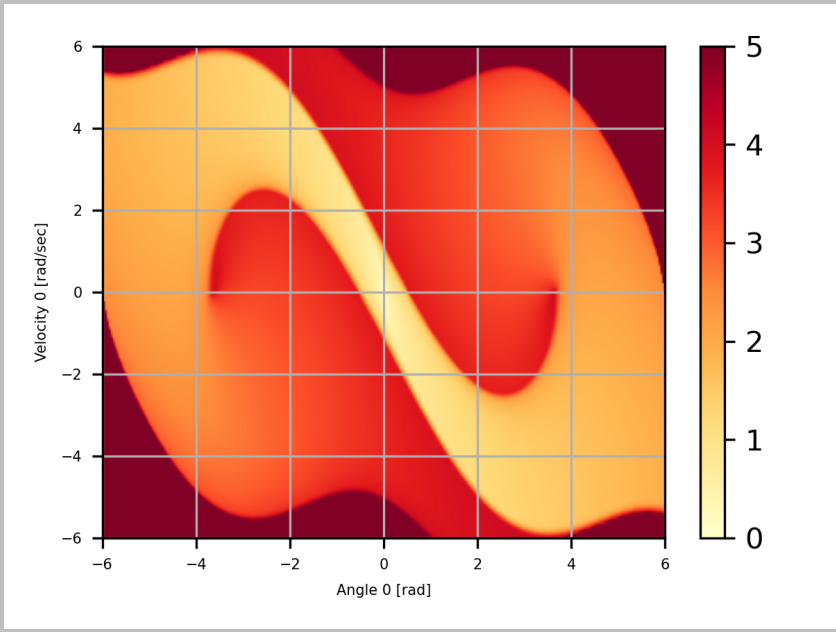


FIGURE 2.2 – Coût-à-venir optimal pour un pendule (temps-minimum)

2.2 Principe d'optimalité

Le **principe d'optimalité**, est la pierre angulaire de la programmation dynamique. Il formalise une notion intuitive mais puissante :

> *Si une trajectoire complète de x_0 à x_N est optimale et passe par un état intermédiaire x_i, alors la sous-trajectoire de x_i jusqu'à x_N est nécessairement elle-même la trajectoire optimale pour un problème qui débuterait à x_i.*

$$[x_0, ..., x_i, ..., x_N] \tag{2.10}$$
$$[x_i, ..., x_N] \tag{2.11}$$

Capsule vidéo
Le principe d'optimalité
`https://youtu.be/EMkpkyMTg4U?si=eDITgpq5ONhRD8-f`

Par exemple, supposons que le chemin optimal en voiture entre Montréal et Québec passe par Drummondville. Le principe d'optimalité affirme alors que le segment de ce trajet allant de Drummondville à Québec est *nécessairement* lui-même le chemin optimal entre ces deux villes. L'idée fondamentale de la programmation dynamique est d'exploiter cette propriété : en résolvant les sous-problèmes (comme le trajet optimal de Drummondville à Québec), on peut construire de manière récursive la solution au problème global.

2.3 Programmation dynamique exacte

L'algorithme de programmation dynamique est la mise en œuvre directe du principe d'optimalité et permet de résoudre exactement le problème de décision formulé à la section 2.1. L'idée fondamentale est de calculer le coût-à-venir optimal en partant de la fin et en remontant le temps, étape par étape. Le processus commence par une **initialisation** à l'étape finale N, où le coût-à-venir optimal est simplement le coût terminal. Ensuite, on effectue une **récursion vers l'arrière** pour chaque étape k, de $N-1$ jusqu'à 0, en utilisant la valeur J^*_{k+1} calculée à l'itération précédente :

$$J^*_N(x_N) = g_N(x_N) \qquad \forall x_N \in \mathcal{X}_N \tag{2.12}$$

$$J^*_k(x_k) = \min_{u_k \in \mathcal{U}_k(x_k)} \left[g_k(x_k, u_k) + J^*_{k+1}(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \forall x_k \in \mathcal{X}_k \quad \text{for } k = N-1, \dots, 0 \tag{2.13}$$

En parallèle du calcul du coût-à-venir, l'opération de minimisation permet d'identifier l'action optimale à chaque étape. La **politique optimale** $\pi^*_k(x_k)$ est donc obtenue en conservant l'action u_k qui minimise la somme du coût instantané et du coût-à-venir futur, ce qui mathématiquement est représenté par l'opérateur $\arg\min$:

$$\pi^*_k(x_k) = \operatorname*{argmin}_{u_k \in \mathcal{U}_k(x_k)} \left[g_k(x_k, u_k) + J^*_{k+1}(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \qquad \forall x_k \in \mathcal{X}_k \tag{2.14}$$

Si l'on décortique cette récursion, l'expression à l'intérieur de l'opérateur min est d'une importance capitale. Notée $Q^*_k(x_k, u_k)$, elle est souvent appelée la **valeur-Q** (*Q-value*) et est un concept central en apprentissage par renforcement. Elle représente le coût-à-venir si, partant de l'état x_k, on choisit une action spécifique u_k et que l'on suit la politique optimale par la suite. L'algorithme de programmation dynamique se résume alors

à trouver, pour un état x_k, l'action u_k qui minimise cette valeur-Q :

$$\underbrace{J_k^*(x_k)}_{\text{Coût-à-venir optimal}} = \underbrace{\min_{u_k \in \mathcal{U}_k(x_k)}}_{\text{Minimum possible}} \underbrace{\left[\underbrace{g_k(x_k, u_k)}_{\text{Coût instantané}} + \underbrace{J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}})}_{\text{Coût-à-venir optimal à l'étape suivante}} \right]}_{\text{Valeur } Q_k^*(x_k, u_k)} \tag{2.15}$$

La politique optimale consiste donc à choisir l'action qui minimise $Q_k^*(x_k, u_k)$, et le coût-à-venir optimal $J_k^*(x_k)$ est la valeur de ce Q^* minimal. Un algorithme fondamental en apprentissage par renforcement, le *Q-Learning*, est entièrement basé sur l'apprentissage de ces valeurs-Q.

Capsule vidéo
Algorithme de programmation dynamique
`https://youtu.be/dfz9k3BGrH0?si=MBxJzpKP0UjnaERe`

J vs Q La distinction entre ces deux fonctions est une question de perspective. La valeur $J^*(x)$ représente le coût-à-venir optimal d'un état x, *avant* le choix de l'action, en supposant que la meilleure action sera prise. Inversement, la valeur $Q^*(x, u)$ représente le coût-à-venir optimal une fois qu'une action spécifique u *a déjà été sélectionnée* dans l'état x (et en supposant que toutes les actions futures seront optimales). Le lien entre les deux est donc une opération de minimisation : la valeur optimale d'un état est la valeur-Q minimale parmi toutes les actions possibles dans cet état, soit $J^*(x) = \min_u Q^*(x, u)$.

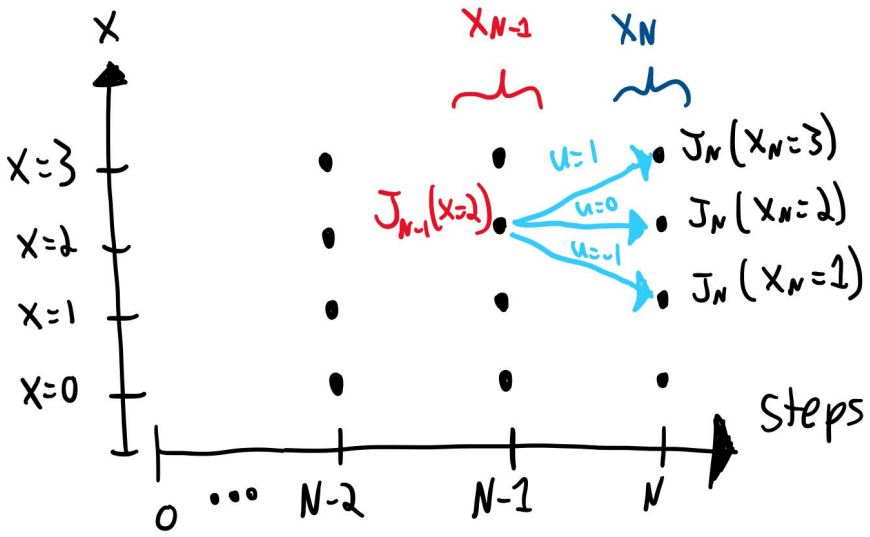


FIGURE 2.3 – Exemple de calcul de coût-à-venir pour l'état $x = 2$ à l'étape $k = N - 1$. Ici pour cet exemple on a 4 états discrets possibles et trois actions possibles. Comme illustré, on doit d'abord avoir évalué le coût terminal de tous les états à l'étape $k = N$, ensuite on calcul de coût-à-venir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

Exemple 2. Navigation optimale

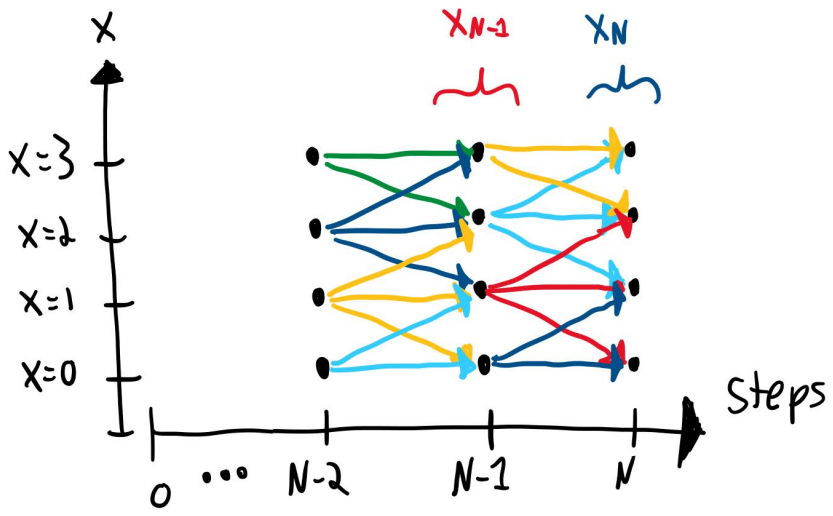


FIGURE 2.4 – L'algorithme de programmation dynamique consiste à exécuter ce calcul pour chaque état à chaque étape en débutant par la fin. Ici chaque couleur représente une étape de minimisation pour un état. Il y a une valeur $Q(x, u)$ pour chaque état-action qui sont ici les arcs, et une valeurs $J(x)$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

Capsule vidéo
Exemple de navigation optimale sur un graphe
https://youtu.be/1GXUNWVgZOU?si=P4hDvWSWoav6nW4x

Exemple 3. Chauffage optimale

Capsule vidéo
Exemple pour une politique de chauffage optimale
https://youtu.be/QwXjiAzDENs?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'équation de Bellman est un principe flexible qui s'adapte à de nombreux contextes. Voici un aperçu des principales variations que nous rencontrerons.

Stochastique

Lorsque l'environnement est stochastique, l'évolution dépend d'une perturbation aléatoire w_k. Pour en tenir compte, on optimise le coût *en espérance*, c'est-à-dire en moyenne sur toutes les issues possibles de w_k.

$$J_k^*(x_k) = \min_{u_k} \underset{w_k}{\mathbb{E}} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.16}$$

Robuste (Minimax)

Dans un contexte adverse (jeux, pires scénarios), on ne cherche pas à optimiser la moyenne, mais à se prémunir contre le pire cas possible. L'espérance est alors remplacée par un opérateur de maximisation, créant une formulation *minimax*.

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.17}$$

À horizon de temps infini

Pour les problèmes sans fin déterminée, les indices de temps k disparaissent (la politique devient stationnaire) et on introduit un **facteur d'escompte** $\alpha < 1$. Il garantit la convergence du coût et donne plus de poids au futur proche.

$$J^*(x) = \min_u \left[g(x, u) + \alpha J^*(\underbrace{f(x, u)}_{x_{k+1}}) \right] \tag{2.18}$$

Formulation en Valeurs-Q (sans modèles)

Lorsque le modèle de la dynamique f est inconnu, il est impossible d'évaluer J^* directement car on ne peut pas prédire x_{k+1}. On travaille alors avec les **valeurs-Q**, $Q^*(x, u)$, qui représentent le coût-à-venir après avoir choisi l'action u.

$$Q^*(x, u) = g(x, u) + \min_{u_{k+1}} [Q^*(x_{k+1}, u_{k+1})] \tag{2.19}$$

À temps continu

Pour les systèmes dont le temps est continu, l'équation de Bellman devient une équation différentielle partielle appelée équation de **Hamilton-Jacobi-Bellman (HJB)**.

$$-\frac{\partial J^*(x,t)}{\partial t} = \min_u \left[g(x, u) + \frac{\partial J^*(x,t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \tag{2.20}$$

$$-\frac{\partial J^*(x,t)}{\partial t} = \min_u \left[g(x,u) + \frac{\partial J^*(x,t)}{\partial x} \underbrace{f(x,u,t)}_{\dot{x}} \right] \tag{2.21}$$

$$J^*(x) = \min_{u \in \mathcal{U}} \mathop{\mathbb{E}}_{w \in \mathcal{W}} [g(x,u,w) + J^*(f(x,u,w))] \tag{2.22}$$

$$J_k^*(x_k) = \min_{u_k \in \mathcal{U}_k} \mathop{\mathbb{E}}_{w_k \in \mathcal{W}_k} \left[g_k(x_k,u_k,w_k) + J_{k+1}^*(\underbrace{f_k(x_k,u_k,w_k)}_{x_{k+1}}) \right] \tag{2.23}$$

2.5 Forces et limites de la programmation dynamique

Forces

— **Flexibilité de la formulation :** Le formalisme peut intégrer des dynamiques non-linéaires, des coûts non-convexes, des contraintes complexes, ainsi que des variables discrètes ou stochastiques.

— **Optimalité globale :** Contrairement à de nombreuses méthodes d'optimisation, la programmation dynamique garantit de trouver la solution globalement optimale, et non un simple optimum local.

Limites

— **Malédiction de la dimensionnalité ("Curse of Dimensionality") :** La complexité de l'algorithme exact croît de manière exponentielle avec le nombre de variables d'état. En pratique, il n'est applicable que pour des problèmes de faible dimension.

— **Nécessité d'un modèle :** L'approche exacte requiert un modèle mathématique de la dynamique du système (f_k) et de la fonction de coût (g_k).

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 un certaine forme de stratégie pour approximer un algorithme de de programmation dynamique.

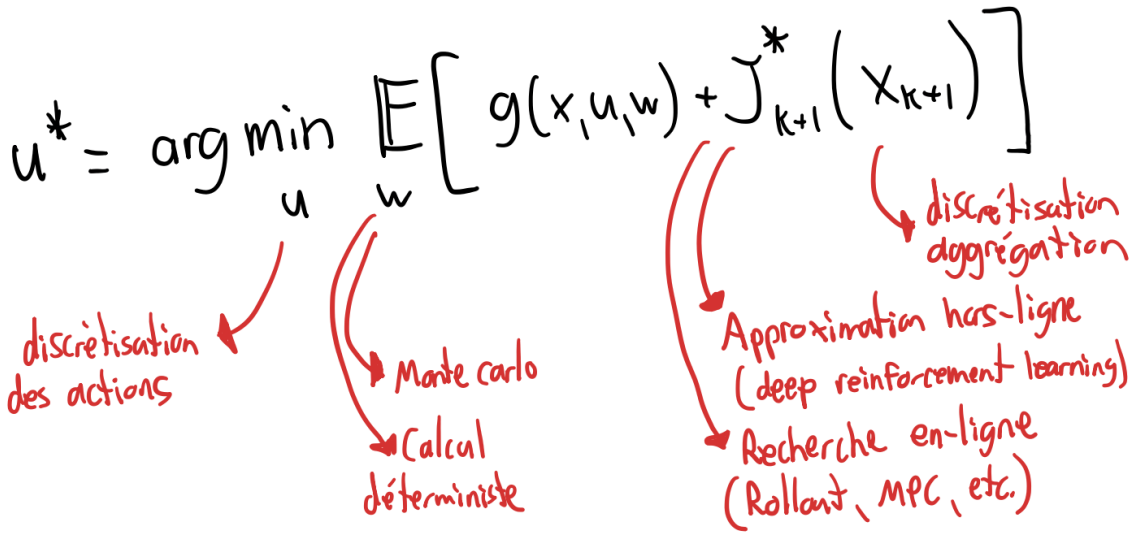


FIGURE 2.5 – Différentes stratégies pour approximer la programmation dynamique exacte.

Capsule vidéo
Méthodes approximatives
`https://youtu.be/jWP9yiIL7gY?si=jLFLnHSRSpXuxS1I`

Chapitre 3

Commande stochastique

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

3.1 Dynamique stochastique

Pour représenter une évolution stochastique, la formulation du problème est modifiée en ajoutant une **variable aléatoire** w_k à la fonction de dynamique. Cette variable, que l'on peut interpréter comme une perturbation ou un bruit, rend l'état suivant x_{k+1} incertain même si l'état x_k et l'action u_k sont connus. L'équation de dynamique devient :

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad \text{où} \quad w_k \in \mathcal{W}_k(x_k) \tag{3.1}$$

ou la variable w_k appartient à un ensemble noté $\mathcal{W}_k(x)$. Un modèle dynamique stochastique est donc plus riche qu'un modèle déterministe. En plus de la fonction f_k, il doit inclure la **loi de probabilité** qui régit la variable w_k. Cette loi peut être conditionnelle à l'état et à l'action. Si w_k prend des valeurs **discrètes**, le modèle est une fonction de masse de probabilité, notée $P(w_k|x_k, u_k)$. Si w_k est une variable **continue**, le modèle est décrit par une fonction de densité de probabilité, notée $p(w_k|x_k, u_k)$.

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=cqnH8BDjGGBLmxi5`

Ce modèle d'évolution stochastique est une manière de construire ce que l'on appelle une *Chaîne de Markov*. En effet, au lieu de le décrire par une fonction de dynamique f_k et une perturbation aléatoire w_k, on peut le caractériser de manière équivalente par ses **probabilités de transition** :

$$P(x_{k+1}|x_k, u_k, k) \tag{3.2}$$

Le lien entre ces deux représentations est direct. La probabilité de transitionner vers un état futur spécifique x_{k+1} est simplement la probabilité que la perturbation aléatoire w_k prenne la valeur exacte qui cause cette transition via la fonction f_k :

$$P\Big(x_{k+1} = f_k(x_k, u_k, w_k)\Big|x_k, u_k, k\Big) = P(w_k|x_k, u_k, k) \tag{3.3}$$

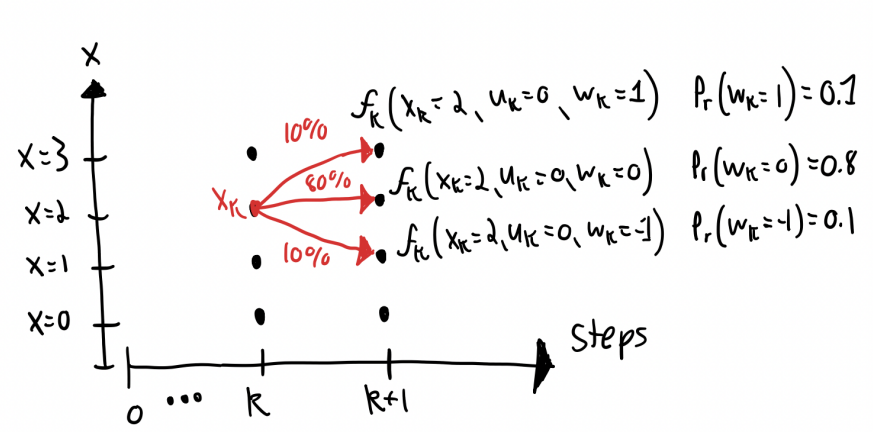


FIGURE 3.1 – Évolution stochastique avec des domaines discrets

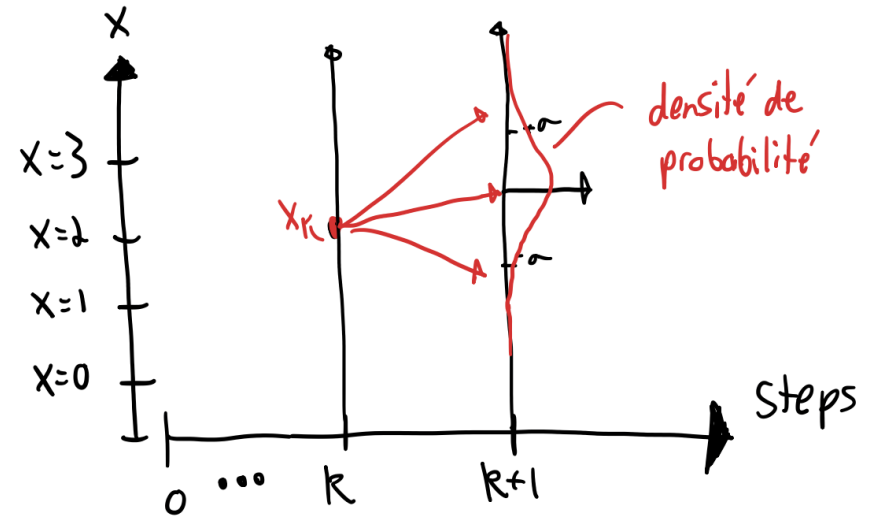


FIGURE 3.2 – Évolution stochastique avec des domaines continus

Il s'agit donc d'un choix de modélisation : on peut définir un environnement soit par sa fonction de dynamique et les probabilités des perturbations (équation (3.1)), soit directement par les probabilités de transition (équation (3.2)).

Propriété de Markov L'hypothèse fondamentale qui sous-tend la programmation dynamique est la propriété de Markov. Intuitivement, cela signifie que l'état observé x_k est "statistiquement suffisant" : il contient toute l'information pertinente de l'historique du système nécessaire pour prédire son futur. Cette définition est cohérente avec celle d'un "vecteur d'état" utilisé dans le domaine de la dynamique et de la commande. Formellement, dans un contexte probabiliste, cette propriété signifie que la probabilité de transitionner vers un état futur x_{k+1} est conditionnellement indépendante de l'historique passé, une fois que l'on connaît l'état présent x_k et l'action u_k :

$$P\Big(x_{k+1}\Big|x_k,u_k,k\Big)=P\Big(x_{k+1}\Big|x_k,u_k,k,\underbrace{x_{k-1},u_{k-1},\dots,x_0,u_0}_{\text{historique}}\Big) \tag{3.4}$$

Autrement dit, l'état x_k encapsule tout le passé pertinent ; il n'y a pas de "variables cachées" qui pourraient influencer le futur à notre insu.

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 la définition du coût-à-venir (équation (2.5)) en ajoutant un opérateur d'espérance $\mathbb{E}$ sur toutes les séquences de perturbations futures possibles :

$$J^\pi(x)=\mathop{\mathbb{E}}_{w_0,w_1,\dots,w_{N-1}}\left[\sum_{k=0}^{N-1}g_k(x_k,u_k,w_k)+g_N(x_N)\right],\quad\text{avec}\quad\begin{cases}x_0=x\\x_{k+1}=f_k(x_k,\pi_k(x_k),w_k)\end{cases} \tag{3.5}$$

Cette fonction objectif représente le coût-à-venir moyen, où chaque trajectoire future possible est pondérée par sa probabilité d'occurrence. Le problème à résoudre devient alors de trouver la politique qui minimise cette valeur attendue.
Heureusement, le principe d'optimalité et l'approche récursive de la programmation dynamique s'appliquent toujours. Pour résoudre ce problème, il suffit d'adapter l'algorithme en ajoutant un calcul d'espérance à l'intérieur de l'opération de minimisation. Autrement dit, on ne choisit plus l'action qui mène au meilleur état suivant de manière certaine, mais celle qui mène au meilleur coût-à-venir *en moyenne* :

$$J_k^*(x_k)=\min_{u_k}\mathop{\mathbb{E}}_{w_k}\left[g_k(x_k,u_k,w_k)+J_{k+1}^*\big(\underbrace{f_k(x_k,u_k,w_k)}_{x_{k+1}}\big)\right] \tag{3.6}$$

$$\pi_k^*(x_k)=\operatorname*{argmin}_{u_k}\mathop{\mathbb{E}}_{w_k}\left[g_k(x_k,u_k,w_k)+J_{k+1}^*\big(\underbrace{f_k(x_k,u_k,w_k)}_{x_{k+1}}\big)\right] \tag{3.7}$$

3.3 Formulation minimax

Une alternative à l'optimisation de l'espérance, souvent associée à la **commande robuste** ou à la théorie des jeux, consiste à se prémunir contre le **pire scénario possible**. Au lieu de minimiser le coût moyen, l'objectif est de minimiser le coût maximal qui pourrait survenir. Le coût-à-venir est donc redéfini en remplaçant l'opérateur d'espérance par un opérateur de maximisation sur les perturbations :

$$J^\pi(x)=\max_{w_0,\dots,w_{N-1}}\left[\sum_{k=0}^{N-1}g_k(x_k,u_k,w_k)+g_N(x_N)\right],\quad\text{avec}\quad\begin{cases}x_0=x\\x_{k+1}=f_k(x_k,\pi_k(x_k),w_k)\end{cases} \tag{3.8}$$

L'algorithme de programmation dynamique peut aussi être adapté à cette formulation. L'agent choisit l'action qui minimise son coût futur, en supposant que la perturbation (ou l'adversaire) choisira l'issue qui maximise ce même coût. On obtient alors la récursion **minimax** :

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \big) \right] \tag{3.9}$$

$$\pi_k^*(x_k) = \operatorname*{argmin}_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \big) \right] \tag{3.10}$$

Cette formulation est typiquement utilisée pour l'intelligence artificielle dans les jeux comme les échecs. Dans ce contexte, les actions de l'adversaire sont modélisées comme des perturbations w_k. Puisqu'on suppose que l'adversaire est rationnel et cherchera à nous nuire, il est logique d'anticiper qu'il choisira l'action qui maximise notre coût.

3.4 Observations Partielles

Jusqu'à présent, nous avons supposé que l'agent avait un accès direct et parfait à l'état x_k de l'environnement. En pratique, cette hypothèse est souvent irréaliste. Si l'observation de l'agent est incomplète ou bruitée, le problème gagne un niveau de complexité. Ce nouveau type de problème est appelé un **processus de décision markovien partiellement observable** (*Partially Observable Markov Decision Process* ou POMDP). De manière analogue à la modélisation de la dynamique stochastique, il existe deux approches équivalentes pour modéliser cette imperfection.
La première approche consiste à définir une **fonction d'observation** h_k qui dépend d'une variable de bruit aléatoire v_k. Cette formulation doit être complétée par la **loi de probabilité** qui régit ce bruit :

$$y_k = h_k(x_k, u_k, v_k), \quad \text{avec } v_k \in \mathcal{V}_k \tag{3.11}$$

où v_k doit être régi par une loi de probabilité qui fait partie du modèle :

$$P(v_k | x_k, u_k, k) \tag{3.12}$$

La seconde approche consiste à définir directement une **distribution de probabilité d'observation**, qui donne la probabilité d'obtenir l'observation y_k sachant l'état x_k et l'action u_k :

$$P(y_k | x_k, u_k, k) \tag{3.13}$$

Ces deux formalismes sont équivalents et il est possible de passer de l'un à l'autre.

Note : Dans le domaine de la commande, les observations y_k sont plus communément appelées les *sorties* du système. La fonction d'observation (équation (3.11)) est très utilisée dans les modèles d'état linéaires, où elle prend la forme classique

$$y = Cx + Du + v \tag{3.14}$$

Capsule vidéo
Observations partielles
`https://youtu.be/ZzT0L1Lqjio`

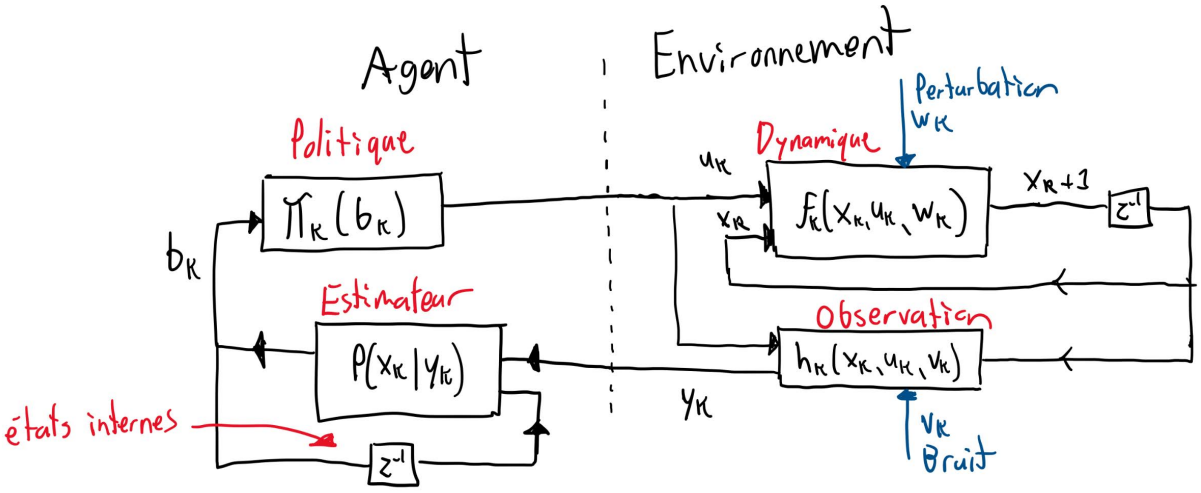


FIGURE 3.3 – Formulation mathématique incluant une fonction d'observation, qui modélise le cas où l'agent n'observe pas les états directement. et un agent avec une dynamique d'estimation.

3.4.1 Espace Croyance

Dans un système partiellement observé, l'observation y_k ne contient pas toute l'information sur l'état réel x_k et ne rencontre pas la propriété de Markov discuté précédemment (équation (3.4)). Pour agir de manière optimale, l'agent ne peut donc pas se baser uniquement sur sa dernière observation ; il doit tenir compte de tout le contexte qu'il a accumulé.

L'historique comme état informationnel

La manière la plus directe de capturer ce contexte est de définir un **état informationnel** qui est l'historique complet des actions et observations passées :

$$I_k = \{y_0, u_0, y_1, u_1, \ldots, y_{k-1}, y_k\} \tag{3.15}$$

Par définition, cet historique I_k est une **statistique suffisante** : il contient toute l'information disponible et respecte donc la propriété de Markov. Théoriquement, il est donc possible de définir une politique optimale qui dépend de cet historique, $u_k = \pi_k^*(I_k)$. Le problème majeur de cette approche est d'ordre pratique : la dimension de I_k augmente à chaque étape, ce qui nécessiterait une mémoire infinie et rendrait les calculs rapidement infaisables.

La croyance comme état informationnel compact

La solution élégante au problème de l'historique de taille croissante est de trouver une représentation de l'état informationnel qui soit *de taille fixe*. C'est le rôle de la **croyance** (*belief*). La croyance est l'état de connaissance de l'agent, représenté par une distribution de probabilité sur l'état réel x_k. Cette distribution est entièrement définie par un paramètre de taille fixe, noté b_k, que l'on nomme le **belief state**.
La forme de b_k dépend de la nature du problème. Par exemple, pour un espace d'états discret, b_k pourrait être le vecteur des probabilités pour chaque état :

$$b_k \triangleq \{P(x_k = x \mid I_k) \text{ pour tout } x \in \mathcal{X}\} \tag{3.16}$$

Pour une incertitude gaussienne, b_k serait la moyenne et la matrice de covariance (μ_k, Σ_k) de la distribution. L'avantage fondamental de cette approche est que b_k est une **statistique suffisante** : il possède la propriété de Markov, car il compresse tout l'historique pertinent sans perte d'information utile pour prédire le futur.

$$P\Big(x_{k+1}\Big|b_k, u_k, k\Big) = P\Big(x_{k+1}\Big|b_k, u_k, k, \underbrace{y_{k-1}, u_{k-1}, \ldots, y_0, u_0}_{\text{historique passé}}\Big) \tag{3.17}$$

Cela permet de convertir le problème original (POMDP sur l'espace des états x) en un problème standard (MDP sur l'espace des croyances paramétré par b). La politique optimale devient alors une fonction de ce paramètre de croyance :

$$u_k = \pi_k^*(b_k) \tag{3.18}$$

Il faut noter que b_k n'est pas statique ; l'agent doit le mettre à jour à chaque nouvelle observation.
Théoriquement, cette mise à jour (de b_k à b_{k+1}) se fait en appliquant la **loi de Bayes**. En pratique, cette mise à jour exacte est souvent impossible à calculer pour des problèmes complexes. On utilise donc des outils spécifiques :

— Le **filtre de Kalman** est une solution exacte pour mettre à jour les paramètres (moyenne et covariance) d'une croyance gaussienne dans le cas des systèmes linéaires.
— Des méthodes approximatives comme le **filtre à particules** sont utilisées dans les cas plus généraux.

3.5 Politique Stochastique

Jusqu'à présent, nous avons considéré des politiques déterministes ($u_k = \pi_k(x_k)$), où une seule action est choisie pour un état donné. Il est cependant parfois avantageux d'utiliser une **politique stochastique**, qui définit une distribution de probabilité sur l'ensemble des actions possibles. Dans ce contexte, la notation π est réutilisée pour représenter cette probabilité conditionnelle :

$$\pi_k(u|x) \triangleq P(u_k = u|x_k = x) \tag{3.19}$$

Cette expression représente la probabilité de choisir l'action u à l'étape k, sachant que l'on se trouve dans l'état x. Cette approche peut sembler contre-intuitive. En effet, dans un contexte d'information parfaite (état x_k observé et dynamique f_k connue), la politique optimale est **généralement déterministe**. La seule exception est le cas où plusieurs actions distinctes mènent exactement au même coût-à-venir optimal ; dans cette situation, une politique stochastique qui choisit aléatoirement parmi ce sous-ensemble d'actions optimales est également considérée comme optimale. Cependant, la stochasticité devient essentielle dans les cas d'**information imparfaite**, notamment pour permettre l'**exploration** dans les algorithmes d'apprentissage par renforcement (lorsque le modèle est inconnu) ou pour gérer l'incertitude dans les problèmes à observations partielles (POMDP).

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'évolutiosn
`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, lorqu'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 :

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

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

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{4.2}$$

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

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

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

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

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

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

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

$$\dot{x} = Ax + Bu + w \tag{4.6}$$

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 :

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

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 défini l'état à l'étape suivante basé sur l'état actuel et une action choisie :

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

C'est une formulation qu'on obtient naturellement pour des environnements artificiels comme des jeux ou 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 vu d'un micro-contrôleur qui prend des décisions dans une boucle qui s'exécute en temps fini. Par exemple, une fonction de transfert utilisant la transformée en Z prend directement cette forme lorsqu'on utilise la représentation d'état. Dans le domaine des asservissements, on travaille souvent avec une approximation linéaire de ce type d'équation que l'on note :

$$x_{k+1} = A_k x_k + B_k u_k + w_k \qquad k = 0, 1, \ldots, N-1 \tag{4.9}$$

4.3 Graphes (états discrets déterministes)

Pour les systèmes ou 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 jeux 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 indexes identifiant un état actuel, un état futur, une action et un coût (récompense). **Attention !** Certaines parties antérieures du matériel (comme les capsultes vidéos) utilisent la notation i et j plutôt que s et s'.

$$x_k \Leftrightarrow s \text{ index du noeud de départ} \tag{4.10}$$
$$u_k \Leftrightarrow s' \text{ index de la destination de l'arc d'action} \tag{4.11}$$
$$x_{k+1} \Leftrightarrow s' \text{ index du noeud d'arrivé} \tag{4.12}$$
$$J_k^*(x_k) \Leftrightarrow J_k^*(s) \tag{4.13}$$
$$\pi_k^*(x_k) \Leftrightarrow \pi_k^*(s) \tag{4.14}$$
$$g_k(x_k, u_k) \Leftrightarrow g_k(s, s') \text{ longueur de l'arc } ss' \tag{4.15}$$

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

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

$$\vdots \tag{4.17}$$

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

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

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

4.3.1 Discrétisation 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.

$$x_k \Leftrightarrow s \text{ index du noeud de départ} \tag{4.20}$$

$$u_k \Leftrightarrow a \text{ index de l'action} \tag{4.21}$$

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

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

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

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' :

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

et une carte des coûts donnée par

$$g_k = g_k(s, s', a) \tag{4.26}$$

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, ou l'opérateur de l'espérence peut être remplacé par une somme avec les probabilités :

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

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

Concrètement, le modèle de l'environnement dans cette situation peut être encodé comme un tenseur d'ordre 3 (si les probabilité 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=5kJxeVAlBuuOn_JW`

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

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

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 :

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

avec

$$x_k \Leftrightarrow s \text{ index du noeud de départ} \tag{4.31}$$
$$u_k \Leftrightarrow a \text{ index de l'action} \tag{4.32}$$
$$x_{k+1} \Leftrightarrow s' \text{ index du noeud d'arrivé} \tag{4.33}$$
$$f_k(x_k,u_k,w_k) \Leftrightarrow p_k(s',r|s,a) \text{ probabilités de transitionner vers } s' \text{ avec un coût } g_k(r) \tag{4.34}$$
$$g_k(x_k,u_k,w_k) \Leftrightarrow g_k(r) \text{ liste de coût possibles} \tag{4.35}$$

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

$$J^*_k(s) = \min_{a\in\mathcal{U}(s)} \sum_{s'}\sum_{r} p_k(s',r|s,a)\Big[g_k(r) + J^*_{k+1}(s')\Big] \tag{4.36}$$

$$\pi^*_k(s) = \operatorname*{argmin}_{a\in\mathcal{U}(s)} \sum_{s'}\sum_{r} p_k(s',r|s,a)\Big[g_k(r) + J^*_{k+1}(s')\Big] \tag{4.37}$$

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-à-venir 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 ou l'horizon de temps est infini et rien d'est dépendent de l'étape actuelle, la situation spécique 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 :

$$f_k(x_k, u_k, w_k) \Rightarrow f(x, u, w) \tag{5.1}$$
$$g_k(x_k, u_k, w_k) \Rightarrow g(x, u, w) \tag{5.2}$$

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

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

ou on introduit un facteur d'escompte $0 \leq \alpha \leq 1$ qui représente un biais pour moins prendre en considération les coûts/récompenses loins dans le futur. Le facteur d'escompte 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 :

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

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 $N \to \infty$, 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 :

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

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

qui est nécessaire et suffisante pour confirmer l'optimalité d'un coût-à-venir 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 indexes ! 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/18KrNlHHT3E?si=F1OOxS1UCOKATgCs

Capsule vidéo
Élments de l'équation de Bellman
https://youtu.be/18KrNlHHT3E?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-l-PRdZibSPT

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

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

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

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

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

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

, et si l'évolution est déterministe, alors :

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

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

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

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

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

5.3 Coût-à-venir d'une politique

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

$$u = \pi(x) \tag{5.15}$$

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.

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

À noter, que dans le cas ou la politique est stochastique, il faut calculer l'espérance selon les probabilité 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 :

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

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 :

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

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 ou 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 ou 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 :

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

et une matrice de probabilité de transition pour la politique :

$$p_{s's}^\pi = \sum_a \pi_{as} \sum_{s'} p_{s'as} \tag{5.20}$$

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

$$J^\pi = r^\pi + \alpha P^\pi J^\pi \tag{5.21}$$

et on peut isoler le vecteur de coup-à-venir :

$$J^\pi = \left[I - \alpha P^\pi\right]^{-1} r^\pi \tag{5.22}$$

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-à-venir $J^*(x,t)$ et la politique $\pi^*(x,t)$ avec :

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

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

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

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

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

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 ou 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/ggOIYkQwXWs?si=tgLOd6LhGVq5KAal`

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

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

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 :

$$\mathbb{E}\left[\underline{w}_k\right] = \underline{0} \tag{6.2}$$

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

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

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

$$Q_k \geq 0 \qquad R_k > 0 \tag{6.4}$$

En appliquant l'algorithme de programmation dynamique on trouve que :
1) le coût-à-venir d'un état $\underline{x}_k$ a la forme quadratique suivante :

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

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 dépent pas de l'état actuel.
2) la loi de commande optimale a la forme linéaire suivante :

$$\pi_k^*(\underline{x}_k) = -K_k\underline{x}_k \tag{6.6}$$

où K_k est la matrice de gain égale à

$$K_k = \left[R_k + B_k^T S_{k+1} B_k\right]^{-1} B_k^T S_{k+1} A_k \tag{6.7}$$

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 :

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

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 :

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

où K est la matrice de gain égale à

$$K = \left[R + B^T S B\right]^{-1} B S A \tag{6.10}$$

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

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

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-à-venir fini peut être garantie par le critère de controllabilité du système, qui est une propriété qui se calcul basé sur la matrice A et B. Celle-çi 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-à-venir 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.

Capsule vidéo
LQR à partir de HJB
`https://youtu.be/YhXwjBQnU-M?si=LPLs8WgaxAWy_f6l`

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

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

où $\underline{x}$ et $\underline{w}$ sont des vecteurs de dimension n et $\underline{w}$ 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 :

$$\mathbb{E}\left[\underline{w}\right] = \underline{0} \tag{6.13}$$

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 :

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

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

$$Q(t) \geq 0 \qquad R(t) > 0 \quad \forall t \tag{6.15}$$

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 :

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

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

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

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

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 :

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

et la politique optimale va avoir la forme linéaire suivante :

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

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 :

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

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 :

$$S(t_f) = S_f \tag{6.22}$$

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 :

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

où K est la matrice de gain égale à

$$K^* = R^{-1}B^TS \tag{6.24}$$

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

$$0 = SA + A^T - SBR^{-1}B^TS + Q \tag{6.25}$$

qui est en fait l'équation de Bellman pour ce cas particulier. Il existe des méthodes numérique efficace pour résoudre cette équation matricielle quadratique. La solution existe si le système dynamique a une propriété qu'on appelle la controllabilité, 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 :

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

sur une trajectoire nominale :

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

La dynamique de l'erreur :

$$\underline{x}_e = \underline{x}_d - \underline{x} \tag{6.28}$$

est égale à

$$\dot{\underline{x}}_e = \dot{\underline{x}}_d - \dot{\underline{x}} \tag{6.29}$$

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

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 prendre une forme linéaire mais dépendant du temps qui peut être solutionné avec la solution LQR pour un horizon de temps fini.

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

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

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

Donc pour un système non-linéaire, sur une trajectoire nominale spécifique, le problème de stabilisation d'une erreur relative à cette trajectoire peut être attaquée 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-à-venir pour chaque état x, et ensuite itérer avec l'opération de programmation dynamique suivante :

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

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-à-venir optimal $J^*(x)$. Ensuite, la politique optimale peut être calculée avec :

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

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 ou 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 :

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

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-à-venir d'une politique spécifique avec les itérations suivantes :

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

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

7.2 Algorithme d'itération de politique

Capsule vidéo
Iteration de politique
`https://youtu.be/5b1o88O8e44?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 :

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

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

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

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

Cette section présente les principes de bases des méthodes sans-modèle (*model-free*), qui visent à trouver des solutions aux équations de Bellman basées sur des interactions avec un environnement inconnu pour lequel on ne connaît pas les équations, i.e. soit la fonction $x' = f(x, u)$ ou les probabilités $P(x'|x, u)$. Il est à noter que le cadre pour les sections sur l'apprentissage est **un horizon de temps infini et un système stationnaire**.

8.1 Valeurs Q

Le premier fondement qui va nous permettre d'apprendre sans modèle est de travailler avec le coût-à-venir de prendre une certaine action u à un état x, appelé la valeur $Q(x, u)$, plutôt que la fonction coût-à-venir $J(x)$ qui représente le coût-à-venir avant que l'action soit sélectionnée. Formellement, pour un système déterministe, la valeur Q est définie par :

$$Q(x, u) = g(x, u) + \alpha J(x') \tag{8.1}$$

où $J(x')$ est le coût-à-venir de l'état futur. Donc la valeur Q est le coût instantané de choisir l'action u, plus le coût-à-venir de l'état suivant. Comme c'est le cas pour le coût-à-venir $J(x)$, la valeur Q dépend de la politique utilisée. On va noter $Q^\pi(x, u)$ une valeur Q d'une politique arbitraire π, et $Q^*(x, u)$ une valeur Q de la politique optimale.
L'équation de Bellman peut être réécrite en fonction de ces valeurs. Pour un système déterministe on obtient :

$$Q^*(x, u) = g(x, u) + \alpha \min_{u'} Q^*(x', u') \tag{8.2}$$

$$\pi^*(x) = \operatorname*{argmin}_u Q^*(x, u) \tag{8.3}$$

Le point important ici est que l'équation de la dynamique $f(x, u)$ n'apparaît pas explicitement dans cette version des équations de Bellman. Premièrement, si on connaît les valeurs Q, on peut calculer la politique optimale directement sans connaître le modèle, en de plus on a mesuré toutes les valeurs $Q^*(x', u')$ possibles, on peut calculer $Q^*(x, u)$ sans le modèle.
Si par exemple on a observé une trajectoire :

$$(x_0, u_0, g_0, x_1, u_1, g_1, x_2, u_2, g_2, \dots) \tag{8.4}$$

chaque tuple de transition $\{x_k, u_k, g_k, x_{k+1}\}$ nous donne l'information nécessaire pour faire une mise à jour de la valeur $Q^*(x_k, u_k)$.
C'est le principe de base d'un algorithme appelé apprentissage-Q, fundamental en apprentissage par renforcement.

Capsule vidéo
Q-Learning
`https://youtu.be/eyUcnKOpwsE?si=45HiCPObZoUF4wow`

8.2 Échantillonnage

Pour un système stochastique, la définition des valeurs Q devient l'espérance de la somme du coût immédiat et du coût futur :

$$Q(x,u) = \mathbb{E}\left[g(x,u) + \alpha J(x')\right] \tag{8.5}$$

et l'équation de Bellman optimale est donnée par :

$$Q^*(x,u) = \mathbb{E}\left[g(x,u) + \alpha \min_{u'} Q^*(x',u')\right] \tag{8.6}$$

Dans les chapitres précédents, les algorithmes de planification calculaient ces espérances de façon exacte car ils avaient accès au modèle complet (les ensembles de perturbations possibles et leurs probabilités d'occurrence). Une idée centrale de l'apprentissage par renforcement est de **remplacer cette espérance mathématique par une moyenne empirique obtenue par échantillonnage**. Si nous observons N échantillons aléatoires indépendants $X_1, X_2, \ldots, X_N$ d'une même variable aléatoire X, la loi des grands nombres nous garantit que leur moyenne converge vers la vraie espérance :

$$\mathbb{E}[X] = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} X_i \tag{8.7}$$

C'est ce principe qui permet d'apprendre directement de l'expérience. Appliqué aux valeurs Q, nous cherchons à estimer :

$$Q(x,u) \approx \frac{1}{N} \sum_{i=1}^{N} q_i \tag{8.8}$$

où chaque q_i est un **échantillon** du coût total observé après avoir pris l'action u dans l'état x lors d'une vraie trajectoire du système.

8.3 Évaluation

Ensuite, il y a deux grandes méthodes de base pour calculer nos échantillons q_i basés sur des trajectoires observées.

8.3.1 Monte-Carlo

La méthode de Monte-Carlo (MC) est conceptuellement la plus simple : elle attend la **fin d'un épisode** complet pour calculer le coût-à-venir réel observé. Pour un état x_k visité à l'instant k dans un épisode qui se termine au temps T, l'échantillon q_k est simplement la somme des coûts réellement encourus jusqu'à la fin :

$$q_k^{\text{MC}} = \sum_{t=k}^{T} \alpha^{t-k} g_t \tag{8.9}$$

La mise à jour de $Q(x,u)$ ne se fait qu'une fois l'épisode terminé.
Propriétés :

- L'estimateur est **non biaisé** car il utilise le vrai retour total.
- Il a une **haute variance** car le retour total dépend de nombreuses actions et transitions stochastiques futures.
- Il ne peut s'appliquer qu'aux tâches épisodiques (qui ont une fin).

8.3.2 TD-Learning (Différence Temporelle)

La méthode de différence temporelle (TD) n'attend pas la fin de l'épisode. Elle utilise une estimation de la valeur de l'état suivant pour mettre à jour l'état courant. C'est le principe du **bootstrapping**.
L'échantillon q_k est construit à partir du coût immédiat observé g_k et de l'estimation *actuelle* de la valeur de l'état suivant $J(x')$:

$$q_k^{\mathrm{TD}} = g_k + \alpha J(x_{k+1}) \tag{8.10}$$

La mise à jour peut se faire à chaque pas de temps, immédiatement après avoir observé la transition (x_k, g_k, x_{k+1}).
Propriétés :

- L'estimateur est **biaisé** initialement, car il dépend de l'estimation courante $J(x_{k+1})$ qui peut être fausse au début.
- Il a une **faible variance** car il ne dépend que d'une seule transition stochastique à la fois.
- Il peut s'appliquer à des tâches continues (sans fin) et permet un apprentissage en temps réel.

8.3.3 TD-λ

À venir !

8.4 Approximation de fonctions

Capsule vidéo
RL avec approximation
`https://youtu.be/CGbdEDGsJnU?si=BDLgkYyqqyOR_RYz`

8.4.1 Le Problème : La Malédiction de la Dimensionnalité

Dans les chapitres précédents, nous avons établi que la solution à un problème de décision séquentielle est caractérisée par une fonction de coût-à-venir optimale, $J^*(x)$, ou une fonction de valeur-action, $Q^*(x, u)$.
Pour des problèmes avec des états et actions **discrets** et de petite taille (comme la navigation sur un graphe), nous pouvions représenter ces fonctions sous forme de **tableaux** (*lookup table*), où chaque case (x, u) stockait une valeur $Q(x, u)$.
Cependant, cette approche ne se met pas bien à l'échelle pour des problèmes plus réalistes, par exemple :

- **Espaces d'états continus** : En robotique, l'état $x = [\theta, \dot{\theta}]$ est un vecteur de nombres réels. Il y a une infinité d'états, rendant un tableau impossible. L'approche évidente, la **discrétisation** (diviser chaque variable continue en k intervalles), se heurte à un mur : si l'état a d variables (dimensions), le nombre total d'états discrets résultants est k^d, un nombre qui croît de façon **exponentielle** avec la dimension de l'état.
- **Espaces d'états discrets de grande taille** : Pour un jeu comme le Go ou les échecs, le nombre d'états est astronomique, dépassant le nombre d'atomes dans l'univers.

C'est ce qu'on appelle la **malédiction de la dimensionnalité** (*Curse of Dimensionality*).
La solution est de cesser de mémoriser les valeurs exactes pour chaque état, et d'utiliser l'**apprentissage machine** pour apprendre une **fonction paramétrée** qui *approxime* la fonction de valeur.

8.4.2 Formulation Générale

L'idée est d'approximer notre fonction cible (que nous nommerons $f(x)$ de façon générique) par un approximateur $\hat{f}(x|w)$. Cet approximateur est défini par un vecteur de **paramètres** (ou **poids**) w, qui sont des arguments supplémentaire qu'on va ajuster lors d'un entraînement.

$$f(x) \approx \hat{f}(x|w)$$

L'annexe B résume les grandes lignes des mathématiques de l'approximation de fonction, et les familles d'approximation les plus courantes. Dans les dernières années, avec les succès de l'apprentissage profond,

il est devenu la norme d'utiliser des réseaux de neurones comme méthode d'approximation de fonction en apprentissage par renforcement.

Capsule vidéo
Approximation de fonctions
`https://youtu.be/p8BizsV8apQ?si=Tj0lqSOQZwZmtn9u`

Dans notre contexte, on va spécifiquement essayer d'approximer soit nos fonctions valeurs :

$$J^*(x) \approx \hat{J}(x|w) \tag{8.11}$$

$$Q^*(x,u) \approx \hat{Q}(x,u|w) \tag{8.12}$$

soit directement la politique :

$$\pi^*(x) \approx \hat{\pi}(x|w) \tag{8.13}$$

Le but est de modifier nos algorithmes, pour non pas apprendre sous la forme d'un grand tableau, mais sous la forme d'un vecteur de paramètres w^* qui approxime ce grand tableau avec beaucoup moins de paramètres.

8.4.3 Apprentissage des Paramètres

On peut modifier nos algorithmes d'apprentissage, comme l'itération de valeurs ou le *Q-learning*, pour agir avec une mise à jour des paramètres, plutôt que de mettre à jour un tableau. En incorporant l'idée d'une descente de gradient stochastique, on cherche à minimiser une fonction de perte (Loss) entre notre prédiction et une "valeur cible" qu'on aurait mis dans dans notre tableau dans un monde parfait.
La perte pour un seul échantillon de transition k est :

$$\mathcal{L}_k(w) = \frac{1}{2}\left(y_k - \hat{Q}(x_k,u_k|w)\right)^2 \tag{8.14}$$

où $\hat{Q}(x_k,u_k|w)$ est notre **prédiction** actuelle.
La règle de mise à jour de la descente de gradient stochastique (SGD) vise à ajuster w pour réduire cette perte :

$$w \leftarrow w - \eta \nabla_w \mathcal{L}_k(w) \tag{8.15}$$

$$w \leftarrow w - \eta\left(-(y_k - \hat{Q}(x_k,u_k|w))\nabla_w \hat{Q}(x_k,u_k|w)\right) \tag{8.16}$$

$$w \leftarrow w + \eta \underbrace{\left(y_k - \hat{Q}(x_k,u_k|w)\right)}_{\text{Erreur de prédiction}} \nabla_w \hat{Q}(x_k,u_k|w) \tag{8.17}$$

Cible

La question clé est : quelle est notre "cible" y_k ? En apprentissage par renforcement, ça serait la fonction valeur, qui est initialement inconnue. **Nous n'avons pas de "vraie" réponse, contrairement à l'apprentissage supervisé.** Ensuite, il y a plusieurs méthodes pour choisir cette cible. Avec l'approche *TD-learning*, nous utilisons l'équation de Bellman pour créer la cible, avec les valeurs les plus à jours possible (bootstrap target). Pour le *Q-learning*, la cible y_k est le coût immédiat g_k plus le coût-à-venir minimal estimé à partir de l'état suivant :

$$y_k = g_k + \alpha \min_{u'} \hat{Q}(x_{k+1},u'|w) \tag{8.18}$$

L'Algorithme avec Semi-Gradient

En substituant cette cible dans notre règle de mise à jour SGD, "l'erreur de prédiction" devient l'**erreur de différence temporelle (TD-Error)**, notée δ_k :

$$\delta_k = \left(g_k + \alpha \min_{u'} \hat{Q}(x_{k+1},u'|w)\right) - \hat{Q}(x_k,u_k|w) \tag{8.19}$$

La mise à jour finale des poids pour le *Q-learning* avec approximation de fonction (aussi appelée **Deep Q-Learning** lorsque $\hat{Q}$ est un réseau de neurones) est donc :

$$w \leftarrow w + \eta \cdot \delta_k \cdot \nabla_w \hat{Q}(x_k, u_k|w) \tag{8.20}$$

Note : On appelle cette méthode "semi-gradient" car on ignore le gradient de la cible y_k par rapport à w lors de la dérivation. On traite la cible comme si c'était une étiquette fixe, ce qui est crucial pour la stabilité de l'algorithme.

Chapitre 9

Apprentissage par renforcement

9.1 Méthodes basées sur la valeur

9.1.1 Q-Learning

Le Q-Learning est un algorithme *off-policy* (hors-politique) : il peut apprendre les valeurs Q^* de la politique optimale en utilisant des données générées par une autre politique (arbitraire), tant que celle-ci explore suffisamment l'espace d'états.

Apprentissage

L'objectif est de mettre à jour itérativement les estimations de $Q(x, u)$ vers la cible optimale définie par l'équation de Bellman, basé sur des tuples de transition $\{x, u, g, x'\}$ observés.
Pour un système déterministe on pourrait simplement applique l'équation de Bellman :

$$Q(x_k, u_k) \leftarrow g_k + \alpha \min_{u'} Q(x_{k+1}, u') \tag{9.1}$$

Pour un système stochastique, on utilise un taux d'apprentissage $\eta \in (0, 1]$ pour effectuer une moyenne glissante sur nos échantillons (la cible TD) :

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \eta \left[\underbrace{g_k + \alpha \min_{u'} Q(x_{k+1}, u')}_{\text{Cible TD}} - Q(x_k, u_k) \right] \tag{9.2}$$

Finalement, avec une approximation de fonction $\hat{Q}(x, u|w)$, on met à jour les poids w par descente de gradient sur l'erreur TD :

$$w \leftarrow w + \eta \left[g_k + \alpha \min_{u'} \hat{Q}(x_{k+1}, u'|w) - \hat{Q}(x_k, u_k|w) \right] \nabla_w \hat{Q}(x_k, u_k|w) \tag{9.3}$$

Capsule vidéo
Q-Learning
`https://youtu.be/eyUcnKOpwsE?si=45HiCPObZoUF4wow`

Politique d'exploration

La politique utilisée pour générer les données doit explorer tous les états et actions possibles. Typiquement, on utilise la méthode ϵ-greedy (epsilon-gourmande) :

— Avec une probabilité ϵ, choisir une action **aléatoire** (exploration).
— Avec une probabilité $1 - \epsilon$, choisir l'**meilleure** action connue actuellement (exploitation).

Mathématiquement :

$$\pi(x) = \begin{cases} \text{aléatoire} \in \mathcal{U} & \text{avec probabilité } \epsilon \\ \underset{u \in \mathcal{U}}{\operatorname{argmin}} \hat{Q}(x,u) & \text{avec probabilité } 1-\epsilon \end{cases} \tag{9.4}$$

Souvent, la valeur de ϵ est réduite progressivement au cours de l'apprentissage (par exemple, de 1.0 à 0.1) pour favoriser l'exploration au début et l'exploitation à la fin.

9.1.2 SARSA

Pour utiliser la méthode SARSA (*State-Action-Reward-State-Action*), contrairement au Q-Learning, on va apprendre la valeur de la politique *actuellement suivie* par l'agent, y compris son comportement d'exploration. C'est un algorithme **on-policy** (en-politique).

Apprentissage

La règle de mise à jour utilise l'action u_{k+1} qui est *réellement choisie* par l'agent à l'état suivant, plutôt que la meilleure action possible. On utilise donc des tuples de transition $\{x, u, g, x', u'\}$ observés.
Pour un système stochastique avec un taux d'apprentissage η :

$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \eta \left[\underbrace{g_k + \alpha Q(x_{k+1},u_{k+1})}_{\text{Cible TD}} - Q(x_k,u_k) \right] \tag{9.5}$$

Avec approximation de fonction $\hat{Q}(x,u|w)$, la mise à jour des poids est :

$$w \leftarrow w + \eta \left[g_k + \alpha \hat{Q}(x_{k+1},u_{k+1}|w) - \hat{Q}(x_k,u_k|w) \right] \nabla_w \hat{Q}(x_k,u_k|w) \tag{9.6}$$

Politique

Comme SARSA apprend la valeur de la politique suivie, il est crucial que cette politique s'améliore au fil du temps pour converger vers l'optimale. On utilise généralement une politique ϵ-greedy qui devient de plus en plus "greedy" (gourmande).

9.2 Méthodes basées sur la politique

Contrairement aux méthodes basées sur des approximations du coût-à-venir (comme Q-Learning et SARSA) qui apprennent d'abord une fonction Q pour en déduire une politique, les méthodes basées sur la politique cherchent à optimiser directement une politique paramétrée par des paramètres θ sans passer par l'intermédiaire des valeurs Q. Typiquement dans ce contexte, la politique va être stochastique, et on représente la densité de probabilité de sélectionner une action u dans l'état x par :

$$\pi_\theta(u|x) = P(u|x,\theta) \tag{9.7}$$

ou θ est un vecteur de paramètres.

9.2.1 Gradient de Politique (Policy Gradient)

L'idée fondamentale est de voir le coût total espéré comme une fonction directe des paramètres de la politique, notée $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{k=0}^{\infty} \alpha^k g_k \right]$$

On cherche alors à minimiser ce coût en effectuant une **descente de gradient** sur les paramètres θ :

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \tag{9.8}$$

Le **Théorème du Gradient de Politique** nous fournit une formule pour calculer ce gradient sans connaître la dynamique du système. Il établit que le gradient du coût total est proportionnel à l'espérance du gradient du logarithme de la politique, pondéré par la valeur Q :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \ln \pi_\theta(u|x) Q^{\pi_\theta}(x,u) \right] \tag{9.9}$$

Intuitivement, cela signifie que pour réduire le coût, on doit déplacer les paramètres θ dans une direction qui *diminue* la probabilité des actions ayant une valeur Q élevée (un coût élevé), et qui *augmente* celle des actions ayant une valeur Q faible.

9.2.2 REINFORCE

L'algorithme REINFORCE est une méthode de Monte-Carlo pour estimer ce gradient. Il utilise le retour total réel q_k^{MC} observé à la fin d'un épisode comme estimation non-biaisée de $Q^{\pi_\theta}(x,u)$.
Pour chaque étape k d'un épisode généré, la mise à jour des paramètres se fait dans la direction qui réduit le coût total espéré :

$$\theta \leftarrow \theta - \eta \cdot q_k^{MC} \cdot \nabla_\theta \ln \pi_\theta(u_k|x_k) \tag{9.10}$$

où $q_k^{MC} = \sum_{t=k}^{T} \alpha^{t-k} g_t$ est le coût-à-venir observé à partir de l'étape k jusqu'à la fin de l'épisode T.

9.3 Exploration

9.3.1 Exploitation vs. exploration

9.3.2 n-armed bandit

À venir !

9.4 Taxonomie des Algorithmes

Le tableau 9.1 résume les principales distinctions entre les familles d'algorithmes d'apprentissage par renforcement vues dans ce chapitre.

Variante	**Description**	**Équation de mise à jour principale (exemple)**
On-policy	Apprend la valeur de la politique suivie (π), incluant l'exploration.	SARSA : $Q(x,u) \leftarrow Q(x,u) + \eta[g + \alpha Q(x',u') - Q(x,u)]$
Off-policy	Apprend la valeur de la politique optimale (π^*) tout en suivant une politique d'exploration différente.	Q-Learning : $Q(x,u) \leftarrow Q(x,u) + \eta[g + \alpha \min_{u'} Q(x',u') - Q(x,u)]$
Monte Carlo	Met à jour les valeurs à la fin d'un épisode complet en utilisant les échantillons de Q réel.	$Q \leftarrow Q + \eta[q - Q]$
TD Learning	Met à jour les valeurs à chaque étape en utilisant une estimation de la valeur future (bootstrapping).	$J(x) \leftarrow J(x) + \eta[g + \alpha J(x') - J(x)]$
Value-based	Apprend une fonction de valeur (J ou Q) et en dérive la politique.	Q-Learning (voir ci-dessus)
Policy-based	Apprend directement une politique paramétrée $\pi_\theta(u\|x)$ par descente de gradient.	REINFORCE : $\theta \leftarrow \theta - \eta \cdot q \cdot \nabla_\theta \ln \pi_\theta(u\|x)$

TABLE 9.1 – Résumé des variantes d'algorithmes d'apprentissage par renforcement

Annexe A

Outils mathématiques

A.1 Ensembles

Pour indiquer qu'une variable x est un élément d'un ensemble de valeurs possibles $\mathcal{X}$, on utilise le symbole d'appartenance $\in$ et on note $x \in \mathcal{X}$. Un ensemble avec un nombre fini d'éléments peut être défini en listant directement ses membres entre accolades :

$$\textbf{Liste :} \quad \mathcal{X} = \{1, 2, 3, 4, 5\} \tag{A.1}$$

Pour les variables continues, on utilise souvent des intervalles. Un intervalle peut inclure ses bornes (inclusif) ou les exclure (exclusif). Par exemple :

$$\textbf{Intervalle inclusif :} \quad \mathcal{X} = [0, 10] \quad \text{Dans ce cas} \quad 0 \in \mathcal{X} \quad \text{et} \quad 12 \notin \mathcal{X} \tag{A.2}$$

$$\textbf{Intervalle exclusif :} \quad \mathcal{X} = (0, 10) \quad \text{Dans ce cas} \quad 0 \notin \mathcal{X} \quad \text{et} \quad 9.999 \in \mathcal{X} \tag{A.3}$$

Une autre manière de définir un ensemble est par condition, en spécifiant la propriété que ses éléments doivent respecter. L'intervalle inclusif précédent peut ainsi s'écrire :

$$\textbf{Condition :} \quad \mathcal{X} = \{x \in \mathbb{R} \,|\, 0 \leq x \leq 10\} \tag{A.4}$$

Cette notation se lit : "$\mathcal{X}$ est l'ensemble des nombres réels x qui sont supérieurs ou égaux à zéro et inférieurs ou 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

Pour une variable aléatoire discrète X qui peut prendre un ensemble fini de N valeurs $\{x_1, x_2, \ldots, x_N\}$, on note la probabilité qu'elle prenne une valeur spécifique x_i par l'expression suivante, qui doit être une valeur entre 0 et 1 :

$$P(X = x_i) \in [0, 1] \tag{A.5}$$

Pour une notation plus compacte, on utilise parfois :

$$p_i \triangleq P(X = x_i) \tag{A.6}$$

où p_i est la probabilité de la i-ème issue possible. Pour qu'un modèle probabiliste soit valide, la somme de toutes les probabilités doit être égale à 1 :

$$\sum_{i}^{N} p_i = 1 \tag{A.7}$$

Exemple 1.

Considérons le lancer d'un dé équilibré à six faces. La variable aléatoire X représente le résultat du lancer. L'ensemble des issues possibles (l'univers) est :

$$X \in \{1, 2, 3, 4, 5, 6\}$$

Si le dé est équilibré, la probabilité est la même pour chaque face :

$$P(X = 1) = 1/6$$
$$P(X = 2) = 1/6$$
$$P(X = 3) = 1/6$$
$$P(X = 4) = 1/6$$
$$P(X = 5) = 1/6$$
$$P(X = 6) = 1/6$$

A.2.2 Probabilité pour une variable continue

Une variable aléatoire continue, peut prendre une infinité de valeur possible, par exemple le cas le plus simple serait une valeur aléatoire avec comme univers (ensemble de possibilité) les nombres réels $\mathbb{R}$. Contrairement aux variables discrètes, la probabilité qu'une variable aléatoire continue X prenne une valeur *exacte* est nulle. Par conséquent, on ne définit la probabilité que sur des **intervalles**. Cette probabilité est calculée à l'aide d'une **fonction de densité de probabilité** (ou PDF, de l'anglais *Probability Density Function*), notée $p(x)$. La probabilité que la variable X se trouve dans un intervalle entre a et b est donnée par l'intégrale de cette fonction sur l'intervalle :

$$P(a < x < b) = \int_a^b p(x)dx \tag{A.8}$$

On peut visualiser la probabilité comme l'aire sous une courbe ; l'aire correspondant à un seul point (une ligne infiniment fine) est nulle. Pour être une fonction de densité valide, son intégrale sur l'ensemble doit être égale à 1 :

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{A.9}$$

Exemple 2. Distribution Normale (Gaussienne)

*L'exemple le plus connu de fonction de densité de probabilité est la **distribution normale** ou **gaussienne**. Elle est définie par :*

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{A.10}$$

*Cette distribution est entièrement caractérisée par ses deux paramètres : sa **moyenne** μ, qui représente le centre de la distribution, et son **écart type** σ, qui contrôle sa dispersion (son étalement).*

A.2.3 Espérance

Le concept de l'espérance, noté $\mathbb{E}[x]$, représente la valeur moyenne d'une variable aléatoire, pondérée par sa probabilité d'occurrence. Les définitions exactes pour les cas discret et continu sont :

$$\mathbb{E}[x] = \sum_{i}^{N} p_i x_i \quad \text{(cas discret)} \quad \text{et} \quad \mathbb{E}[x] = \int_{-\infty}^{\infty} x\, p(x) dx \quad \text{(cas continu)} \tag{A.11}$$

L'espérance est un opérateur linéaire et possède plusieurs propriétés utiles :

$$\mathbb{E}[x+y] = \mathbb{E}[x] + \mathbb{E}[y] \tag{A.12}$$

$$\mathbb{E}[ax] = a\ \mathbb{E}[x] \quad \text{où } a \text{ est une constante} \tag{A.13}$$

$$\mathbb{E}[xy] \neq \mathbb{E}[x]\,\mathbb{E}[y] \quad \text{en général, sauf si les variables } x \text{ et } y \text{ sont indépendante)} \tag{A.14}$$

A.2.4 Probabilité jointe et conditionnelle

La **probabilité conditionnelle** est la probabilité qu'un événement se produise sachant qu'un autre événement a déjà eu lieu. On la note :

$$P(B = b | A = a) \tag{A.15}$$

Cette expression se lit : "la probabilité que la variable B prenne la valeur b, *sachant que* la variable A a pris la valeur a". Par ailleurs, la **probabilité jointe** est la probabilité que deux événements se produisent simultanément, et se note :

$$P(B = b, A = a) \tag{A.16}$$

Dans ces notations, la barre verticale signifie "sachant que" et la virgule signifie "et". Ces deux types de probabilités sont liés par la règle du produit, qui permet de calculer une probabilité jointe à partir d'une probabilité conditionnelle :

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A) \tag{A.17}$$

A.2.5 Loi de Bayes

En réarrangeant la règle du produit de l'équation (A.17), on obtient la célèbre **loi de Bayes** :

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{A.18}$$

Cette relation est centrale pour les problèmes de décision avec observations partielles, car elle permet de mettre à jour une croyance à la lumière de nouvelles informations. Son utilité fondamentale est d'inverser la probabilité conditionnelle. Typiquement, on cherche à connaître la probabilité d'une **cause** (ex : l'état d'un système, B) sachant qu'on a observé un **symptôme** (ex : une mesure, A). Le théorème de Bayes nous permet de calculer $P(B|A)$ en utilisant $P(A|B)$, qui est souvent plus facile à modéliser.

Exemple 3. Exemple Conceptuel : Nuages et Pluie

Considérons deux événements liés à la météo :
- ***Événement N :*** *Le ciel est nuageux.*
- ***Événement P :*** *Il pleut.*

La ***probabilité jointe****, $P(N, P)$, est la probabilité que les deux événements se produisent en même temps (un temps nuageux* ***et*** *pluvieux). La* ***probabilité conditionnelle****, $P(P|N)$, est la probabilité qu'il pleuve* ***sachant que*** *nous avons déjà constaté que le ciel est nuageux. La* ***loi de Bayes*** *nous permet d'inverser cette relation pour répondre à une question différente. Supposons qu'il soit facile d'estimer $P(P|N)$ (la probabilité de l'effet "pluie" sachant la cause "nuages"), mais que nous souhaitions connaître $P(N|P)$ (la probabilité de la cause "nuages" sachant que l'on observe l'effet "pluie").*
La loi de Bayes formalise ce raisonnement inversé :

$$P(Cause \mid Effet) = \frac{P(Effet \mid Cause) \times P(Cause)}{P(Effet)}$$

Appliquée à notre exemple :

$$P(N|P) = \frac{P(P|N) \times P(N)}{P(P)}$$

Traduit en mots : *La probabilité que le ciel soit nuageux (cause) sachant qu'il pleut (effet observé) est égale à la probabilité qu'il pleuve un jour nuageux, multipliée par la probabilité habituelle d'avoir des nuages, le tout divisé par la probabilité habituelle qu'il pleuve. Cela nous permet de mettre à jour notre croyance sur la présence de nuages en se basant sur l'observation de la pluie.*

Annexe B

Approximation de fonctions

Capsule vidéo
Approximation de fonctions
`https://youtu.be/p8BizsV8apQ?si=oOI-NNueGjrwpVgr`

Une approximation de fonction tente d'approximer une fonction complexe $y = f(x)$ ou une relation observé dans des données $y_i = f(x_i)$, avec un certain nombre de paramètres w :

$$f(x) \approx \hat{f}(x|w) \tag{B.1}$$

On cherche généralement les paramètres w qui vont minimiser l'erreur entre la prédiction de la fonction approximée et la vrai relation :

$$w^* = \underset{w}{\operatorname{argmin}}\, \mathcal{L} \tag{B.2}$$

$$\mathcal{L} = \mathbb{E}\left[\frac{1}{2}\left(f(x_i) - \hat{f}(x_i|w)\right)^2\right] \tag{B.3}$$

Ce qui est équivalent sur un certain nombre N d'échantillons, dans le cas d'une relation observé sur des données, à

$$\mathcal{L} = \frac{1}{N}\sum_i^N \frac{1}{2}\left(f(x_i) - \hat{f}(x_i|w)\right)^2 \tag{B.4}$$

Le gradient de la fonction perte de cet objectif est donc :

$$\frac{\partial \mathcal{L}}{\partial w} = -\frac{1}{N}\sum_i^N \left(f(x_i) - \hat{f}(x_i|w)\right)\frac{\partial \hat{f}}{\partial w} \tag{B.5}$$

Descente du gradient stochastique La méthode de la descente du gradient stochastique consiste à perte à jour les paramètres de l'approximation, basé le gradient calculé un échantillon à la fois, avec un paramètre de taux d'apprentissage η :

$$w \Leftarrow w + \eta\left(f(x_i) - \hat{f}(x_i|w)\right)\frac{\partial \hat{f}}{\partial w} \tag{B.6}$$

B.1 Approximation linéaires

Une catégorie importante d'approximations de fonction sont construit comme la sommation pondéré de noyau qui peuvent être non-linéaire :

$$\hat{f}(x|w) = \sum_i w_i \phi_i(x) = \underline{w}^T \underline{\phi}(x) \tag{B.7}$$

Si les noyaux non-linéaires sont fixés d'avance, et que les seuls paramètres de l'approximation sont les poids w_i, alors ce type de fonction a des avantages intéressants : **1)** on peut trouver les poids optimaux de façon explique avec la méthode des moindres-carrés et **2)** on peut avoir des garanties de convergence pour plusieurs méthodes en ligne.

Gradient

Le gradient de la fonction perte avec ce type de fonction est de :

$$\frac{\partial \mathcal{L}}{\partial \underline{w}} = -\frac{1}{N} \sum_i^N \left(f(x_i) - \underline{w}^T \underline{\phi}(x) \right) \underline{\phi}^T(x) \tag{B.8}$$

lorsque calculé pour un ensemble de N échantillons, et de :

$$\frac{\partial \mathcal{L}_i}{\partial \underline{w}} = -\left(f(x_i) - \underline{w}^T \underline{\phi}(x) \right) \underline{\phi}^T(x) \tag{B.9}$$

et pour un seul échantillon.

Solution des moindres-carrés

À venir !

B.1.1 Polynômes

$$\hat{f}(x|w) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \dots \tag{B.10}$$

B.1.2 Fourrier

$$\hat{f}(x|w) = \sum_i w_i cos(i\pi x) \tag{B.11}$$

B.1.3 Base radiales

$$\hat{f}(x|w) = \sum_i w_i e^{-(\frac{x-\mu_i}{2\sigma_i^2})^2} \tag{B.12}$$

B.2 Approximations non-linéaires

Voir réseaux de neurones....

Annexe C

Exercises

C.1 Introduction et formulation du problème

C.1.1 Fonction coût vs récompense

Considérons un problème dont l'objectif est de minimiser la fonction de coût instantané suivante :

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

Cette fonction représente une erreur de positionnement au carré ($(x_d - x)^2$) et une pénalité quadratique sur l'énergie utilisée (u^2).
On souhaite cependant utiliser un algorithme qui est programmé pour *maximiser* une fonction de récompense.

1. Proposez une fonction de récompense $r(x, u)$ qui permettrait à l'algorithme de résoudre le problème de minimisation de coût original.
2. Prouvez formellement que la politique optimale qui minimise le coût cumulatif $\sum g(x, u)$ est la même que celle qui maximise la récompense cumulative $\sum r(x, u)$ que vous avez proposée.

C.1.2 Limites de la formulation coût/récompense cumulative

Cet exercice a pour but de réfléchir de manière critique à la généralité de la formulation de coût cumulatif. La structure d'un coût ou d'une récompense additive est-elle limitative ? Autrement dit, existe-t-il des objectifs qui ne pourraient pas être formulés sous cette forme ?
Analysez les deux situations spécifiques suivantes :

1. **Objectif multiplicatif :** Considérons un problème où les gains sont multiplicatifs à chaque étape (par exemple, un jeu où le gain de l'étape k est multiplié par celui de l'étape $k + 1$). Un tel objectif peut-il être transformé pour s'inscrire dans le cadre d'une fonction cumulative (additive) ? Si oui, comment ?
2. **Objectif multi-critères :** Considérons un problème multi-objectif, où l'on cherche à optimiser simultanément plusieurs critères, parfois contradictoires (ex : minimiser le temps de trajet *et* la consommation d'énergie). Est-il toujours possible de représenter un tel problème avec une seule fonction de coût scalaire ? Quelles sont les approches possibles et leurs limites ?

C.1.3 Formes possibles des politiques optimales

Pour chacun des cas suivants, cet exercice demande une réflexion intuitive (sans calculs formels) sur la nature de la politique optimale.

1. **Politique dépendant du temps :** Décrivez un scénario où la politique optimale doit nécessairement dépendre du temps (ou de l'étape k). Fournissez un exemple concret et un contre-exemple où une politique optimale serait stationnaire (indépendante du temps).
2. **Politique stochastique :** Décrivez un scénario où la politique optimale devrait être stochastique, c'est-à-dire qu'elle choisirait les actions selon une distribution de probabilité plutôt que de manière déterministe. Fournissez un exemple concret.

C.1.4 Apprendre à voler

Explorez l'exemple de code suivant qui utilise l'algorithme PPO pour entraîner un drone à voler.

Exercice de code
Planar Drone with PPO
`https://colab.research.google.com/drive/1-EuRPiyf2GvOn8p_17jlvvEIE188WT1m?usp=sharing`

1. Combien d'étapes d'entraînement sont nécessaires pour apprendre une politique de vol adéquate ?
2. Vérifiez si la politique apprise se généralise bien en la testant avec différentes conditions initiales.
3. Observez la politique (loi de commande) qui est affichée à la fin de l'entraînement. Comment l'interprétez-vous ? Ressemble-t-elle à une loi de commande que vous connaissez ?
4. Modifiez les paramètres de la fonction de coût pour obtenir un comportement de vol plus "agressif" (par exemple, en pénalisant moins l'utilisation de l'énergie). Qu'observez-vous ?

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 cet exercice, le code d'amorce est disponible à la page Colab suivante :

Exercice de code
Fonction coût pour un pendule
`https://colab.research.google.com/drive/1BzMh7bBNgflchsbgWkDFkgP2s-5HNqW9?usp=sharing`

Tâches à réaliser

Pour chacune des situations décrites ci-dessous, exécutez le code et procédez à l'analyse demandée dans la section suivante.

a) **Situation de référence :** Exécutez le code avec la fonction de coût quadratique par défaut.
b) **Pénalité sur la position :** Ajustez les valeurs dans la matrice Q pour pénaliser plus fortement l'erreur en position du pendule.
c) **Temps minimal :** Modifiez les fonctions $g(x,u,t)$ et $h(x,t)$ pour obtenir une solution qui minimise le temps de stabilisation.
d) **[Optionnel] Exploration libre :** Testez et explorez d'autres variantes de fonction de coût.

Analyse demandée

Pour **chaque tâche** effectuée, analysez et interprétez les trois éléments suivants. Notez les changements par rapport à la situation de référence et tentez d'expliquer leurs causes.

1. La figure du **coût-à-venir (J^*)** : elle représente le coût minimal encouru à partir de n'importe quel état initial.
2. La **loi de commande** générée : la surface montrant le couple appliqué en fonction de l'angle et de la vitesse.
3. La **trajectoire simulée** : le comportement du système partant de la position basse.

Note : Pour plusieurs raisons, l'algorithme peut avoir de la difficulté à converger pour certaines fonctions de coût. Essayez des changements mineurs si c'est le cas. Le but ici n'est pas de vous faire 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

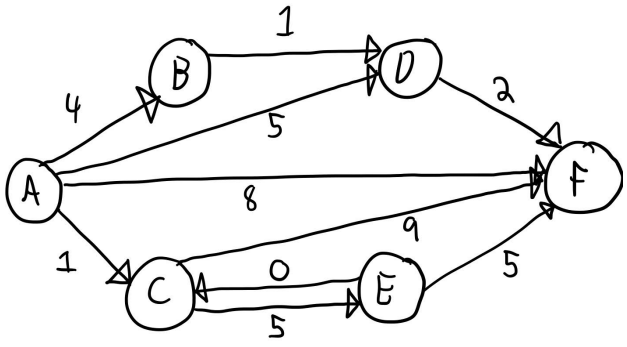


FIGURE C.1 – Graphique qui représente des chemins possibles pour aller vers la position F

Appliquez l'algorithme de programmation dynamique exacte :

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

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

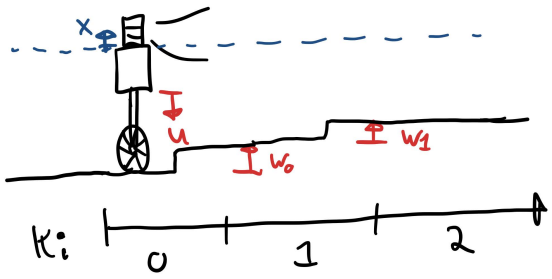
pour résoudre les actions optimales (choix de l'arc à suivre) pour se rendre à l'état cible (Noeud F) à partir de l'état actuel (les Noeuds $x_k \in [A, B, C, D, E]$) et de l'index de temps actuel k. Le coût de chaque option de chemin est représenté par le chiffre indiqué pour chaque arc sur le graphique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



Considérons un robot équipé d'une suspension active qui se déplace sur un terrain accidenté. L'objectif est de calculer une loi de commande optimale sur un horizon de **deux pas de temps** ($N = 2$). À chaque pas de temps k, la suspension peut être ajustée d'une hauteur u_k. L'état du système, x_k, représente l'écart de hauteur du robot par rapport à une trajectoire désirée. L'évolution de cet état est donnée par :

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

$$x_2 = x_1 + u_1 + w_1 \tag{C.5}$$

où les variables w_k sont des variations de hauteur du terrain. Dans cette première version du problème (déterministe), on suppose que les variations de hauteur du terrain w_k sont connues à l'avance, par exemple grâce à un balayage LIDAR du chemin.
L'objectif est double : on cherche à la fois à minimiser l'erreur de hauteur finale du robot (à $N = 2$) et à pénaliser les variations de hauteur brusques à chaque instant, car elles pourraient perturber les instruments. La fonction de coût à optimiser pour refléter cet objectif est définie comme suit :

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

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

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

On cherche à minimiser la somme de ces coûts additifs :

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

Utilisez l'algorithme de programmation dynamique exacte pour calculer les lois de commande optimales $\pi_0^*(x_0, w_0)$ et $\pi_1^*(x_1, w_1)$. Notez bien que, puisque les perturbations w_k sont connues à l'avance, la politique optimale à chaque étape sera une fonction de l'état actuel x_k et de la perturbation connue w_k.
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) :

$$\text{Dynamique :} \quad x_1 = ax_0 + u_0 \tag{C.10}$$
$$x_2 = ax_1 + u_1 \tag{C.11}$$

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 :

$$\text{Coûts :} \quad g_0(x_0, u_0) = r_0 u_0^2 \tag{C.12}$$
$$g_1(x_1, u_1) = r_1 u_1^2 \tag{C.13}$$
$$g_2(x_2) = q_2(x_2 - x_d)^2 \tag{C.14}$$

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 le 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 :

$$\text{Politiques} \quad u_0 = \pi_0(x_0) \tag{C.15}$$
$$u_1 = \pi_1(x_1) \tag{C.16}$$

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

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

Calculez le coût-à-venir et la politique optimale à l'étape $k = 0$ et à l'étape $k = 1$. Le tableau suivant peux 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

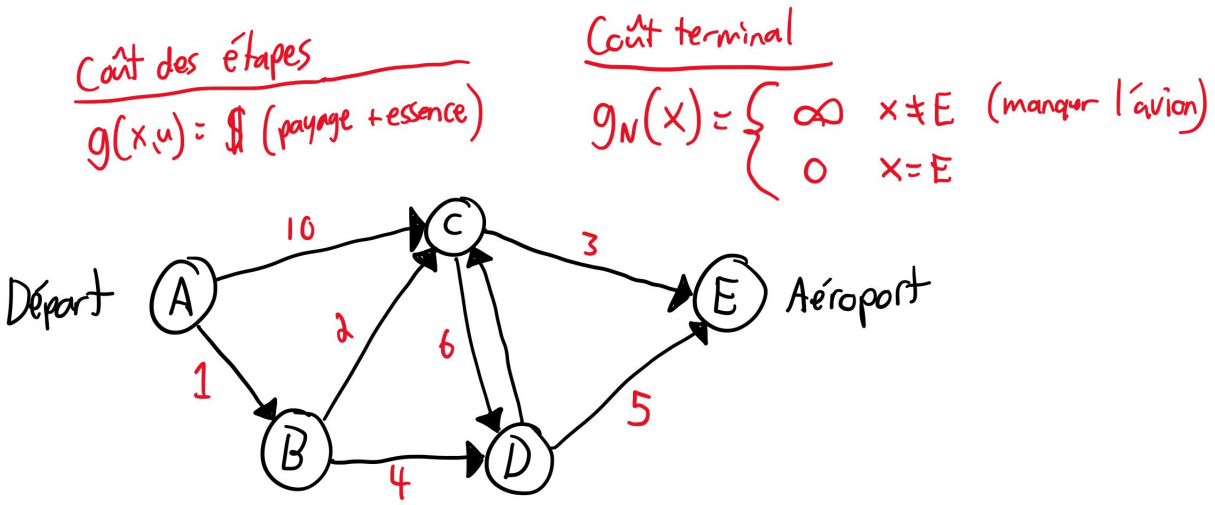


FIGURE C.2 – Problème d'optimisation d'un trajet vers l'aéroport

Appliquez l'algorithme de programmation dynamique exacte :

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

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

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 d'espérance d'une variable aléatoire.

Cet exercice reprend le problème de la suspension active (exercice C.2.2), mais dans un contexte stochastique. On considère maintenant que les irrégularités du terrain w_k sont des perturbations aléatoires inconnues à l'avance. La distribution de probabilité des perturbations est connue et donnée par :

$$P(w_0 = 1) = 0.5 \tag{C.20}$$
$$P(w_0 = 0) = 0.5 \tag{C.21}$$
$$P(w_1 = 1) = 0.5 \tag{C.22}$$
$$P(w_1 = 0) = 0.5 \tag{C.23}$$

Votre tâche est de recalculer la loi de commande optimale $\{\pi_0^*(x_0), \pi_1^*(x_1)\}$. L'objectif est de minimiser l'**espérance** du coût-à-venir total :

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

en utilisant la récursion de la programmation dynamique stochastique :

$$J_k^*(x_k) = \min_{u_k} \mathop{\mathbb{E}}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\right] \tag{C.25}$$
$$\pi_k^*(x_k) = \operatorname*{argmin}_{u_k} \mathop{\mathbb{E}}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\right] \tag{C.26}$$

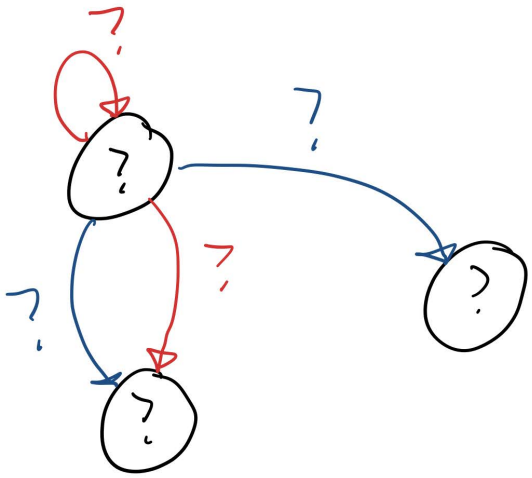
Le tableau suivant peut vous aider pour synthétiser les résultats :

$k =$	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 sous la forme d'un Processus de Décision Markovien (MDP).
— Application de la programmation dynamique stochastique à horizon fini.



Une diva est en résidence dans votre casino pour effectuer des spectacles tous les soirs, et il reste N représentations prévues 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 est de lui acheter un cadeau coûtant x dollars, une stratégie qui fonctionne avec une probabilité de p_c. Lorsque la stratégie du cadeau échoue, ou lorsqu'on ne lui offre pas de cadeau, la diva insatisfaite refuse de chanter le soir suivant et reste insatisfaite. Un spectacle annulé représente une perte de y dollars pour le casino. On considère qu'on peut offrir un cadeau à la diva pour tenter de la convaincre de chanter une fois par jour.

1) Définissez le problème sous la structure d'un Processus de Décision Markovien (MDP) :

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

2) Calculez le coût-à-venir 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'échecs et vous voulez optimiser votre choix de stratégie à chaque partie pour maximiser vos chances de gagner la série. La règle de bris d'égalité est la suivante : si le pointage est nul après deux parties, des parties supplémentaires sont jouées et le premier joueur à remporter une victoire gagne la série.
Vous avez le choix entre deux stratégies dont vous connaissez les performances :

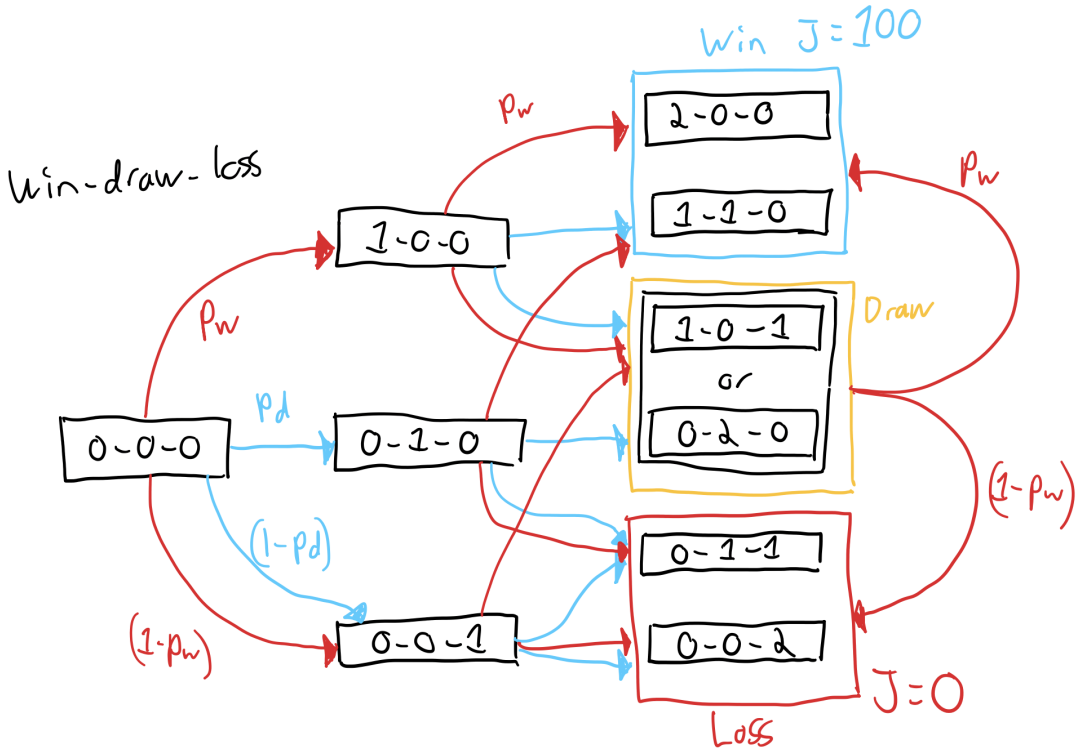
— **Stratégie Défensive :** Mène à une partie nulle avec une probabilité p_d et à une défaite avec une probabilité $1 - p_d$. Elle ne mène jamais à une victoire.
— **Stratégie Agressive :** Mène à une victoire avec une probabilité p_w et à une défaite avec une probabilité $1 - p_w$. Elle ne mène jamais à une partie nulle.

Avec $p_d = 0.99$ et $p_w = 0.49$, déterminez la probabilité de gagner la série pour chacune des politiques suivantes :

1. Toujours utiliser la stratégie agressive.
2. Toujours utiliser la stratégie défensive.
3. Utiliser la stratégie agressive d'abord, puis la stratégie défensive pour la deuxième partie.
4. Choisir la stratégie optimale en fonction de l'état (le pointage actuel de la série).

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

Finalement, répondez à la question conceptuelle suivante : est-il possible d'avoir plus de 50% de chances de gagner la série, même si la probabilité de gagner une seule partie avec votre meilleure stratégie de victoire (p_w) est inférieure à 50% ?

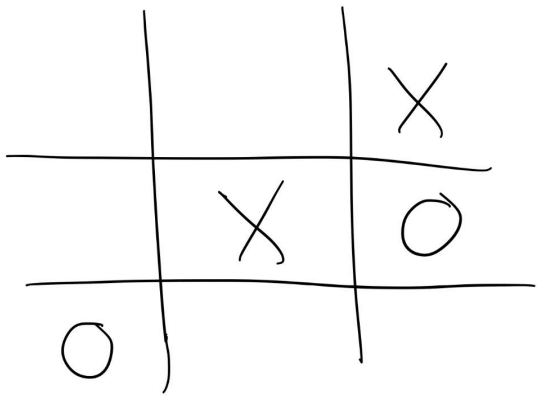


C.4 Commande robuste

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.

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

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 :

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

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 :

$$\mathbb{E}\left[\underline{w}_k\right] = \underline{0} \tag{C.29}$$

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

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

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

$$Q_k \geq 0 \qquad R_k > 0 \tag{C.31}$$

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

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

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 dépent pas de l'état actuel.
2) la loi de commande optimale a la forme linéaire suivante :

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

où K_k est la matrice de gain égale à

$$K_k = \left[R_k + B_k^T S_{k+1} B_k\right]^{-1} B_k^T S_{k+1} A_k \tag{C.34}$$

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 :

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

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 $1 \times n$;

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

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 :

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

Ensuite, basée sur l'hypothèse que la solution est

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

1. Calculez

$$\frac{\partial J^*}{\partial \underline{x}} = ? \tag{C.39}$$

2. Calculez

$$\frac{\partial J^*}{\partial t} = ? \tag{C.40}$$

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_f6l`

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`

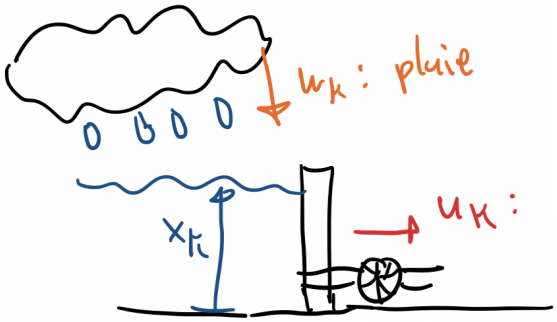
1. Analysez le code source qui implémente les solutions LQR
2. Exécutez le code de l'exemple et analysez le comportement de la politique LQR pour un horizon de temps infini
3. Simulez la trajectoire pour divers conditions initiales
4. Exécutez le code de l'exemple et analysez le comportement de la politique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

$$x_{k+1} = \min\left[x_k - u_k + w_k, 4\right] \tag{C.41}$$

où

$$x_k \in \{0, 1, 2, 3, 4\} \tag{C.42}$$
$$u_k \in \{0, 1, 2\} \tag{C.43}$$
$$u_k \leq x_k \tag{C.44}$$
$$w_k = \begin{cases} 0 & P = 0.4 \\ 1 & P = 0.4 \\ 3 & P = 0.2 \end{cases} \tag{C.45}$$

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 dépendent des précipitations aléatoires avec les probabilités données.
On cherche à minimiser les pertes financières donnés par :

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

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 :

$$J = \lim_{N \to \infty} \mathbb{E}\left[\sum_{k=0}^{N} \alpha^k g_k\right] \tag{C.47}$$

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

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

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/1P2sjOo9T31-6rkTBHsK3ZFaGMntvBDzn?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.

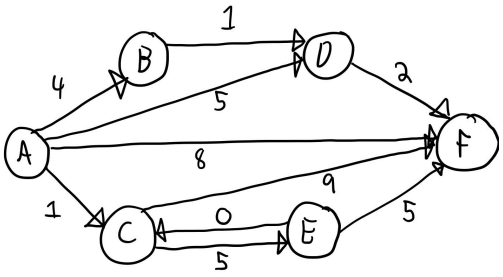
1. Comparez le coût-à-venir de cette politique avec la solution optimale (voir exercice C.6.2)
2. Comparez le coût-à-venir de plusieurs états initiaux, avec le coût d'une trajectoire simulée qui débute sur ces états initiaux.
3. Concevez une politique plus performante et validez numériquement avec l'algorithme.

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 effectués. 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) :

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

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-à-venir $J(x)$ 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 :

$$x_{k+1} = 0.5x_k + u_k \tag{C.50}$$

$$g_k = x_k^2 + u_k^2 \tag{C.51}$$

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-à-venir optimale (donc indirectement la loi de commande optimale) avec seulement les données en utilisant l'approximation polynomiale d'ordre 2 suivante :

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

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

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

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 environs. 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-à-venir 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 substitué 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-à-venir d'une étape J_k et la suivante J_{k+1} :

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

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

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

C.7.4 *Q-Learning* avec échantillonnage

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

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

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

$$Q^* = \mathbb{E}\left[q_i\right] = \frac{1}{N} \sum_{i=1}^{N} q_i \tag{C.57}$$

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

$$\text{Loss function :} \quad \mathcal{L} = \mathbb{E}\left[(y - \hat{y}(\theta))^2\right] = \frac{1}{N} \left[\sum_i \underbrace{(y_i - \hat{y}_i(\theta))^2}_{e_i^2} \right] \tag{C.58}$$

$$\text{Learning law :} \quad \theta \Leftarrow \theta - \eta \frac{\partial e_i^2}{\partial \theta} \tag{C.59}$$

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

$$\text{Loss function :} \quad \mathcal{L} = \mathbb{E}\left[(q_i - \hat{Q})^2\right] = \frac{1}{N} \sum_i (q_i - \hat{Q})^2 \tag{C.60}$$

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

$$\hat{Q} = \theta \tag{C.61}$$

Déterminez la loi d'apprentissage.

C.7.5 Descente du gradient stochastique

L'exemple de code suivant met en oeuvre un algorithme simple de descente du gradient stochastique :

Exercice de code
Descente du gradient stochastique
`https://colab.research.google.com/drive/1t2psC4eMHrm_cuX8Rs0cWqOXfg4xATqO?usp=sharing`

1) Analyser le code source pour comprendre l'algorithme.
2) Implémentez un taux d'apprentissage variable pour améliorer la performance de l'algorithme.
3) Vérifiez si la performance est influencé si on fait varier le nombre d'échantillon et le niveau de bruit.

C.7.6 Approximation d'un coût-à-venir

L'exemple de code suivant illustre divers divers méthode d'approximation de fonction appliquées à approximer un cout-à-venir optimal :

Exercice de code
Approximation d'un coût-à-venir
`https://colab.research.google.com/drive/1PwQXyqWxWhPN3HSQBBXVroTGOGWaRVRO?usp=sharing`

Dans cet exemple, on trouve la solution par programmation dynamique sans approximation et ensuite on utilise le résultat pour approximer la fonction cout-à-venir par apprentissage supervisé.
1) Analyser le code source pour comprendre les fonctions d'approximations utilisée.
2) Augmenter graduellement la résolution de l'approximation (multiples gaussiennes) pour obtenir une bonne approximation de la fonction coût-à-venir.
3) Analyser l'approximation quadratique et le coût-à-venir de la solution LQR ? Vérifiez si l'approximation est meilleur si on analyse le problème pour un plus petit domaine proche de la cible.

C.8 Apprentissage par renforcement appliqué

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`

C.8.2 Tutorial pour la librairie *Gymnasium*

Exercice de code
Introduction à l'environnement Gymnasium
`https://drive.google.com/file/d/1xGLNuhxn2PN8hRSeTYpsCzsvdVGZSy3D/view?usp=sharing`

C.8.3 Environnement à partir d'une définition de problème

1) Identifiez un problème sur lequel vous voulez travailler pour votre projet de session, et décrivez le en quelques lignes de texte.

Note : Si votre projet n'est pas encore défini, essayez de faire un choix préliminaire, ce n'est pas grave si le sujet change par la suite, mais c'est un plus pour prendre de l'avance.

---

2) Écrivez le pseudo-code d'une classe représentant un environnement *gymnasium*, qui défini le problème. Plus précisément les points importants sont (pour les détails voir `https://gymnasium.farama.org/introduction/create_custom_env/`) :
Espace observations et actions Quelle est la nature des variables (discrètes vs continues), dimensions, bornes (min/max). Il faut définir l'ensemble des possibles actions et observations. C'est un attribut nécesssaire pour la classe gym : `https://gymnasium.farama.org/api/spaces/`.
Fonction *step(u)* Quelle est la nature des états (variables mémoires internes dans la classe), comment les états sont mis à jours, est-ce que l'évolution est stochastique ou déterministe, calcul des observations, calcul de la récompense, est-ce qu'il y a un état terminal ? etc.. Je vous suggère de séparer les étapes dans cette fonction en sous-étapes :

$$x_{next} = f(x, u) \tag{C.62}$$
$$reward = -g(x, u) \tag{C.63}$$
$$observations = h(x_{next}, u) \tag{C.64}$$
$$terminated = t(x_{next}, u) \tag{C.65}$$

Fonction *reset* Ici vous devez déterminez la distribution de l'état initial. Est-ce que l'état initial est toujours la même valeur dans votre problème, ou bien est-ce qu'il y a plusieurs conditions initial possibles ?

Exemple Voir au lien suivant un exemple d'environnement gym basé sur les définitions standards pour un système dynamique : `https://github.com/SherbyRobotics/pyro/blob/master/pyro/tools/sys2gym.py`

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.