
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 4 août 2024

Préface

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



Capsule vidéo

Série de capsules vidéos associées

<https://youtube.com/playlist?list=PL6adNeJOA8UtNs1NQfAHAzcjHcQixBSnu>

Sources externes utiles :

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

Table des matières

1	Programmation dynamique	4
1.1	Introduction	4
1.2	La prise de décision en séquence	4
1.3	Formulation du problème	6
1.3.1	Coût-à-venir	7
1.3.2	Politique optimale	7
1.3.3	Terminologie	8
1.4	Principe d'optimalité	9
1.5	Programmation dynamique exacte	9
1.6	Variations sur un thème de Bellman	12
1.7	Programmation dynamique approximée	13
2	Commande stochastique	14
2.1	Dynamique stochastique	14
2.2	Optimisation de l'espérance	14
2.3	Formulation minimax (commande robuste)	14
2.4	Observations Partielles	14
3	Modèles d'évolution	15
3.1	Équations de différence	15
3.2	Équations différentielles (temps continus)	15
3.3	Graphes (États discrets)	15
3.4	Chaînes de Markov (états discrets avec évolution stochastique)	15
4	Équation de Bellman	16
4.1	Horizon de temps infini	16
4.2	Équation de Bellman	16
4.3	Équation de Hamilton–Jacobi–Bellman	16
5	Solutions Analytiques	17
5.1	LQR à temps discret	17
6	Algorithmes	18
6.1	Algorithme d'itération de valeurs (<i>Value-iteration</i>)	18
6.2	Algorithme d'itération de loi de commande (<i>policy-iteration</i>)	18
6.3	TD-Learning	18
6.4	Q-Learning	18
6.5	Sarsa	18

A Outils mathématiques	19
A.1 Probabilités	19
A.1.1 Espérance	19
A.2 Opérations	19
A.2.1 Minimum/maximum	19
B Exercices	21
B.1 Programmation dynamique	21
B.1.1 Fonction coût vs récompense	21
B.1.2 Politique fonction du temps	21
B.1.3 Politique stochastique	21
B.1.4 Fonction de coût pour un pendule	21
B.1.5 Navigation optimale dans un graphe	23
B.1.6 Loi de commande pour une suspension active	24
B.1.7 Loi de commande pour une suspension active II	25
B.1.8 Commande stochastique pour une diva à l'opéra	26
B.1.9 Commande minimax pour tic-tac-toe	27
B.1.10 Solution LQR par programmation dynamique	28
B.1.11 Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur	30
B.1.12 <i>Q-learning</i> pour une navigation optimale	32
B.1.13 Apprentissage d'une fonction $Q(x, u)$ approximée	33
B.1.14 Génération d'une base de données	33

Chapitre 1

Programmation dynamique

1.1 Introduction

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



Capsule vidéo

Introduction

<https://youtu.be/1ThW0UnkVyY?si=wW1a0-0YpvR-vYbL>

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 un horizon de temps fini. On pourrait considérer que c'est un petit détour, plusieurs autres ouvrages présente 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 introduit ainsi, et de plus, avec cette formulation le liens est plus direct à faire avec les concepts de la science des asservissement. On va donc utiliser une représentation mathématique basée sur les équations de différence, et introduire l'aspect probabiliste dans un deuxième temps.

1.2 La prise de décision en séquence

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

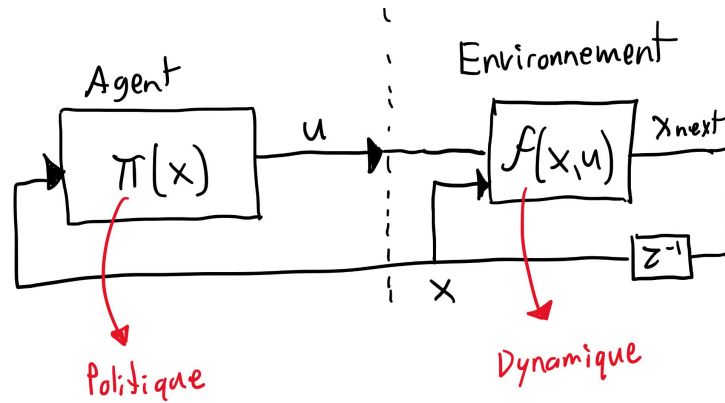


FIGURE 1.1 – Dynamique en boucle fermée avec un agent

Un comportement défini par une politique à concevoir

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

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

où on note u la variable qui représente l'action et x la variable qui représente l'état de l'environnement que l'agent observe. Dans un contexte d'asservissement le terme utilisé pour la fonction π serait la *loi de commande*. La forme de la fonction π peut aller d'une équation analytique, d'un réseau de neurone à un tableau de données en mémoire (look-up table), etc.

Un objectif formulé avec une fonction scalaire cumulative

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

$$J = \sum g(x, u) \quad (1.2)$$

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

Exemple 1. Loi de commande pour un robot

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

Exemple 2. Navigation d'un véhicule

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

Exemple 3. Achats d'une action

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

1.3 Formulation du problème

La formulation mathématique de base, qui sera utilisée pour introduire les principes et les différentes approches de solution, est une approche en temps discret ou on va considérer qu'on veut optimiser une politique pour un nombre fini N d'étapes futurs. De plus, on va débiter avec des équations déterministes et l'hypothèse que l'agent observe directement l'état de l'environnement. On traitera plus tard les situations où l'horizon de temps est infini (Chapitre 4), l'évolution est stochastique (Chapitre 2) et quand l'agent a accès à une observation partielle ou bruitée de l'état (Chapitre 2.4). Donc voici les ingrédients :

Environnement/Dynamique/Évolution

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

$$x_{k+1} = f_k(x_k, u_k) \quad k = 0, 1, \dots, N-1 \quad (1.3)$$

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

Politique/Loi de commande

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

$$u_k = \pi_k(x_k) \quad (1.4)$$

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

$$\pi = \{\pi_0, \dots, \pi_{N-1}\} \quad (1.5)$$

Fonction objectif (coût ou récompense)

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

$$J(\underbrace{x_0, \dots, x_N, u_0, \dots, u_{N-1}}_{\text{Trajectoire}}) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N) \quad (1.6)$$

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

Contraintes

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

$$x_k \in X_k \quad (1.7)$$

et un ensemble U_k d'action possible, qui peut dépendre de l'état actuel.

$$u_k \in U_k(x_k) \quad (1.8)$$

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

$$\pi \in \Pi \quad (1.9)$$

1.3.1 Coût-à-venir

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

$$J_\pi(x) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N) \quad \text{avec} \quad x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k)) \quad (1.10)$$

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

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

1.3.2 Politique optimale

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

$$J^*(x) = \min_{\pi \in \Pi} J_\pi(x) \quad (1.11)$$

$$\pi^* = \arg \min_{\pi \in \Pi} J_\pi(x) \quad (1.12)$$

$$(1.13)$$

1.3.3 Terminologie

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

Exemple 4. Pendule Simple en temps minimum

Le concept de coût-à-venir se 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 1.2, représente une temps-à-venir optimal calculé numériquement.

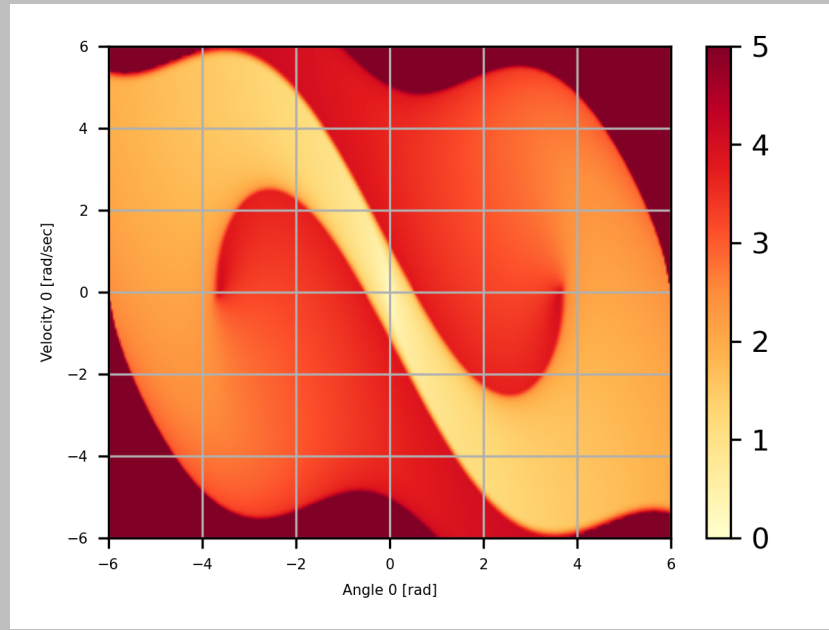


FIGURE 1.2 – Coût-à-venir optimal pour un pendule (temps-minimum)

1.4 Principe d'optimalité

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

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

$$[x_0, \dots, x_i, \dots, x_N] \quad (1.14)$$

$$[x_i, \dots, x_N] \quad (1.15)$$



Capsule vidéo

Le principe d'optimalité

<https://youtu.be/EMkpkYMTg4U?si=eDITgpq50NhrD8-f>

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

1.5 Programmation dynamique exacte

L'algorithme de programmation dynamique permet de résoudre exactement le problème de décision en séquence formulé à la section 1.3. L'idée est de partir de la fin, et de calculer de le coût-à-venir en reculant

étape par étape :

$$J_N^*(x_N) = g_N(x_N) \quad \forall x_N \in X_N \quad (1.16)$$

$$\vdots \quad (1.17)$$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \quad \forall x_k \in X_k \quad (1.18)$$

$$\vdots \quad (1.19)$$

$$J_0^*(x_0) = \min_{u_0 \in U_0(x_0)} \left[g_0(x_0, u_0) + J_1^*(\underbrace{f_0(x_0, u_0)}_{x_1}) \right] \quad \forall x_0 \in X_0 \quad (1.20)$$

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

$$\pi_k^*(x_k) = \underset{u_k \in U_k(x_k)}{\operatorname{argmin}} \left[g_k(x_k, u_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \quad \forall x_k \in X_k \quad (1.21)$$

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

$$\underbrace{J_k^*(x_k)}_{\text{Coût-à-venir optimal}} = \min_{\substack{u_k \in U_k(x_k) \\ \text{Minimum possible}}} \left[\underbrace{g_k(x_k, u_k)}_{\text{Coût instantané}} + \underbrace{J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}})}_{\text{Coût-à-venir optimal à l'étape suivante}} \right] \quad (1.22)$$



Capsule vidéo

Algorithme de programmation dynamique

<https://youtu.be/dfz9k3BGrH0?si=MBxJzpKPOUjnaERe>

Exemple 5. Navigation optimale



Capsule vidéo

Exemple de navigation optimale sur un graphe

<https://youtu.be/1GXUNWVgZOU?si=P4hDvWSWoav6nW4x>

Exemple 6. Chauffage optimale



Capsule vidéo

Exemple pour une politique de chauffage optimale

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

1.6 Variations sur un thème de Bellman

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

Stochastique

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

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \right) \right] \quad (1.23)$$

Robuste

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

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^* \left(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}} \right) \right] \quad (1.24)$$

À horizon de temps infini

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

$$J^*(x) = \min_u \left[g(x, u) + \alpha J^* \left(\underbrace{f(x, u)}_{x_{k+1}} \right) \right] \quad (1.25)$$

Sans modèles (apprentissage par renforcement)

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

$$Q^*(x, u) = g(x, u) + \min_{u_{k+1}} \left[Q^* \left(\underbrace{f(x, u)}_{x_{k+1}}, u_{k+1} \right) \right] \quad (1.26)$$

À temps continu

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

$$-\frac{\partial J^*(x, t)}{\partial t} = \min_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (1.27)$$

1.7 Programmation dynamique approximée

À venir !

Chapitre 2

Commande stochastique

2.1 Dynamique stochastique

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad (2.1)$$

$$w_k \in W_k(x) \quad (2.2)$$

2.2 Optimisation de l'espérance

On cherche donc ici à minimiser l'espérance du coûts-à-venir,

$$J_\pi = \mathbb{E} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad (2.3)$$

avec l'algorithme de programmation dynamique suivant :

$$J_k^*(x_k) = \min_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + \underbrace{J_{k+1}^*(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (2.4)$$

$$u_k^*(x_k) = \operatorname{argmin}_{u_k} \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + \underbrace{J_{k+1}^*(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (2.5)$$

2.3 Formulation minimax (commande robuste)

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + \underbrace{J_{k+1}^*(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (2.6)$$

$$u_k^*(x_k) = \operatorname{argmin}_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + \underbrace{J_{k+1}^*(f_k(x_k, u_k, w_k))}_{x_{k+1}} \right] \quad (2.7)$$

2.4 Observations Partielles

Chapitre 3

Modèles d'évolution

3.1 Équations de différence

À venir !

3.2 Équations différentielles (temps continu)

À venir !

3.3 Graphes (États discrets)

Lorsqu'on travail avec un problème où les états et actions possibles sont discrets, c'est à dire qu'il peuvent prendre un nombre fini de valeurs. Par exemple, pour une automobile la transmission a généralement autour de 6 options discrètes (marche arrière, neutre, 1ère vitesse, 2e vitesse, etc.) tandis que la commande d'accélération est une variable continue qui peut prendre n'importe quel valeur à l'intérieur d'une certaine plage. Pour les systèmes ou tout est discret, il est possible de représenter les états comme des noeuds sur un graphe et les actions possible comme des arcs qui nous mène vers un autre état. Dans ce contexte on peut simplifier la notation.

$$J_k^*(i) = \min_{j \in U(i)} [a_{ij}^k + J_{k+1}^*(j)] \quad (3.1)$$

$$j_k^*(i) = \operatorname{argmin}_{j \in U(i)} [a_{ij}^k + J_{k+1}^*(j)] \quad (3.2)$$

3.4 Chaînes de Markov (états discrets avec évolution stochastique)

À venir !

Chapitre 4

Équation de Bellman

4.1 Horizon de temps infini

$$J = \lim_{N \rightarrow \infty} \mathbb{E} \left[\sum_{k=0}^N \alpha^k g_k \right] \quad (4.1)$$

4.2 Équation de Bellman

À venir !

4.3 Équation de Hamilton–Jacobi–Bellman

$$0 = \min_u \left[g(x, u) + \frac{\partial J^*(x, t)}{\partial x} \underbrace{f(x, u, t)}_{\dot{x}} \right] \quad (4.2)$$

Chapitre 5

Solutions Analytiques

5.1 LQR à temps discret

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

$$\underline{x}_{k+1} = f_k(\underline{x}_k, \underline{u}_k, \underline{w}_k) = A_k \underline{x}_k + B_k \underline{u}_k + \underline{w}_k \quad (5.1)$$

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

$$\mathbb{E}[\underline{w}_k] = 0 \quad (5.2)$$

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

$$J = \mathbb{E} \left[\sum_{k=0}^{N-1} \underbrace{\left(\underline{x}_k^T Q_k \underline{x}_k + \underline{u}_k^T R_k \underline{u}_k \right)}_{g_k(\underline{x}_k, \underline{u}_k, \underline{w}_k)} + \underbrace{\underline{x}_N^T Q_N \underline{x}_N}_{g_N(\underline{x}_N)} \right] \quad (5.3)$$

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

$$Q_k \geq 0 \quad R_k > 0 \quad (5.4)$$

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

$$J_k^*(\underline{x}_k) = \underline{x}_k^T S_k \underline{x}_k + c \quad (5.5)$$

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

$$\underline{u}_k^* = c_k^*(\underline{x}_k) = -K_k \underline{x}_k \quad (5.6)$$

où K_k est la matrice de gain égale à

$$K_k = [R_k + B_k^T S_{k+1} B_k]^{-1} B_k^T S_{k+1} A_k \quad (5.7)$$

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

$$S_k = Q_k + A_k^T \left(S_{k+1} - S_{k+1}^T B_k^T [R_k + B_k^T S_{k+1} B_k]^{-1} B_k S_{k+1} \right) A_k \quad (5.8)$$

Chapitre 6

Algorithmes

- 6.1 Algorithme d'itération de valeurs (*Value-iteration*)
- 6.2 Algorithme d'itération de loi de commande (*policy-iteration*)
- 6.3 TD-Learning
- 6.4 Q-Learning
- 6.5 Sarsa

Annexe A

Outils mathématiques

A.1 Probabilités

Probabilité pour une variable discrète :

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

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

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

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

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

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

$$(\text{A.7})$$

Probabilité pour une variable continue, la probabilité est définie sur une interval :

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

ou $p(x)$ est une fonction de densité de probabilité.

A.1.1 Espérance

$$\mathbb{E}[x] = \sum p_i x_i \quad \text{or} \quad \mathbb{E}[x] = \int x p(x)dx \quad (\text{A.9})$$

$$\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y] \quad (\text{A.10})$$

$$\mathbb{E}[ax] = a \mathbb{E}[x] \quad (\text{A.11})$$

$$\mathbb{E}[xy] \neq \mathbb{E}[x] \mathbb{E}[y] \quad (\text{A.12})$$

A.2 Opérations

A.2.1 Minimum/maximum

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.

Annexe B

Exercices

B.1 Programmation dynamique

B.1.1 Fonction coût vs récompense

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

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

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

B.1.2 Politique fonction du temps

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

B.1.3 Politique stochastique

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

B.1.4 Fonction de coût pour un pendule

Compétences à développer :

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

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



Exercice de code

Fonction coût pour un pendule

https://colab.research.google.com/drive/1vC1sja-aP9SIBskjB-W_5s7WFMaam-ji?usp=sharing

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

Pour chacune des situations suivantes :

a) Situation de référence : exécuter le code avec la fonction coût quadratique par défaut.

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

analysez :

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

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

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

B.1.5 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

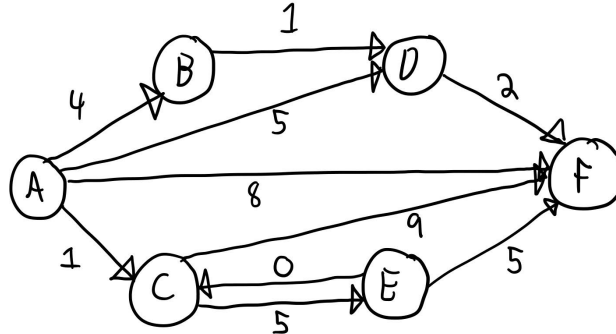


FIGURE B.1 – Graphique qui représente des chemins possibles pour aller vers la position F

Appliquez l'algorithme de programmation dynamique exacte :

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

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

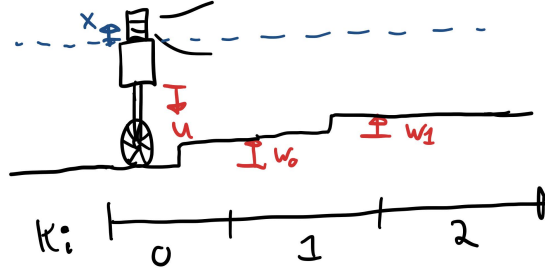
pour résoudre les actions optimales (choix de l'arc à suivre) pour se rendre à l'état cible (Noeud F) à partir de l'état actuel (les Noeuds $x_k \in [A, B, C, D, E]$) et de l'index de temps actuel k . Le coût de chaque option de chemin est représenté par le chiffre indiqué pour chaque arc sur le graphique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

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

B.1.6 Loi de commande pour une suspension active

Compétences à développer :

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



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

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

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

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

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

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

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

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

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

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

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

B.1.7 Loi de commande pour une suspension active II

Compétences à développer :

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

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

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

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

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

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

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

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

avec l'algorithme de programmation dynamique suivant :

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

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

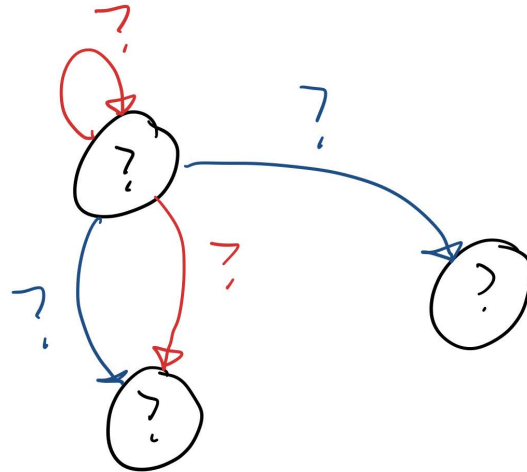
Le tableau suivant peut vous aider pour synthétiser les résultats :

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

B.1.8 Commande stochastique pour une diva à l'opéra

Compétences à développer :

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



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

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

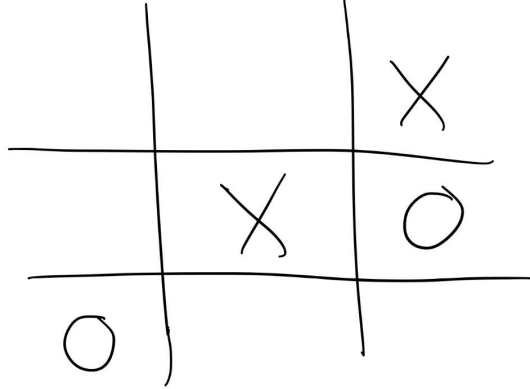
1. Déterminez les états possibles
2. Déterminez les actions possibles
3. Illustrez le processus graphiquement avec des noeuds (états) et les transitions possibles
4. Déterminez les probabilités de transitions
5. Déterminez les coûts associés aux transitions.

2) Calculer le coût-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 $N = 4$.

B.1.9 Commande minimax pour tic-tac-toe

Compétences à développer :

— Algorithme minimax



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

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

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

B.1.10 Solution LQR par programmation dynamique

Compétences à développer :

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

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

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

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

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

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

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

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

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

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

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

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

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

où K_k est la matrice de gain égale à

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

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

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

Notes sur la dérivation d'un scalaire par un vecteur

Lorsqu'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 $1 \times n$;

$$\underline{z} = \frac{\partial y}{\partial \underline{x}} = \left[\frac{\partial y}{\partial x_1} \quad \dots \quad \frac{\partial y}{\partial x_n} \right] \Leftrightarrow z_i = \frac{\partial y}{\partial x_i} \quad (\text{B.26})$$

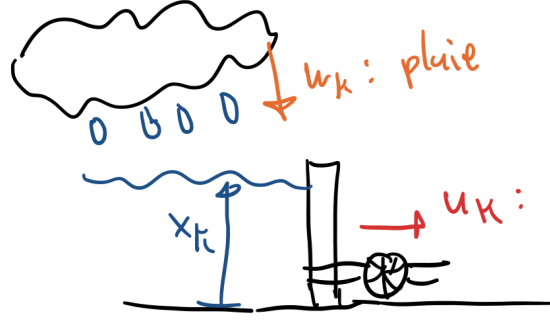
TABLE B.1 – Identités pour la dérivation d'une fonction scalaire par un vecteur

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}

B.1.11 Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur

Compétences à développer :

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



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

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

où

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

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

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

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

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

On cherche à minimiser les pertes financières donnés par :

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

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

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

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

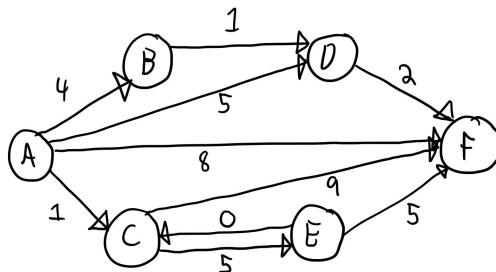
$$u_k^* = c^*(x_k) = \begin{cases} ? & \text{si } x_k = 1 \\ ? & \text{si } x_k = 2 \\ ? & \text{si } x_k = 3 \\ ? & \text{si } x_k = 4 \end{cases} \quad (\text{B.34})$$

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

B.1.12 *Q-learning* pour une navigation optimale

Compétences à développer :

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



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

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

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

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

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

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

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

B.1.13 Apprentissage d'une fonction $Q(x, u)$ approximée

Compétences à développer :

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

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

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

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

B.1.14 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 :

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

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

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

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

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

Comparaison avec la solution analytique LQR

Le système ci-dessus a une dynamique linéaire et un coût quadratique. Calculez le coût-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 substituer la matrice S par la solution de l'équation de Riccati algébrique. Ici toutes les matrices sont 1x1 donc des scalaires.
