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 5 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 a établir les liens entre les approches issues du domaine de l'ingénierie (la science des asservissements et la commande optimale) et les approches issues des sciences informatiques (recherche opérationnelle et l'apprentissage par renforcement) qui ont en fait les même bases mathématiques. Ces notes visent principalement à donner à un lecteur issue du domaine de l'ingénierie les bases pour comprendre et utiliser les approches numériques issues des sciences informatiques.

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

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 . . . 8
  - 1.3.3 Terminologie . . . 9
- 1.4 Principe d'optimalité . . . 10
- 1.5 Programmation dynamique excate . . . 10
- 1.6 Variations sur un thème de Bellman . . . 14
- 1.7 Forces et limites de la programmation dynamique . . . 15
  - 1.7.1 Forces . . . 15
  - 1.7.2 Limites . . . 15
- 1.8 Programmation dynamique approximée . . . 16

2 Commande stochastique **17**
- 2.1 Dynamique stochastique . . . 17
- 2.2 Optimisation de l'espérance . . . 18
- 2.3 Formulation minimax (commande robuste) . . . 19
- 2.4 Observations Partielles . . . 19
  - 2.4.1 Espace croyance (draft !) . . . 19

3 Modèles d'évolution **21**
- 3.1 Équations de différence . . . 21
- 3.2 Équations différentielles (temps continus) . . . 21
- 3.3 Graphes (états discrets déterministes) . . . 21
- 3.4 Chaînes de Markov (états discrets stochastiques) . . . 21

4 Équation de Bellman **22**
- 4.1 Horizon de temps infini . . . 22
- 4.2 Équation de Bellman . . . 22
- 4.3 Équation de Hamilton–Jacobi–Bellman . . . 22

5 Solutions Analytiques **23**
- 5.1 LQR à temps discret . . . 23

6 Algorithmes **24**
- 6.1 Algorithme d'itération de valeurs (*Value-iteration*) . . . 24
- 6.2 Algorithme d'itération de loi de commande (*policy-iteration*) . . . 24
- 6.3 TD-Learning . . . 24
- 6.4 Q-Learning . . . 24
- 6.5 Sarsa . . . 24

A Outils mathématiques **25**
A.1 Probabilités . . . 25
A.1.1 Espérance . . . 25
A.2 Opérations . . . 25
A.2.1 Minimum/maximum . . . 25
A.3 Bayes . . . 25

B Exercises **28**
B.1 Programmation dynamique . . . 28
B.1.1 Fonction coût vs récompense . . . 28
B.1.2 Politique fonction du temps . . . 28
B.1.3 Politique stochastique . . . 28
B.1.4 Fonction de coût pour un pendule . . . 28
B.1.5 Navigation optimale dans un graphe . . . 30
B.1.6 Loi de commande pour une suspension active . . . 31
B.1.7 Loi de commande pour une suspension active II . . . 32
B.1.8 Commande stochastique pour une diva à l'opéra . . . 33
B.1.9 Commande minimax pour tic-tac-toe . . . 34
B.1.10 Solution LQR par programmation dynamique . . . 35
B.1.11 Gestion d'un barrage optimale pour un horizon de temps infini par itération de valeur 37
B.1.12 *Q-learning* pour une navigation optimale . . . 39
B.1.13 Apprentissage d'une fonction $Q(x, u)$ approximée . . . 40
B.1.14 Génération d'une base de données . . . 40

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 offres peut 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 asservis classique, comme contrôler un bras robot en choisissant la tension appliquée aux moteurs basé sur une observation de sa position, que pour des problèmes probabiliste dans un contexte de finance, comme choisir quand acheter ou vendre une action en observant l'évolution de son prix, ou bien un problème d'intelligence artificielle comme choisir la pièce à déplacer lors d'une partie d'échec en observant la position des pièces sur l'échiquier.

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

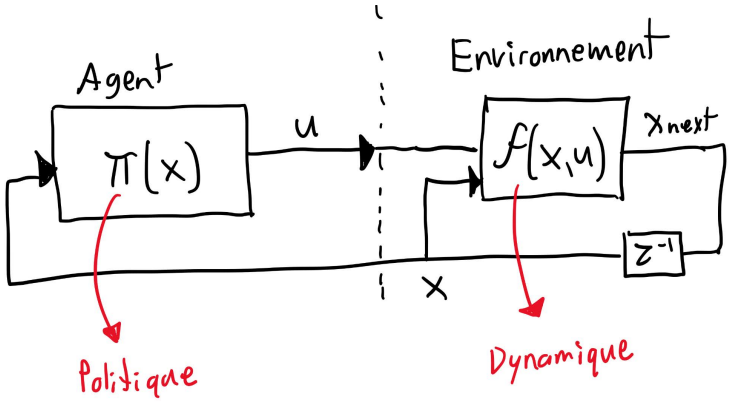


FIGURE 1.1 – Dynamique en boucle fermée avec un agent

Un comportement défini par une politique à concevoir

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

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

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 utilisée. Typiquement, dans le domaine de la commande on formule l'objectif comme minimiser une fonction coût et dans le domaine de l'apprentissage par renforcement, on formule l'objectif comme maximiser une fonction récompense. Les deux formulations sont équivalentes et interchangeables (il suffit de changer le signe). Je vais dans ces notes utiliser par défaut la formulation d'une fonction coût qu'on désire minimiser. La fonction sera noté :

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

où J est le coût cumulatif à minimiser et $g(x,u)$ est un coût instantané pour une étape. La forme cumulative de la fonction coût est centrale pour utiliser le principe de la programmation dynamique, mais ce n'est pas vraiment restrictif comme définition, tous les problèmes peuvent être reformulé sous cette forme. Lorsque que notre agent prend les 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.

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

Exemple 1. Loi de commande pour un robot

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

Exemple 2. Navigation d'un véhicule

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

Exemple 3. Achats d'une action

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

1.3 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 veux optimiser une politique pour un nombre fini N d'étapes futurs. De plus, on va débuter avec des équations déterministes et l'hypothèse que l'agent observe directement l'état de l'environnement. On traitera plus tard les situations ou 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 bruité 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 :

$$x_{k+1} = f_k(x_k, u_k) \qquad k = 0, 1, \ldots, N-1 \tag{1.3}$$

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

$$u_k = \pi_k(x_k) \tag{1.4}$$

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

$$\pi = \{\pi_0, \ldots, \pi_{N-1}\} \tag{1.5}$$

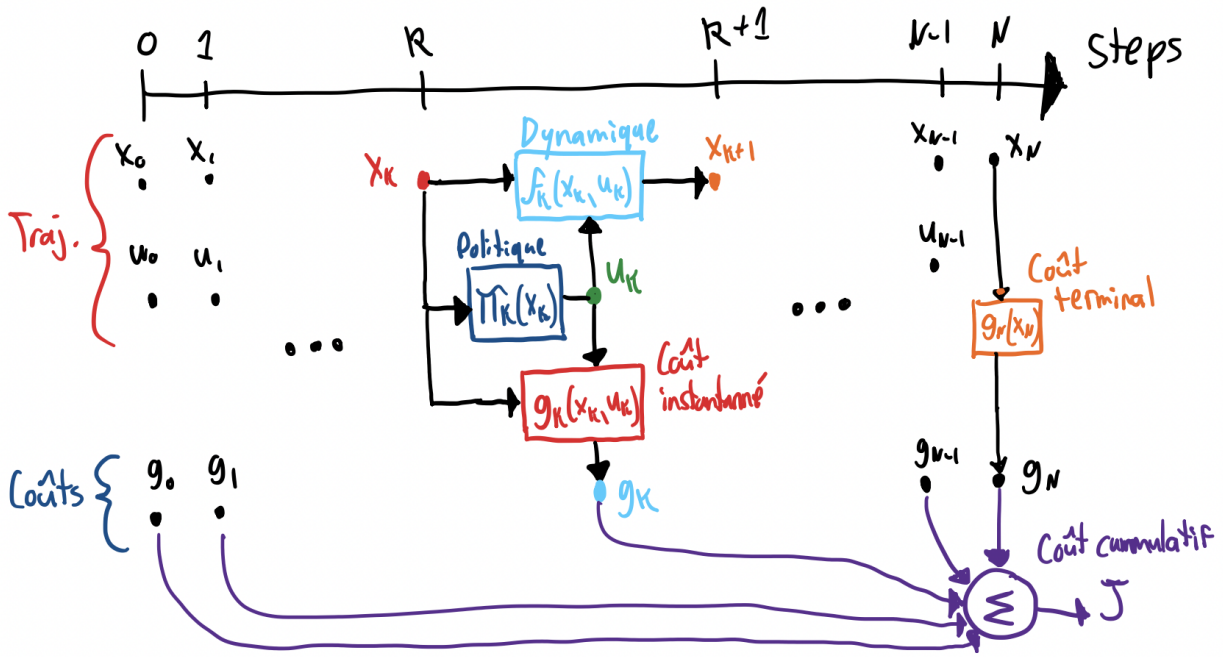


FIGURE 1.2 – Évolution du système en boucle fermée étape par étape.

Fonction objectif (coût ou récompense)

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

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

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

$$x_k \in X_k \qquad u_k \in U_k(x_k) \tag{1.7}$$

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

$$\pi \in \Pi \tag{1.8}$$

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 débuter à l'état x et exécuter la politique π sur un horizon N :

$$J_\pi(x) = \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N) \quad \text{avec} \quad x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k)) \tag{1.9}$$

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

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

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.

$$J^*(x) = \min_{\pi \in \Pi} J_\pi(x) \tag{1.10}$$

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

$$\tag{1.12}$$

1.3.3 Terminologie

Variable	Termes	Définition
$f_k(x_k, u_k)$	Dynamique, Système, Environnement, Processus, *Plant*,	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, *policy*	Fonction qui définit la décision de l'agent comme une fonction de l'observation de l'état.
x	État, *State*	Variable qui représente toute l'information pour prédire l'évolution future du système.
u	Action, décision, entrée du système, *control input*	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, *control set*	Ensemble représentant les actions qui sont possible lorsque l'état du système est x à l'étape k
$J(x_0, ..., x_N)$	Fonction de coût cumulative, (opposé de la) fonction de récompense, *value fonction*	Fonction scalaire qui représente à quel point une trajectoire est bonne en fonction de l'objectif.
$J_\pi(x)$	Coût à venir, *cost-to-go*	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 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 1.3, représente une temps-à-venir optimal calculé numériquement.

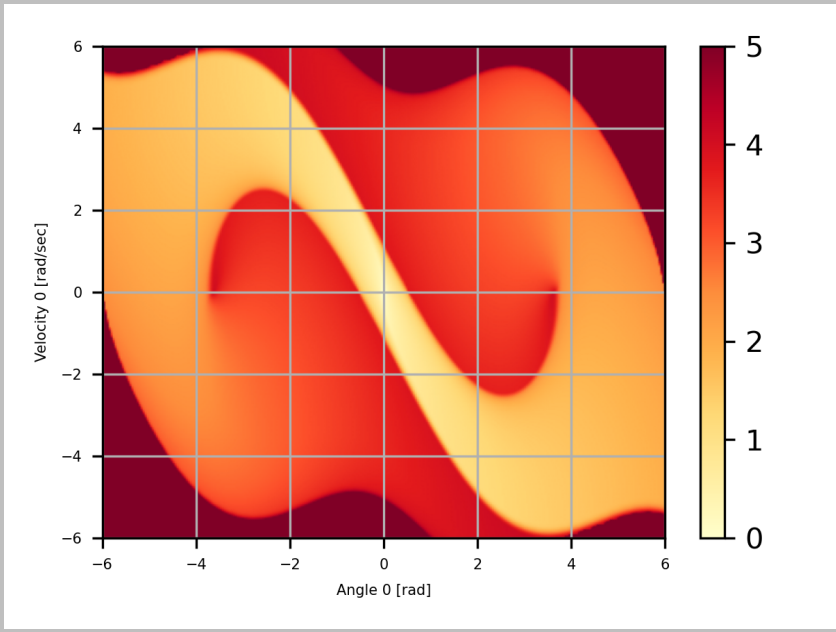


FIGURE 1.3 – Coût-à-venir optimal pour un pendule (temps-minimum)

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 débuterait à x_i pour aller à x_N.

$$[x_0, ..., x_i, ..., x_N] \tag{1.13}$$
$$[x_i, ..., x_N] \tag{1.14}$$

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

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 :

$$J_N^*(x_N) = g_N(x_N) \qquad \forall x_N \in X_N \tag{1.15}$$

$$\vdots \tag{1.16}$$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \qquad \forall x_k \in X_k \tag{1.17}$$

$$\vdots \tag{1.18}$$

$$J_0^*(x_0) = \min_{u_0 \in U_0(x_0)} \left[g_0(x_0, u_0) + J_1^*(\underbrace{f_0(x_0, u_0)}_{x_1}) \right] \qquad \forall x_0 \in X_0 \tag{1.19}$$

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

$$\pi_k^*(x_k) = \operatorname*{argmin}_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}}) \right] \qquad \forall x_k \in X_k \tag{1.20}$$

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 :

$$\underbrace{J_k^*(x_k)}_{\text{Coût-à-venir optimal}} = \underbrace{\min_{u_k \in U_k(x_k)}}_{\text{Minimum possible}} \underbrace{\left[\underbrace{g_k(x_k, u_k)}_{\text{Coût instantané}} + \underbrace{J_{k+1}^*(\underbrace{f_k(x_k, u_k)}_{x_{k+1}})}_{\text{Coût-à-venir optimal à l'étape suivante}} \right]}_{\text{Valeur } Q_k^*(x_k, u_k)} \tag{1.21}$$

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

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

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

Exemple 5. Navigation optimale

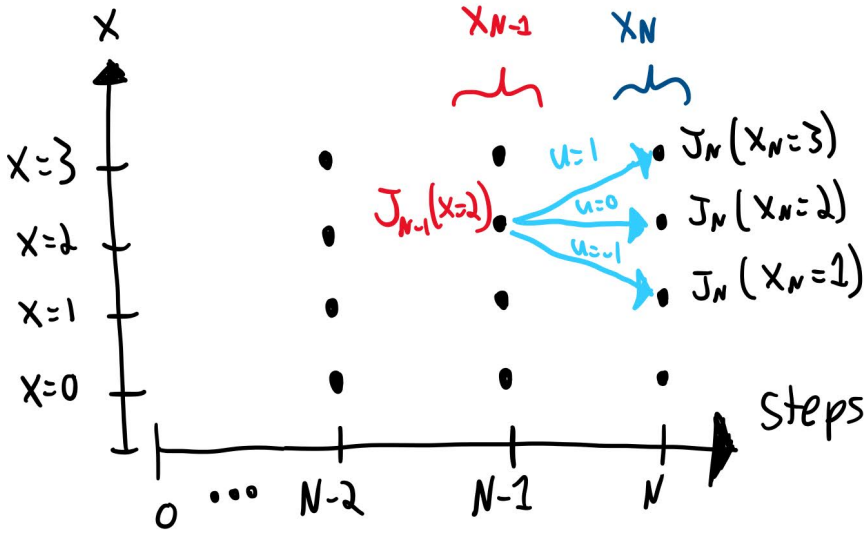


FIGURE 1.4 – Exemple de calcul de coût-à-venir pour l'état $x = 2$ à l'étape $k = N - 1$. Ici pour cet exemple on a quatres états discrets possibles et trois actions possibles. Comme illustré, on doit dabord avoir évalué le coût terminal de tout les états à l'étape $k = N$, ensuite on calcul de cout-à-venir de chaque action possible et on minimise.

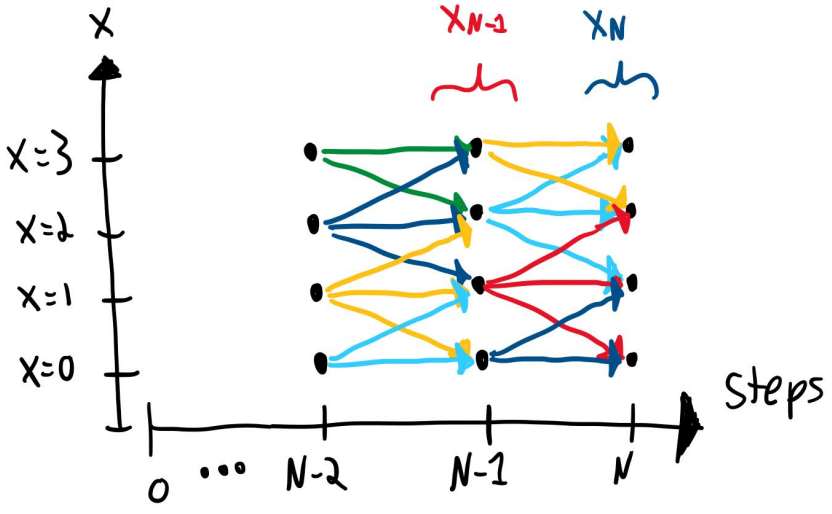


FIGURE 1.5 – L'algorithme de programmation dynamique, consiste en executant ce calcul pour chaque état à chaque étape en débutant par la fin. Ici chaque couleur représente une étape de minimisation pour un état. Il y a une valeur $Q(x, u)$ pour chaque état-action qui sont ici les arcs, et une valeurs $J(x)$ pour chaque état qui sont les noeuds.

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 pondéré des coûts futur possible :

$$J_k^*(x_k) = \min_{u_k} \underset{w_k}{\mathbb{E}} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{1.22}$$

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 :

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

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 :

$$J^*(x) = \min_u \left[g(x, u) + \alpha J^*(\underbrace{f(x, u)}_{x_{k+1}}) \right] \tag{1.24}$$

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 :

$$Q^*(x, u) = g(x, u) + \min_{u_{k+1}} \left[Q^*(\underbrace{f(x, u)}_{x_{k+1}}, u_{k+1}) \right] \tag{1.25}$$

À temps continu

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

$$-\frac{\partial J^*(x,t)}{\partial t} = \min_u \left[g(x, u) + \frac{\partial J^*(x,t)}{\partial x} \underbrace{f(x, u, t))}_{\dot{x}} \right] \tag{1.26}$$

1.7 Forces et limites de la programmation dynamique

1.7.1 Forces

Formulation

Non-linéarités

Contraintes

1.7.2 Limites

Malédiction des dimensions

1.8 Programmation dynamique approximée

À venir !

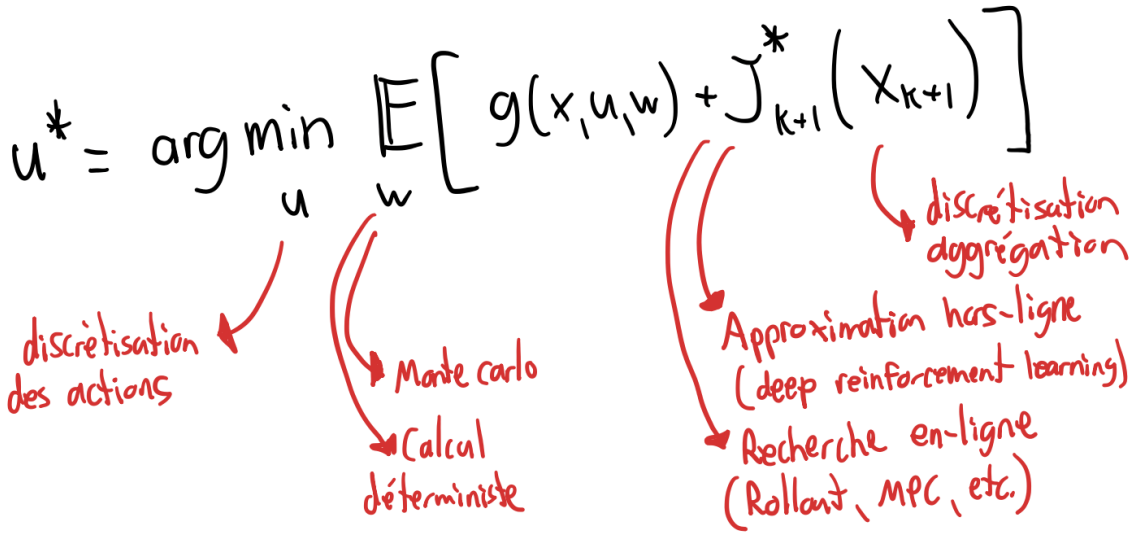


FIGURE 1.6 – Différentes stratégies pour approximer la programmation dynamique exacte.

Chapitre 2

Commande stochastique

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

2.1 Dynamique stochastique

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

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

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

$$w_k \in W_k(x) \tag{2.2}$$

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

$$P(w_k|x_k, u_k, k) \qquad \text{ou} \qquad P(a < w_k < b|x_k, u_k, k) = \int_a^b p(w_k|x_k, u_k, k)dw \tag{2.3}$$

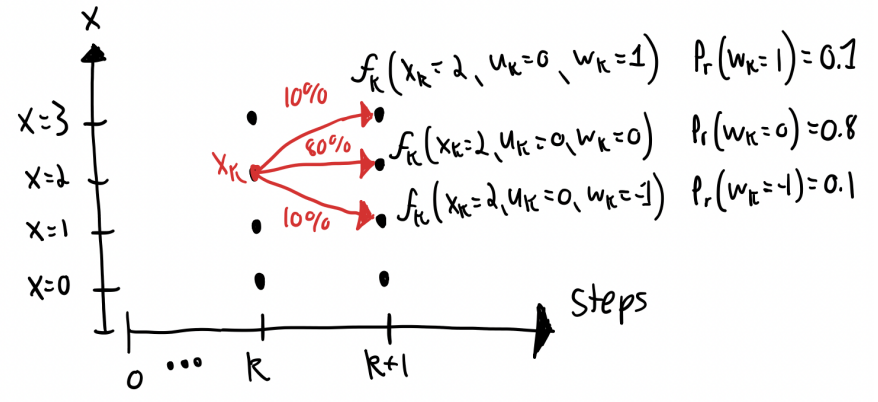


FIGURE 2.1 – Évolution stochastique avec des domaines discrets

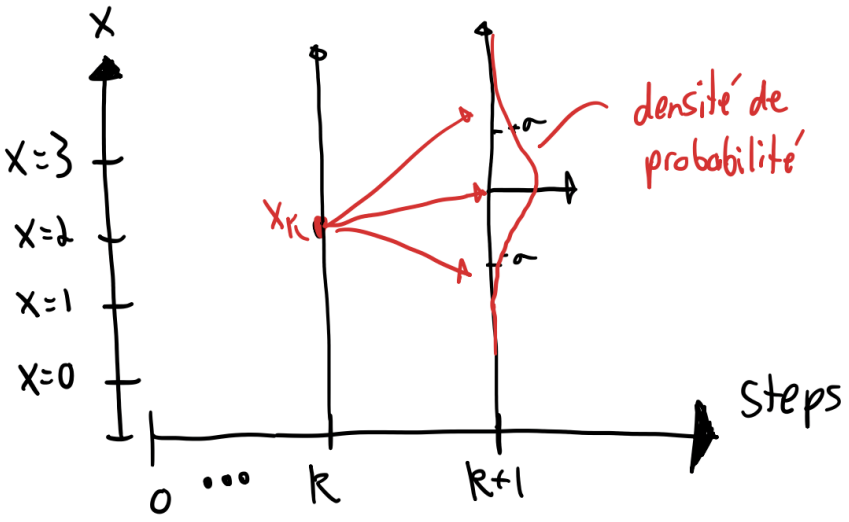


FIGURE 2.2 – Évolution stochastique avec des domaines continus

Ce modèle d'évolution stochastique est une représentation possible de ce qu'on appelle une *Chaîne de Markov*. En effet, il est possible de convertir les équations (2.1) et (2.3) en probabilités de transitions, la représentation habituelle d'une chaîne de Markov :

$$P(x_{k+1}|x_k, u_k, k) \tag{2.4}$$

puisque :

$$P(x_{k+1} = f_k(x_k, u_k, w_k)|x_k, u_k, k) = P(w_k|x_k, u_k, k) \tag{2.5}$$

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

$$P(x_{k+1}|x_k, u_k, k) = P(x_{k+1}|x_k, u_k, k, \underbrace{x_{k-1}, u_{k-1}, x_{k-2}, u_{k-2}, \ldots, x_1, u_1, x_0, u_0}_{\text{historique}}) \tag{2.6}$$

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

2.2 Optimisation de l'espérance

La formulation la plus standard pour un problème de décision séquentielles dans un contexte stochastique (aussi appeler dans la littérature *MDP* pour processus de décision de Markov) est de chercher à minimiser l'espérance du coûts-à-venir. On modifie donc la définition de l'équation (1.9), pour rajouter un calcul de l'espérance par rapport à toutes les valeurs possibles que peuvent prendre les perturbations :

$$J_\pi(x) = \mathop{\mathbb{E}}_{w_0, w_1, \ldots, w_{N-1}} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad \text{avec} \quad x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k), w_k) \tag{2.7}$$

Ce qui représente une moyenne des coût-à-venir possible, pondérés en fonction de leurs probabilités d'occurrence. Le problème à résoudre devient alors de trouver la politique qui minimise cette expression de l'équation (2.10). Par chance, on peut réutiliser l'approche récursive de programmation dynamique qui débute par la fin, il suffit de modifier l'algorithme de programmation dynamique exacte pour ajouté un calcul d'espérance sur le coût-à-venir de chaque option d'action :

$$J_k^*(x_k) = \min_{u_k} \mathop{\mathbb{E}}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.8}$$

$$u_k^*(x_k) = \operatorname*{argmin}_{u_k} \mathop{\mathbb{E}}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.9}$$

2.3 Formulation minimax (commande robuste)

Une autre formulation possible, associé aux concepts de commande robuste, est de plutôt vouloir minimiser le pire scénario possible, et minimiser

$$J_\pi(x) = \max_{w_0, w_1, \ldots, w_{N-1}} \left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \quad \text{avec} \quad x_0 = x \quad \text{et} \quad x_{k+1} = f_k(x_k, \pi_k(x_k), w_k) \tag{2.10}$$

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

$$J_k^*(x_k) = \min_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.11}$$

$$u_k^*(x_k) = \operatorname*{argmin}_{u_k} \max_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}^*(\underbrace{f_k(x_k, u_k, w_k)}_{x_{k+1}}) \right] \tag{2.12}$$

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

2.4 Observations Partielles

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

$$y_k = h_k(x_k, u_k, v_k) \tag{2.13}$$

où v_k est une variable aléatoire représentant du bruit, voir Figure 2.3. Le problème est appelé dans la littérature *POMDP*, pour processus de décision markovien partiellement observable, lorsque les états sont discrets. Dans la littérature sur la commande, on nommes souvent les observations les sorties du système.

2.4.1 Espace croyance (draft !)

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

$$Pr(x_k = x_i | u_k, y_k, \ldots, u_1, y_1, u_0, y_0) \tag{2.14}$$

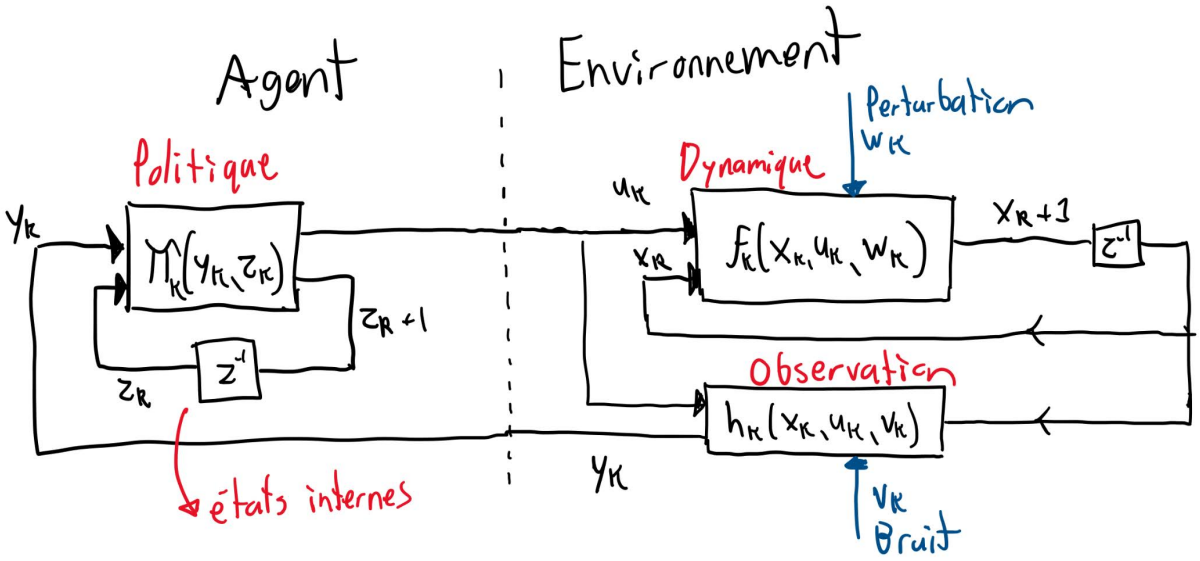


FIGURE 2.3 – Formulation mathématique incluant une fonction observation (utile pour représenter quand l'agent n'observe pas les états directement) et politique qui contient ses propres états et sa dynamique.

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

$$P(x_{k+1}|b_k, u_k, k) = P(x_{k+1}|b_k, u_k, k, \underbrace{y_{k-1}, u_{k-1}, y_{k-2}, u_{k-2}, \ldots, y_1, u_1, y_0, u_0}_{\text{historique}}) \tag{2.15}$$

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

$$u_k = \pi_k^*(b_k) \tag{2.16}$$

Il est toutefois à noter, le processus de mettre à jour l'espace croyance en fonction de nouvelles observations est un processus dynamique qui doit être mis en oeuvre par l'agent. Jusqu'à maintenant on assumait pouvoir observer directement l'état de l'environnement et la politique optimale était toujours une fonction statique entre les observations et les actions. Avec des observations partielles, ce n'est plus le cas, la politique optimale va avoir sa propre dynamique et ses états internes, qui est associée avec le processus de mise à jour de la distribution de probabilité de l'espace de croyance.
La mise à jour de l'espace de croyance, peut être vue théoriquement comme implémenter la règle de Bayes (voir Section A.3). Je dit théoriquement car en pratique c'est un problème très dur généralement insoluble exactement même sur des problème très simples. Il y a toutefois des outils pour des situations particulières, par exemple le filtre de Kalman est une solution à ce problème lorsque le système est linéaire et avec du bruit gaussien. Donc théoriquement la mise à jour de la croyance basée sur des nouvelles observations est :

$$P(x_{k+1}|y_k) = \frac{P(y_k|x_{k+1})P(x_{k+1})}{P(y_k)} \tag{2.17}$$

Détails à venir !

Chapitre 3

Modèles d'évolution

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

3.1 Équations de différence

À venir !

3.2 Équations différentielles (temps continus)

À venir !

3.3 Graphes (états discrets déterministes)

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 :

$$x_k \Leftrightarrow \ i \text{ index du noeud de départ} \tag{3.1}$$
$$u_k \Leftrightarrow \ j \text{ index de la destination} \tag{3.2}$$
$$x_{k+1} \Leftrightarrow \ j \text{ index du noeud d'arrivé} \tag{3.3}$$
$$g_k(x_k, u_k) \Leftrightarrow a_{ij}^k \text{ longueur de l'arc } ij \tag{3.4}$$

$$J_k^*(i) = \min_{j \in U(i)} \left[a_{ij}^k + J_{k+1}^*(j) \right] \tag{3.5}$$
$$\pi_k^*(i) = \operatorname*{argmin}_{j \in U(i)} \left[a_{ij}^k + J_{k+1}^*(j) \right] \tag{3.6}$$

3.4 Chaînes de Markov (états discrets stochastiques)

À venir !

Chapitre 4

Équation de Bellman

4.1 Horizon de temps infini

$$J = \lim_{N \to \infty} \mathbb{E}\left[\sum_{k=0}^{N} \alpha^k g_k\right] \tag{4.1}$$

4.2 Équation de Bellman

À venir !

4.3 Équation de Hamilton–Jacobi–Bellman

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

Chapitre 5

Solutions Analytiques

5.1 LQR à temps discret

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

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

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

$$\mathbb{E}\left[\underline{w}_k\right] = \underline{0} \tag{5.2}$$

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

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

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

$$Q_k \geq 0 \qquad R_k > 0 \tag{5.4}$$

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

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

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

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

où K_k est la matrice de gain égale à

$$K_k = \left[R_k + B_k^T S_{k+1} B_k\right]^{-1} B_k S_{k+1} A_k \tag{5.7}$$

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 :

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

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 :

$$P(x = 1) = 1/6 \tag{A.1}$$
$$P(x = 2) = 1/6 \tag{A.2}$$
$$P(x = 3) = 1/6 \tag{A.3}$$
$$P(x = 4) = 1/6 \tag{A.4}$$
$$P(x = 5) = 1/6 \tag{A.5}$$
$$P(x = 6) = 1/6 \tag{A.6}$$
$$\tag{A.7}$$

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

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

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

A.1.1 Espérance

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

$$\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y] \tag{A.10}$$
$$\mathbb{E}[ax] = a\,\mathbb{E}[x] \tag{A.11}$$
$$\mathbb{E}[xy] \neq \mathbb{E}[x]\,\mathbb{E}[y] \tag{A.12}$$

A.2 Opérations

A.2.1 Minimum/maximum

A.3 Bayes

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{A.13}$$

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A) \tag{A.14}$$

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

Exercises

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 :

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

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

B.1.2 Politique fonction du temps

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

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 donner 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 utilisez 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

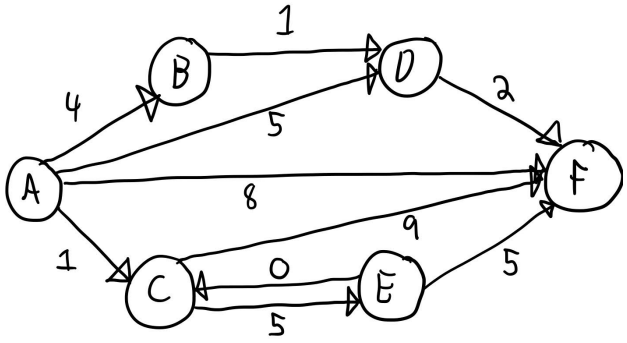


FIGURE B.1 – Graphique qui représente des chemins possibles pour aller vers la position F

Appliquez l'algorithme de programmation dynamique exacte :

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

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

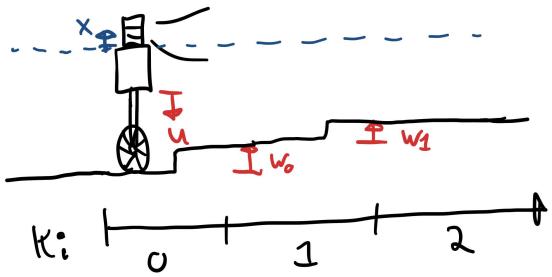
pour résoudre les actions optimales (choix de l'arc à suivre) pour se rendre à l'état cible (Noeud F) à partir de l'état actuel (les Noeuds $x_k \in [A, B, C, D, E]$) et de l'index de temps actuel k. Le coût de chaque option de chemin est représenté par le chiffre indiqué pour chaque arc sur le graphique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

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

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

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 :

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

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

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

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

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

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 d'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 :

$$P(w_0 = 1) = 0.5 \tag{B.10}$$
$$P(w_0 = 0) = 0.5 \tag{B.11}$$
$$P(w_1 = 1) = 0.5 \tag{B.12}$$
$$P(w_1 = 0) = 0.5 \tag{B.13}$$

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 :

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

avec l'algorithme de programmation dynamique suivant :

$$J_k^*(\underline{x}_k) = \min_{\underline{u}_k} \mathop{\mathrm{E}}_{\underline{w}_k}\left[g_k(\underline{x}_k, \underline{u}_k) + J_{k+1}^*(\underbrace{f_k(\underline{x}_k, \underline{u}_k)}_{\underline{x}_{k+1}})\right] \tag{B.15}$$

$$u_k^*(\underline{x}_k) = \operatorname*{argmin}_{\underline{u}_k} \mathop{\mathrm{E}}_{\underline{w}_k}\left[g_k(\underline{x}_k, \underline{u}_k) + J_{k+1}^*(\underbrace{f_k(\underline{x}_k, \underline{u}_k)}_{\underline{x}_{k+1}})\right] \tag{B.16}$$

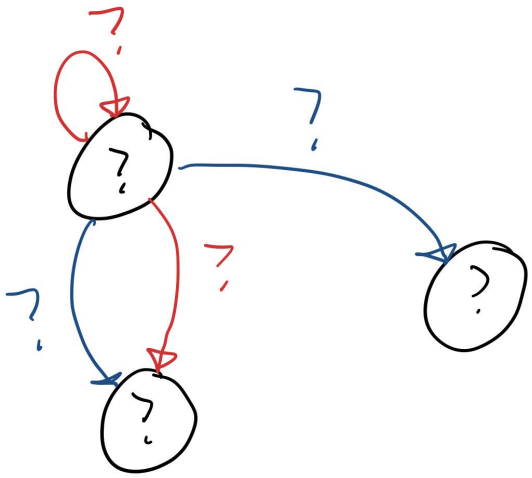
Le tableau suivant peut vous aider pour synthétiser les résultats :

$k =$	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 :

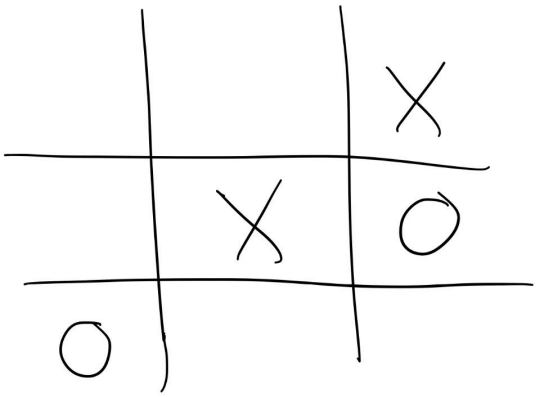
1. Déterminez les états possibles
2. Déterminez les actions possibles
3. Illustrez le processus graphiquement avec des noeuds (états) et les transitions possibles
4. Déterminez les probabilités de transitions
5. Déterminez les coûts associés aux transitions.

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.

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

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 :

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

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

$$\mathbb{E}\left[\underline{w}_k\right] = \underline{0} \tag{B.19}$$

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

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

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

$$Q_k \geq 0 \qquad R_k > 0 \tag{B.21}$$

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 :

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

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

$$\underline{u}_k^* = c_k^*(\underline{x}_k) = -K_k\underline{x}_k \tag{B.23}$$

où K_k est la matrice de gain égale à

$$K_k = \left[R_k + B_k^T S_{k+1} B_k\right]^{-1} B_k S_{k+1} A_k \tag{B.24}$$

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 :

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

Notes sur la dérivation d'un scalaire par un vecteur

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

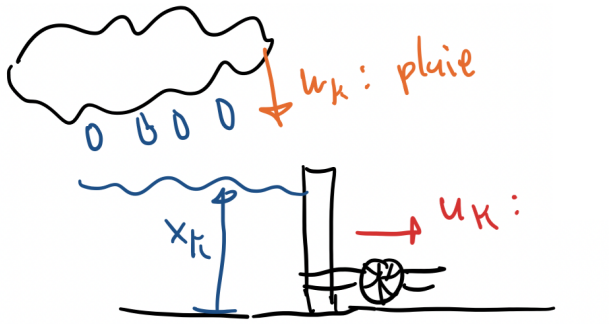
$$\underline{z} = \frac{\partial y}{\partial \underline{x}} = \left[\begin{array}{ccc} \frac{\partial y}{\partial x_1} & ... & \frac{\partial y}{\partial x_n} \end{array} \right] \quad \Leftrightarrow \quad z_i = \frac{\partial y}{\partial x_i} \tag{B.26}$$

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 :

$$x_{k+1} = \max\left[x_k - u_k + w_k, 4\right] \tag{B.27}$$

où

$$x_k \in [0, 1, 2, 3, 4] \tag{B.28}$$
$$u_k \in [0, 1, 2] \tag{B.29}$$
$$u_k \leq x_k \tag{B.30}$$
$$w_k = \begin{cases} 0 & P = 0.4 \\ 1 & P = 0.4 \\ 3 & P = 0.2 \end{cases} \tag{B.31}$$

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 :

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

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 :

$$J = \lim_{N \to \infty} \mathbb{E}\left[\sum_{k=0}^{N} \alpha^k g_k\right] \tag{B.33}$$

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

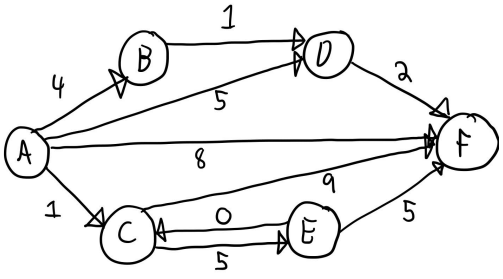
$$u_k^* = c^*(x_k) = \begin{cases} ? & \text{si} \quad x_k = 1 \\ ? & \text{si} \quad x_k = 2 \\ ? & \text{si} \quad x_k = 3 \\ ? & \text{si} \quad x_k = 4 \end{cases} \tag{B.34}$$

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 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) :

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

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 obtenus 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 :

$$x_{k+1} = 0.5x_k + u_k \tag{B.36}$$

$$g_k = x_k^2 + u_k^2 \tag{B.37}$$

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 :

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

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

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

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 travaillez 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 substitué la matrice S par la solution de l'équation de Riccati algébrique. Ici toutes les matrices sont 1x1 donc des scalaires.
