

Stabilisation d'un pendule inversé à l'aide d'un algorithme d'apprentissage automatique

Yahya Benaissa

Introduction



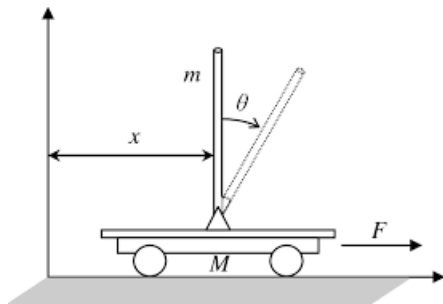
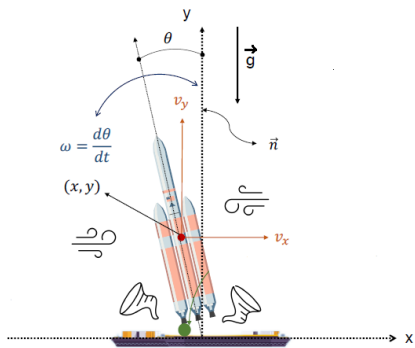
Figure – Segway



Figure – Les pattes d'une fusée

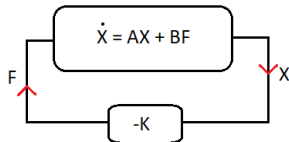


Figure – Robot bipède

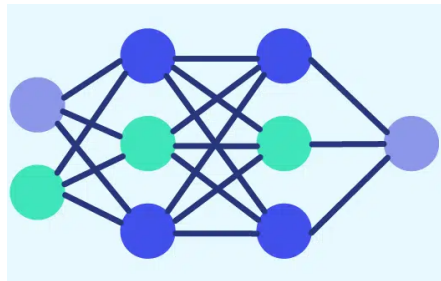


LQR : Un des piliers du contrôle moderne

le LQR généralisé est utilisé pour des problèmes de suivi de trajectoire en aviation, y compris en présence de turbulences et de mesures imprécises.



Vers un contrôle intelligent



L'apprentissage par renforcement permet à un agent d'apprendre à contrôler un système instable par essais et erreurs, sans connaissance préalable de sa dynamique

Problématique

Comment implémenter un algorithme d'apprentissage automatique au sein du pendule inversé afin de fournir une meilleure robustesse au contrôle ?

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

Paramètres :

- M : masse du chariot
- m : masse du pendule
- L : longueur de la tige
- I : moment d'inertie de la tige autour de son pivot
- b : coefficient de frottement raie-chariot
- c : couple de frottement

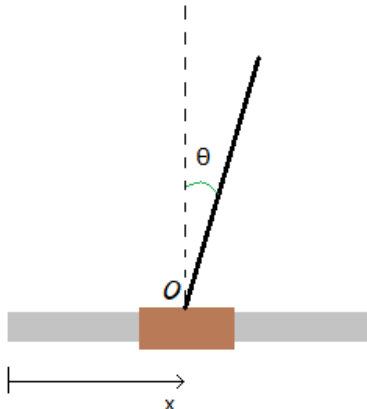
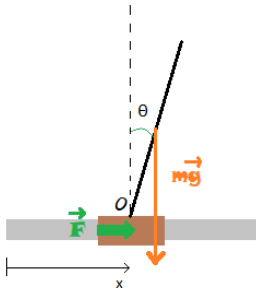


Figure – Schéma du système pendule chariot

PFD au point O au système (tige+chariot)

$$\begin{aligned}
 & \left\{ \begin{array}{c} \vec{F} \\ \vec{0} \end{array} \right\}_O + \left\{ \begin{array}{c} m\vec{g} \\ -mgL \sin \theta \vec{e}_z \end{array} \right\}_O + \left\{ \begin{array}{c} M\vec{g} \\ \vec{0} \end{array} \right\}_O + \left\{ \begin{array}{c} \vec{R}_N \\ \vec{0} \end{array} \right\}_O + \left\{ \begin{array}{c} -b\dot{x}\vec{e}_x \\ \vec{0} \end{array} \right\}_O + \left\{ \begin{array}{c} \vec{0} \\ c\dot{\theta}\vec{e}_z \end{array} \right\}_O \\
 &= \left\{ \begin{array}{c} M\vec{a}_M \\ \vec{0} \end{array} \right\}_O + \left\{ \begin{array}{c} m\vec{a}_m \\ (-I\ddot{\theta} - mL\ddot{x} \cos \theta + mL\dot{x}\dot{\theta} \sin \theta)\vec{e}_z \end{array} \right\}_O
 \end{aligned}$$



Équations finales du mouvement

Les équations dynamiques du système chariot–pendule peuvent s'écrire :

$$\begin{cases} (M + m)\ddot{x} + b\dot{x} + mL\ddot{\theta} \cos \theta - mL\dot{\theta}^2 \sin \theta & = F \\ I\ddot{\theta} + c\dot{\theta} + mL\ddot{x} \cos \theta - mgL \sin \theta - mL\dot{x}\dot{\theta} \sin \theta & = 0 \end{cases}$$

Caractéristiques du modèle :

- Non-linéarités : $\cos \theta$, $\sin \theta$, $\dot{\theta}^2 \sin \theta$, $\dot{x}\dot{\theta} \sin \theta$
- Couplage via $\ddot{x}$ et $\ddot{\theta}$

Point d'équilibre

$\theta = 0$: Pendule vers le haut

Point d'équilibre

$\theta = 0$: Pendule vers le haut

Hypothèses pour la linéarisation

- Petit angle : $\theta \ll 1$
- Petite vitesse angulaire : $\dot{\theta} \ll 1$

$$\sin \theta \approx \theta, \quad \cos \theta \approx 1, \quad \dot{\theta} \sin \theta \approx 0$$

Point d'équilibre

$\theta = 0$: Pendule vers le haut

Hypothèses pour la linéarisation

- Petit angle : $\theta \ll 1$
- Petite vitesse angulaire : $\dot{\theta} \ll 1$

$$\sin \theta \approx \theta, \quad \cos \theta \approx 1, \quad \dot{\theta} \sin \theta \approx 0$$

Équations linéarisées :

$$\begin{cases} (M + m)\ddot{x} + mL\ddot{\theta} + b\dot{x} = F \\ mL\ddot{x} + I\ddot{\theta} + c\dot{\theta} - mgL\theta = 0 \end{cases}$$

Représentation matricielle :

$$\Delta = (M + m)l - m^2 L^2$$

$$\underbrace{\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix}}_{\dot{X}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{lb}{\Delta} & -\frac{m^2 g L^2}{\Delta} & \frac{m L c}{\Delta} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{blm}{\Delta} & \frac{(M+m)mg l}{\Delta} & -\frac{(M+m)c}{\Delta} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}}_X + \underbrace{\begin{bmatrix} 0 \\ \frac{l}{\Delta} \\ 0 \\ -\frac{mL}{\Delta} \end{bmatrix}}_B F$$

C'est cette représentation qui sert de base au contrôleur LQR.

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

Principe du LQR

Le LQR s'applique au système linéaire :

$$\dot{X}(t) = AX(t) + BF(t)$$

Principe du LQR

Le LQR s'applique au système linéaire :

$$\dot{X}(t) = AX(t) + BF(t)$$

Critère quadratique à minimiser :

$$J(u) = \int_0^{\infty} (X(t)^T Q X(t) + F(t)^T R F(t)) dt$$

Principe du LQR

Le LQR s'applique au système linéaire :

$$\dot{X}(t) = AX(t) + BF(t)$$

Critère quadratique à minimiser :

$$J(u) = \int_0^\infty (X(t)^T Q X(t) + F(t)^T R F(t)) dt$$

avec :

$$Q = \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_{\dot{x}} & 0 & 0 \\ 0 & 0 & q_\theta & 0 \\ 0 & 0 & 0 & q_{\dot{\theta}} \end{bmatrix}$$

R est un scalaire positif

Principe du LQR

Le LQR s'applique au système linéaire :

$$\dot{X}(t) = AX(t) + BF(t)$$

Critère quadratique à minimiser :

$$J(u) = \int_0^\infty (X(t)^T Q X(t) + F(t)^T R F(t)) dt$$

avec :

$$Q = \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_{\dot{x}} & 0 & 0 \\ 0 & 0 & q_\theta & 0 \\ 0 & 0 & 0 & q_{\dot{\theta}} \end{bmatrix}$$

R est un scalaire positif

Objectif

Trouver F^* tel que $J(F^*) = \min_F J(F)$ et $\lim_{t \rightarrow \infty} X(t) = 0$

Solution LQR

La commande optimale est :

$$F(t) = -KX(t)$$

voir demo annexe

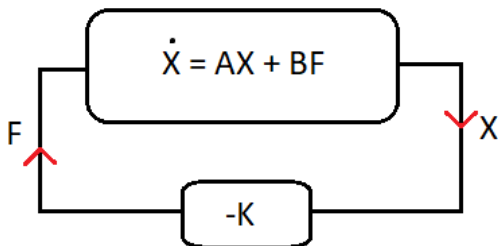
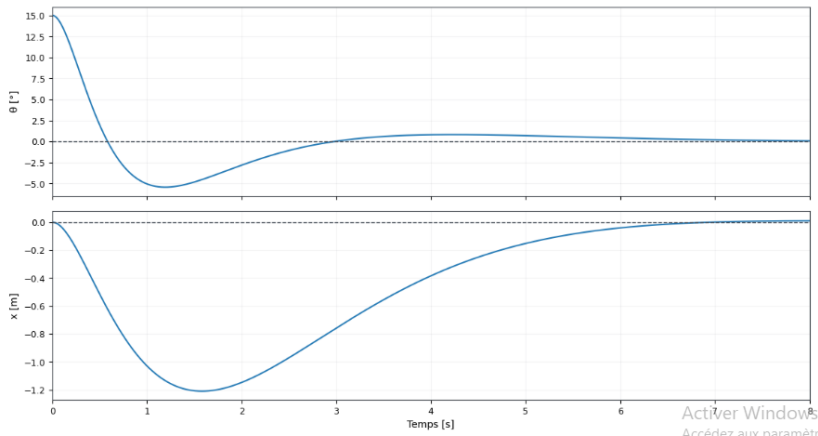


Figure – Boucle de rétroaction LQR

Simulation numérique



Le LQR arrive à stabiliser le pendule avec une position angulaire initiale de 15 degrés, (proche de 0).

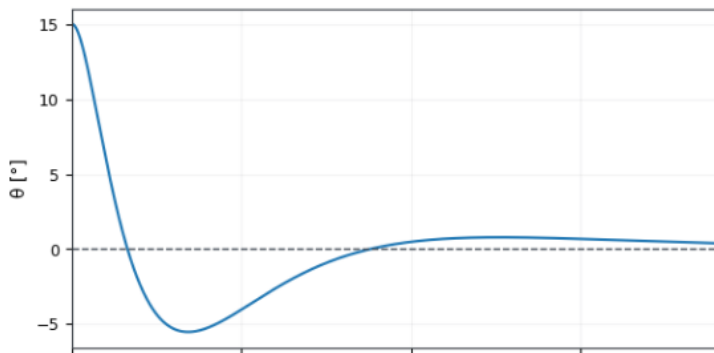
Importance du choix de Q et R

Premier choix

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

$$R = 0.1$$

Importance du choix de Q et R



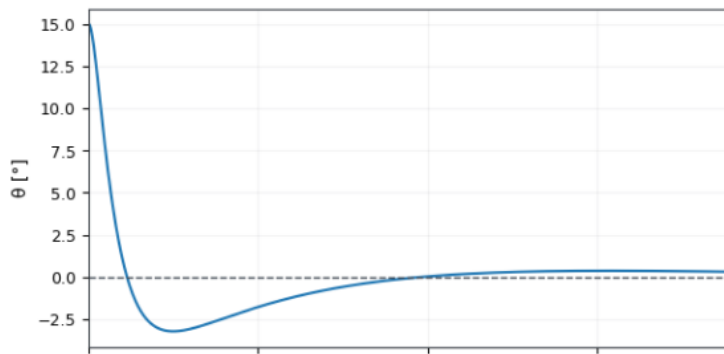
Dépassement angulaire : 40% ; Temps de stabilisation : 1.5 s

Deuxième choix :

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

$$R = 0.01$$

Importance du choix de Q et R



Dépassement angulaire : 18% ; Temps de stabilisation : 1.3 s

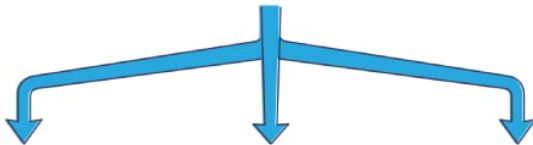
Problèmes

- Le LQR demande une modélisation
- Hypothèse d'un système non linéaire
- Hypothèses d'un environnement fixe

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning**
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

MACHINE LEARNING



**Apprentissage
supervisé**



**Apprentissage
non supervisé**



**Apprentissage
par renforcement**

Qu'est-ce que l'apprentissage par renforcement ?

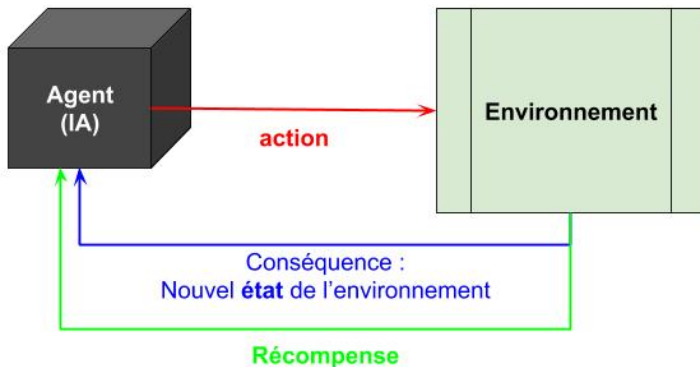


Figure – Apprentissage par renforcement

Qu'est-ce que l'apprentissage par renforcement ?

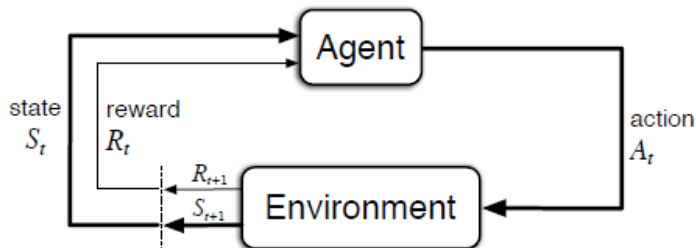


Figure – Boucle d'apprentissage

Q-learning

On définit l'espace d'état : $S = \{s_1, s_2, s_3, s_4\}$, et l'espace d'actions $A = \{a_1, a_2, a_3\}$

Action/Etat	s_1	s_2	s_3	s_4
a_1	5	4	5	1
a_2	25	30	31	20
a_3	0	-4	1	2

Concept clé

Le Q-learning cherche, pour l'état s_i , l'action a_i qui a la plus grande évaluation $Q(s_i, a_i)$ dans la Q-table, et suivant la récompense r_i , il met à jour la valeur $Q(s_i, a_i)$.

- $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ états du système
- $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ actions disponibles
- $r_t \in \mathbb{R}$ récompense à l'étape t
- $\pi : \mathcal{S} \rightarrow \mathcal{A}$ politique de l'agent

Fonction valeur $Q(s, a)$

Pour un état $s \in \mathcal{S}$ et une action $a \in \mathcal{A}$, la **fonction valeur action-état** $Q(s, a)$ est le *retour cumulé actualisé espéré* obtenu en exécutant a depuis s , puis en suivant la politique π :

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{+\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right]$$

La fonction **optimale** $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ donne la valeur maximale atteignable depuis (s, a) .

Plus $Q(s, a)$ est élevé, plus l'action a est **prometteuse** depuis l'état s .

La politique π

La politique d'un algorithme Q-learning est :

$$\pi(s) = \operatorname{argmax}_a Q(s, a)$$

Règle de mise à jour de la fonction valeur

la fonction valeur action-état peut se définir aussi par la relation

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

voir démo annexe

Avec une exploration suffisante, la Q-table converge selon cette relation vers la Q^* -table optimale.

Algorithme du Q-learning

Initialiser $Q(s, a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrairement, et
 $Q(\text{état-terminal}, \cdot) = 0$

Répéter (pour chaque épisode) :

Initialiser S

Répéter (pour chaque étape de l'épisode) :

Choisir A depuis S en utilisant la politique dérivée de Q (e.g.,
 ϵ -greedy)

Effectuer l'action A , observer R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

jusqu'à ce que S soit terminal

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé**
- 5 Commande optimale : Approche hybride
- 6 Annexe

- $S :$
 $\{(i, j, k, l) / i, j, k, l \in [1, 10]\}$
- $A : \{+F, -F\}$
- $r = \cos \theta - 0.1 * \dot{\theta}^2 - 0.7$

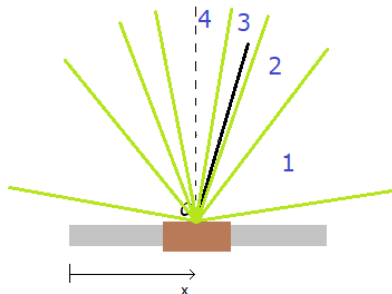


Figure – Discrétisation de l'espace d'états

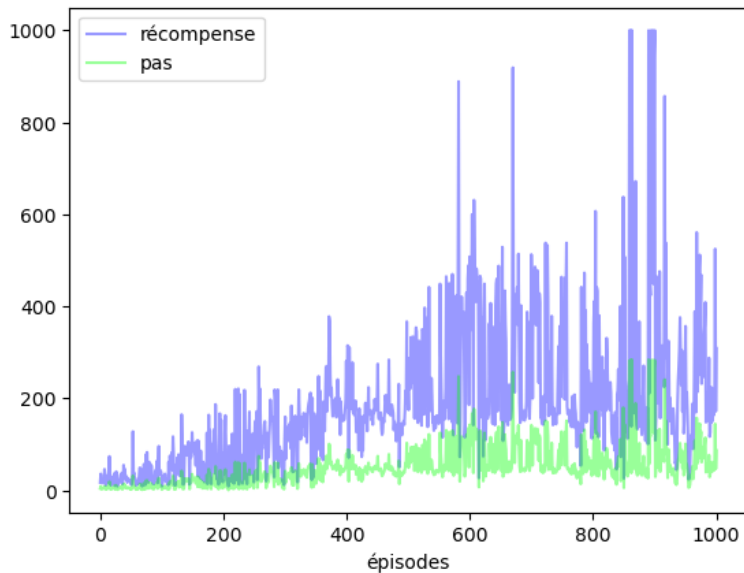
Discrétisation de l'espace d'état

Variable	Plage	Nombre de bins
Position x	$[-4.8, 4.8]$ m	10
Vitesse $\dot{x}$	$[-3.5, 3.5]$ m/s	10
Angle θ	$[-3.1, 3.1]$ rad	10
Vitesse angulaire $\dot{\theta}$	$[-3.5, 3.5]$ rad/s	10

Taille de la table Q

$$10 \times 10 \times 10 \times 10 \times |A| = 20000$$

Résultats de la simulation



Problème

- Nécessité d'un très grand nombre d'épisodes

Solution

Choix d'une approche hybride, qui repose sur le LQR, qui ajuste pour chaque état, la matrice de gain K .

$$F = -K(X)X$$

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

Nouvelle formulation

- $S : \{(i, j, k, l) / i, j, k, l \in [1, 10]\}$
- $r = -(2x^2 + 0.5\dot{x}^2 + 1000\theta^2 + 5\dot{\theta}^2 + 0.01u^2) + 20$

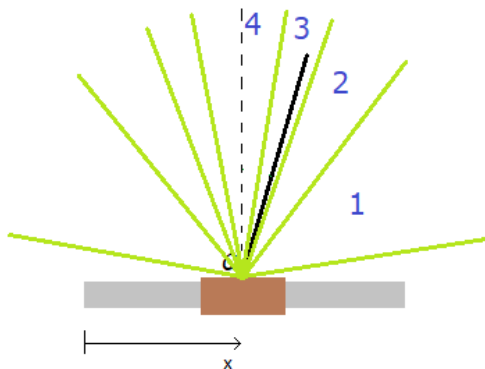


Figure – Discrétisation de l'espace d'états

On définit l'espace des actions comme suit :

```
K_set = [np.array([[K1 + ((i-1.5)*K_position)/10  
                    , K2 + ((j-1.5)*K_vitesse)/10  
                    , K3 + ((k-1.5)*K_theta)/10  
                    , K4 + ((l-1.5)*K_vitesseang)/10]])  
        for i in range(4)  
        for j in range(4)  
        for k in range(4)  
        for l in range(4)]+[K_lqr]
```

Sachant que $K_{lqr}=[K1,K2,K3,K4]$

Résultats avec une petite perturbation($\theta_0 = 0.4rad$)

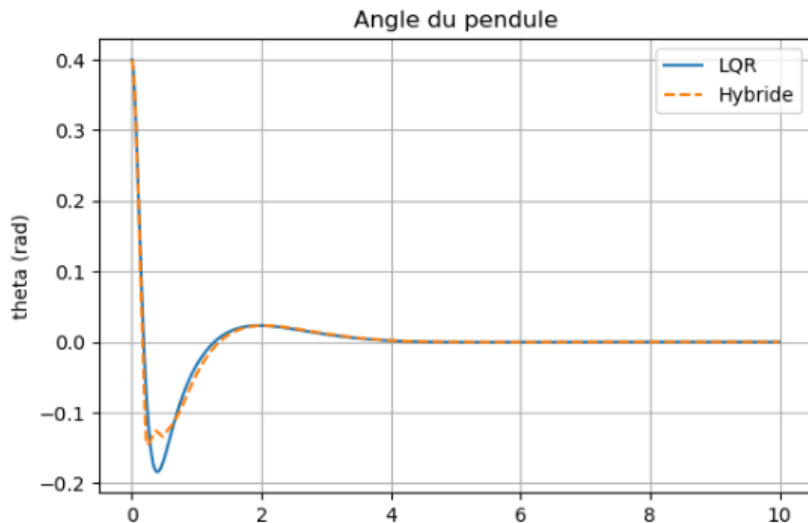


Figure – Evolution de θ en fonction du temps

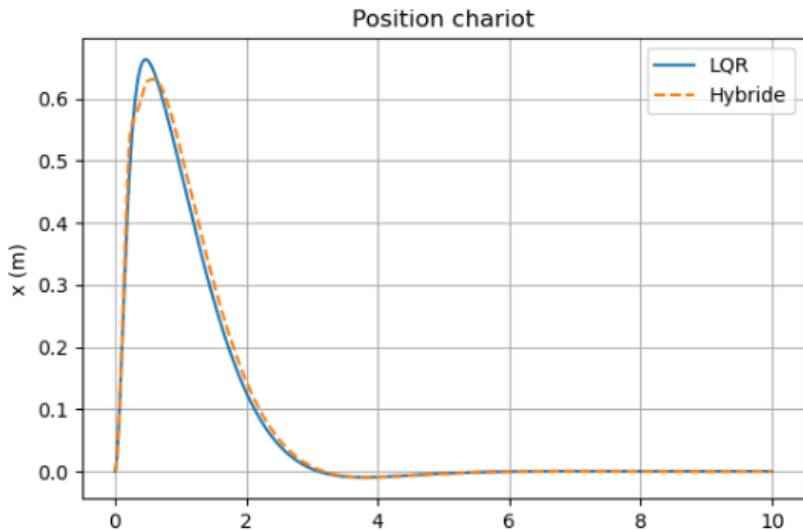
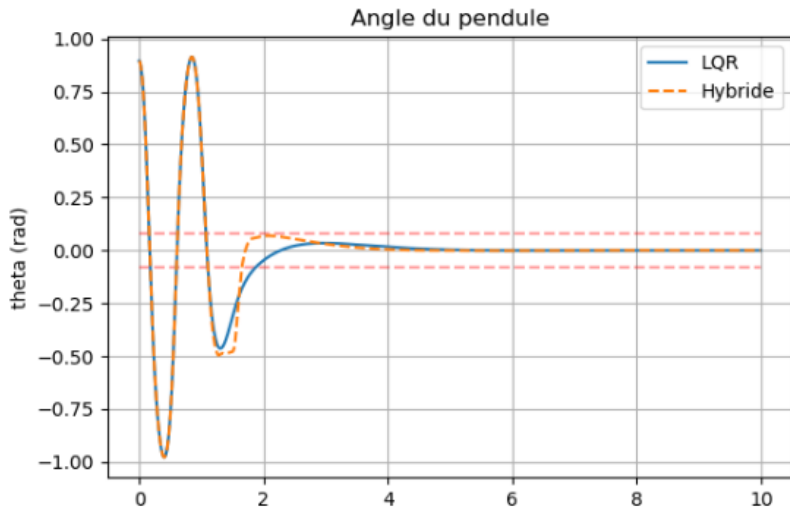


Figure – Evolution de x en fonction du temps

Résultats avec une grande perturbation($\theta_0 = 0.9rad$)



Position chariot

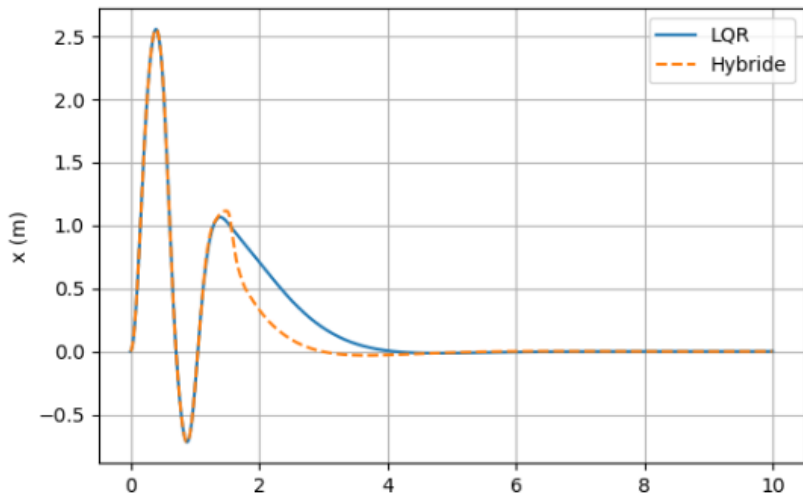


Tableau de comparaison

Cas testé	Méthode	Dép. θ (%)	T. stab. θ	T. stab. x
$\theta_0 = 0.4$ rad	LQR	47.5	1.0 s	2.5 s
$\theta_0 = 0.4$ rad	Hybride	32.5	1.0 s	2.5 s
$\theta_0 = 0.9$ rad	LQR	100	1.8 s	3.5 s
$\theta_0 = 0.9$ rad	Hybride	100	1.6 s	2.6 s

Conclusion

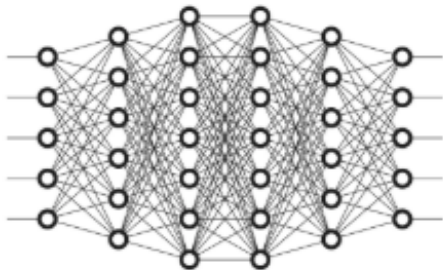


Figure – Réseau de neurones

Merci pour votre attention



O. Boubaker, “The inverted pendulum : A fundamental benchmark in control theory and robotics,” in *Proc. Int. Conf. Education and E-Learning Innovations*, pp. 1–6, Jul. 2012.



L. B. Prasad, B. Tyagi, et H. O. Gupta, “Optimal control of nonlinear inverted pendulum system using PID controller and LQR : performance analysis without and with disturbance input,” *International Journal of Automation and Computing*, vol. 11, no. 6, pp. 661–670, 2014.



D. R. Tyner et A. D. Lewis, “Geometric Jacobian linearization and LQR theory,” *Journal of Geometric Mechanics*, vol. 2, no. 4, pp. 397–440, 2010.



R. S. Sutton et A. G. Barto, *Reinforcement Learning : An Introduction*, 2nd ed. Cambridge, MA, USA : MIT Press, 2018.



S. Israilov, L. Fu, J. Sanchez-Rodriguez, F. Fusco, G. Allibert, C. Raufaste, et M. Argentina, "Reinforcement learning approach to control an inverted pendulum : A general framework for educational purposes," *PLOS ONE*, vol. 18, no. 2, p. e0280071, Feb. 2023, doi : 10.1371/journal.pone.0280071.



T. V. Bui, "Stabilisation d'un double pendule inversé par microprocesseur," Ph.D. dissertation, École Nationale Supérieure des Mines de Paris, Paris, France, 1983.

Plan de la présentation

- 1 Mise en équation du système du pendule inversé
- 2 Méthode de contrôle par LQR
- 3 Introduction du Q-learning
- 4 Implémentation du Q-learning dans le pendule inversé
- 5 Commande optimale : Approche hybride
- 6 Annexe

Annexe A : Démonstration de la solution LQR

Système linéaire continu :

$$\dot{x} = Ax + Bu, \quad x(0) = x_0, \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^p$$

Critère quadratique à horizon infini :

$$J(u) = \int_0^{\infty} (x^T Q x + u^T R u) dt$$

avec $Q = Q^T \geq 0$ et $R = R^T > 0$ (données de conception).

Stratégie de démonstration matricielle :

On va montrer que pour toute commande possible $u(t)$, on peut écrire une identité matricielle qui décompose J comme une somme d'un terme positif et d'un terme fixé. La minimisation sera alors immédiate par annulation du terme variable, ce qui fera naturellement apparaître $u^* = -Kx$.

Soit $P = P^T > 0$ une matrice symétrique définie positive à déterminer.
Considérons la fonction $V(x) = x^T P x$.

Sa dérivée totale le long des trajectoires de $\dot{x} = Ax + Bu$ est :

$$\begin{aligned}\frac{d}{dt} (x^T P x) &= \dot{x}^T P x + x^T P \dot{x} \\ &= (Ax + Bu)^T P x + x^T P (Ax + Bu) \\ &= x^T A^T P x + u^T B^T P x + x^T P A x + x^T P B u \\ &= x^T (A^T P + P A) x + u^T B^T P x + x^T P B u\end{aligned}$$

En regroupant les deux derniers termes (qui sont transposés l'un de l'autre, donc scalaires égaux) :

$$\frac{d}{dt} (x^T P x) = x^T (A^T P + P A) x + 2x^T P B u$$

On réécrit $x^\top Qx + u^\top Ru$ en ajoutant et soustrayant $\frac{d}{dt}(x^\top Px)$:

$$x^\top Qx + u^\top Ru = \frac{d}{dt}(x^\top Px) + x^\top Qx + u^\top Ru - \frac{d}{dt}(x^\top Px)$$

En substituant l'expression de $\frac{d}{dt}(x^\top Px)$ précédemment calculée :

$$\begin{aligned} &= \frac{d}{dt}(x^\top Px) + x^\top Qx + u^\top Ru - x^\top (A^\top P + PA)x - 2x^\top PBu \\ &= \frac{d}{dt}(x^\top Px) + x^\top [Q - A^\top P - PA]x + u^\top Ru - 2x^\top PBu \end{aligned}$$

On fait apparaître un carré sur u en complétant :

$$u^\top Ru - 2x^\top PBu = (u + R^{-1}B^\top Px)^\top R(u + R^{-1}B^\top Px) - x^\top PBR^{-1}B^\top Px$$

car $R > 0$ permet de factoriser.

En substituant la complétion du carré, on obtient :

$$\begin{aligned} \mathbf{x}^\top \mathbf{Q} \mathbf{x} + \mathbf{u}^\top \mathbf{R} \mathbf{u} &= \frac{d}{dt}(\mathbf{x}^\top \mathbf{P} \mathbf{x}) + \mathbf{x}^\top \underbrace{\left[\mathbf{Q} - \mathbf{A}^\top \mathbf{P} - \mathbf{P} \mathbf{A} + \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \right]}_{=: \Delta(\mathbf{P})} \mathbf{x} \\ &\quad + (\mathbf{u} + \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{x})^\top \mathbf{R} (\mathbf{u} + \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{x}) \end{aligned}$$

Choix clé de $\mathbf{P}$ — Équation de Riccati Algébrique (ARE) :

On impose $\Delta(\mathbf{P}) = 0$, c'est-à-dire que $\mathbf{P}$ est solution de :

$$\boxed{\mathbf{A}^\top \mathbf{P} + \mathbf{P} \mathbf{A} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} + \mathbf{Q} = 0}$$

Alors on obtient cette simplification :

$$\mathbf{x}^\top \mathbf{Q} \mathbf{x} + \mathbf{u}^\top \mathbf{R} \mathbf{u} = \frac{d}{dt}(\mathbf{x}^\top \mathbf{P} \mathbf{x}) + (\mathbf{u} + \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{x})^\top \mathbf{R} (\mathbf{u} + \mathbf{R}^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{x})$$

On intègre l'identité sur $[0, +\infty[$:

$$J(u) = \int_0^\infty \left[\frac{d}{dt}(\mathbf{x}^\top P \mathbf{x}) + (u + R^{-1} B^\top P \mathbf{x})^\top R (u + R^{-1} B^\top P \mathbf{x}) \right] dt$$

Le premier terme s'intègre télescopiquement :

$$\int_0^\infty \frac{d}{dt}(\mathbf{x}^\top P \mathbf{x}) dt = \mathbf{x}^\top(\infty) P \mathbf{x}(\infty) - \mathbf{x}_0^\top P \mathbf{x}_0$$

Pour une commande stabilisante $\mathbf{x}(t) \rightarrow 0$, le terme en ∞ s'annule.

$$J(u) = \mathbf{x}_0^\top P \mathbf{x}_0 + \int_0^\infty (u + R^{-1} B^\top P \mathbf{x})^\top R (u + R^{-1} B^\top P \mathbf{x}) dt$$

Puisque $R > 0$, le terme intégral est toujours ≥ 0 , et il est nul si et seulement si :

$$u(t) + R^{-1} B^\top P \mathbf{x}(t) = 0 \quad \forall t \geq 0$$

La commande qui annule ce terme et minimise donc J est uniquement :

$$u^*(t) = -R^{-1} B^\top P \mathbf{x}(t) =: -K \mathbf{x}(t), \quad K = R^{-1} B^\top P$$

La valeur minimale du critère est $J^* = \mathbf{x}_0^\top P \mathbf{x}_0$.

Annexe B : Code python LQR

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as la

M, m, l, g, b = 1.0, 0.3, 0.5, 9.81, 0.1
I = (1/3)*m*(2*l)**2
den = I*(M+m) - M*m*l**2

A=np.array([[0,1,0,0],
            [0,-I*b/den,(m**2*g*l**2)/den,-m*l*c/den],
            [0,0,0,1],
            [0,-(m*l*b)/den,m*g*l*(M+m)/den,-(M+m)c/den]])
B = np.array([[0],[I/den],[0],[-m*l/den]])
```

```

def lqr(Q, R):
    P = la.solve_continuous_are(A, B, Q, R)
    return np.linalg.inv(R) @ B.T @ P

def simulate(K, x0, T=8.0, dt=0.004):
    t = np.arange(0, T, dt)
    xs = np.zeros((len(t), 4)); xs[0] = x0
    us = np.zeros(len(t))
    Acl = A - B @ K
    for i in range(len(t)-1):
        us[i] = float((-K @ xs[i]).item())
        xs[i+1] = xs[i] + dt * (Acl @ xs[i])
    us[-1] = float((-K @ xs[-1]).item())
    return t, xs, us

```

Annexe C : Démonstration de la règle de mise à jour Q-learning

Un MDP est un tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ où $\mathcal{S}$ est l'espace d'états, $\mathcal{A}$ l'espace d'actions, $P(s'|s, a)$ la probabilité de transition vers s' depuis (s, a) , $r(s, a, s')$ la récompense immédiate reçue, et $\gamma \in [0, 1]$ le facteur d'actualisation.

Pour une politique π , on définit la **Q-fonction** (fonction de valeur action-état) comme l'espérance du retour cumulé actualisé en partant de l'état s , en prenant l'action a , puis en suivant π :

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{+\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right]$$

En conditionnant sur la première transition, on peut décomposer Q^π de façon récursive. La récompense immédiate est $r(s, a, s')$, et après être arrivé en s' , l'agent choisit une action a' selon $\pi(a'|s')$ et accumule $\gamma Q^\pi(s', a')$. En moyennant sur s' :

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a) \left[r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right]$$

La Q-fonction **optimale** est $Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$. En remplaçant la somme sur π par un max sur a' , on obtient **l'équation d'optimalité de Bellman** :

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) \left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

Connaître Q^* suffit à reconstruire la politique optimale :

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a).$$

En pratique, on ne connaît ni $P(s'|s, a)$ ni r de façon explicite. On ne dispose que de **transitions observées** de la forme $(s_t, a_t, r_{t+1}, s_{t+1})$ générées par interaction avec l'environnement.

L'équation d'optimalité de Bellman peut se réécrire comme une espérance :

$$Q^*(s_t, a_t) = E \left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \right]$$

L'idée centrale du Q-Learning est de remplacer cette espérance par **un seul échantillon** observé à chaque instant t . On appelle *cible* (ou *target*) la quantité :

$$y_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a')$$

où Q_t désigne l'estimation courante de Q^* à l'instant t .

On cherche à réduire l'écart entre $Q_t(s_t, a_t)$ et la cible y_t . On définit l'**erreur de Bellman empirique** (aussi appelée erreur TD) :

$$\delta_t = \underbrace{r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a')}_{y_t} - Q_t(s_t, a_t)$$

On minimise la perte $|\delta_t|$ par rapport à $Q_t(s_t, a_t)$, d'où la mise à jour par descente de gradient stochastique :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \delta_t$$

En développant δ_t , on obtient la **règle de mise à jour Q-Learning** (Watkins, 1989) :

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \right]$$

avec $\alpha_t \in [0, 1]$ le **taux d'apprentissage**. Seule la case (s_t, a_t) est modifiée ; toutes les autres entrées de la Q-table restent inchangées.

Annexe D : Convergence Q-learning

Le théorème de convergence repose sur trois hypothèses.

(H1) — Exploration infinie. Toutes les paires $(s, a) \in \mathcal{S} \times \mathcal{A}$ sont visitées un nombre infini de fois. Cela garantit que toutes les entrées de la Q-table sont mises à jour indéfiniment.

(H2) — Conditions de Robbins-Monro sur les taux d'apprentissage. Les taux $\alpha_t(s, a)$ vérifient :

$$\sum_{t=0}^{\infty} \alpha_t(s, a) = +\infty \quad \text{et} \quad \sum_{t=0}^{\infty} \alpha_t^2(s, a) < +\infty$$

La première condition assure que les mises à jour ont un effet global cumulatif non nul. La seconde assure que le bruit d'échantillonnage finit par être amorti.

(H3) — Récompenses bornées. Il existe $R_{\max} < +\infty$ tel que $R(s, a, s') \leq R_{\max}$ pour tout (s, a, s') . Cela garantit que le bruit d'échantillonnage reste borné en variance.

Théorème (Watkins & Dayan, 1992).

Soit $(Q_t)_{t \geq 0}$ la suite générée par la règle de mise à jour Q-Learning, initialisée arbitrairement. Sous les hypothèses (H1), (H2) et (H3), la suite converge presque sûrement vers la Q-fonction optimale Q^* :

$$Q_t(s, a) \xrightarrow[t \rightarrow \infty]{\text{p.s.}} Q^*(s, a) \quad \forall Q_t(s, a) \in \mathcal{S} \times \mathcal{A}$$

La preuve se déroule en trois étapes :
contraction de l'opérateur de Bellman, dynamique de l'erreur globale, puis application d'un lemme de convergence stochastique.

On pose $\Delta_t(s, a) = Q_t(s, a) - Q^*(s, a)$, l'erreur de l'estimation à l'instant t .

On introduit l'**opérateur de Bellman optimal** $\mathcal{T}$, défini pour toute fonction Q par :

$$\mathcal{T}Q(s, a) = \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') \right]$$

C'est exactement la valeur que $Q(s, a)$ devrait avoir si la mise à jour utilisait l'espérance exacte plutôt qu'un échantillon. On définit alors le **bruit d'échantillonnage** :

$$F_t(s, a) = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - \mathcal{T}Q_t(s_t, a_t)$$

C'est la différence entre la cible observée et sa valeur en espérance. Puisque $E[r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') \mid \mathcal{F}_t] = \mathcal{T}Q_t(s_t, a_t)$, on a $E[F_t \mid \mathcal{F}_t] = 0$: F_t est une **martingale-différence**.

La règle Q-Learning se réécrit donc :

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t [\mathcal{T}Q_t(s_t, a_t) - Q_t(s_t, a_t) + F_t(s_t, a_t)]$$

On munit l'espace des Q-fonctions de la norme sup

$$\|Q\|_{\infty} = \max_{s,a} |Q(s, a)|.$$

Lemme. Pour toutes Q-fonctions Q et Q' , on a

$$\|\mathcal{T}Q - \mathcal{T}Q'\|_{\infty} \leq \gamma \|Q - Q'\|_{\infty}.$$

Démonstration. Fixons (s, a) quelconque. On calcule :

$$\begin{aligned} |\mathcal{T}Q(s, a) - \mathcal{T}Q'(s, a)| &= \gamma \left| \sum_{s'} P(s'|s, a) \left[\max_{a'} Q(s', a') - \max_{a'} Q'(s', a') \right] \right| \\ &\leq \gamma \sum_{s'} P(s'|s, a) \left| \max_{a'} Q(s', a') - \max_{a'} Q'(s', a') \right| \end{aligned}$$

On utilise que $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$, donc :

$$\leq \gamma \sum_{s'} P(s'|s, a) \max_{a'} |Q(s', a') - Q'(s', a')| \leq \gamma \|Q - Q'\|_{\infty}$$

Comme cela est vrai pour tout (s, a) , on conclut

$\|\mathcal{T}Q - \mathcal{T}Q'\|_{\infty} \leq \gamma \|Q - Q'\|_{\infty}$. Puisque $\gamma < 1$, $\mathcal{T}$ est une contraction stricte, et Q^* est son unique point fixe. □

Puisque Q^* est le point fixe de $\mathcal{T}$, on a $Q^*(s, a) = \mathcal{T}Q^*(s, a)$. On soustrait $Q^*(s_t, a_t)$ des deux membres de la règle Q-Learning réécrite :

$$\Delta_{t+1}(s_t, a_t) = (1 - \alpha_t) \Delta_t(s_t, a_t) + \underbrace{\alpha_t [\mathcal{T}Q_t(s_t, a_t) - \mathcal{T}Q^*(s_t, a_t)]}_{\leq \gamma \|\Delta_t\|_\infty \text{ par contraction}} +$$

$$\alpha_t F_t(s_t, a_t)$$

En passant à la norme sup sur toutes les paires (s, a) , et en notant que $(1 - \alpha_t) \|\Delta_t\| + \alpha_t \gamma \|\Delta_t\|_\infty \leq (1 - \alpha_t(1 - \gamma)) \|\Delta_t\|_\infty$, on obtient :

$$\|\Delta_{t+1}\|_\infty \leq (1 - \alpha_t(1 - \gamma)) \|\Delta_t\|_\infty + \alpha_t |F_t|$$

Cette inégalité dit que l'erreur globale décroît à vitesse $(1 - \alpha_t(1 - \gamma))$, perturbée par le terme de bruit $\alpha_t |F_t|$. Il reste à montrer que ce bruit disparaît.

Lemme de convergence stochastique (Robbins-Siegmund, 1971). Soit (V_t) un processus positif vérifiant $V_{t+1} \leq (1 - \beta_t)V_t + \beta_t\epsilon_t$ avec $\beta_t \in [0, 1]$, $\sum \beta_t = +\infty$, $\sum \beta_t^2 < +\infty$, et $\epsilon_t \rightarrow 0$ presque sûrement. Alors $V_t \rightarrow 0$ presque sûrement.

Application à notre cadre. On pose :

$$V_t = \|\Delta_t\|_\infty, \quad \beta_t = \alpha_t(1 - \gamma), \quad \epsilon_t = \frac{\|F_t\|}{1 - \gamma}$$

Les β_t héritent des conditions de Robbins-Monro via le facteur $(1 - \gamma) > 0$: $\sum \beta_t = +\infty$ et $\sum \beta_t^2 < +\infty$ sont satisfaites.

Par la loi forte des grands nombres pour les martingales-différences bornées (condition H3), et grâce à l'exploration infinie (H1) qui garantit que les $\alpha_t(s, a)$ vérifient H2 pour chaque paire, on a $F_t \rightarrow 0$ presque sûrement.

Le lemme s'applique et donne $\Delta_{t\infty} \rightarrow 0$ presque sûrement, c'est-à-dire $Q_t(s, a) \rightarrow Q^*(s, a)$ pour toute paire (s, a) .

Annexe E : Code python "Q-learning pur"

```
import gymnasium as gym
import numpy as np
import random
import math
import matplotlib.pyplot as plt

env = gym.make('CartPole-v1')

alpha      = 0.1, gamma      = 0.99, epsilon    = 0.99
episodes   = 1000, max_steps = 1000

state_bins = [np.linspace(-4.8, 4.8, 10),
               np.linspace(-3.5, 3.5, 10),
               np.linspace(-3.1, 3.1, 10),
               np.linspace(-3.5, 3.5, 10)]
```

```

def discretize_state(state):
    state_index = []
    for i in range(len(state)):
        state_index.append(np.digitize(state[i],
                                       state_bins[i]) - 1)
    return tuple(state_index)

q_table=np.zeros([10,10,10,10,env.action_space.n])

def q_learning(state,action,reward,next_state,done):
    state_idx      = discretize_state(state)
    next_state_idx = discretize_state(next_state)
    best_future_q  = np.max(q_table[next_state_idx])
    current_q      = q_table[state_idx + (action,)]
    q_table[state_idx + (action,)] = current_q +
    alpha * (reward+gamma*best_future_q-current_q)

```

```

Y = [], Z = [], s = [], d = []

for episode in range(episodes):
    state = env.reset()[0]
    done = False
    steps = 0
    L = 0
    while not done and steps < max_steps:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(
                q_table[discretize_state(state)])

        next_state, rew, done, _, _ = env.step(action)
        reward = math.cos(state[2])
                -0.7 - (state[3]**2)*0.1

        if not done:
            q_learning(state, action,
                       reward, next_state, done)
        else:
            state_idx = discretize_state(state)
            q_table[state_idx + (action,)] = -100

        state = next_state
        steps += 1
        L += reward

    epsilon = max(0.1, epsilon * 0.995)

```

```

X = np.linspace(0, 1000, 1000)
Y = Y + [steps]
Z = Z + [L]

if episode == 0:
    s = [steps]
    d = [L]
else:
    y = s[-1]
    w = d[-1]
    s = s + [(y + steps) / (episode + 1)]
    d = d + [(w + L) / (episode + 1)]

if episode % 500 == 0:
    print(f"Episode {episode}: {steps} pas.")
    print(L)

plt.plot(X,Y,color=(0, 0, 1, 0.4),label="recompense")
plt.plot(X,Z,color=(0, 1, 0, 0.4),label="pas")
plt.xlabel('episodes')
plt.legend()
plt.show()
env.close()

```

Annexe F : Code python "Approche hybride vs LQR"

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_continuous_are
M,m,l = 0.5,0.4,0.8
g=9.81; b=0.02; dt=0.005; T=7.0
I      = (1/3)*m*(2*l)**2
den    = I*(M+m) - M*m*l**2

A=np.array([[0,1,0,0],
            [0,-I*b/den,(m**2*g*l**2)/den,-m*l*c/den],
            [0,0,0,1],
            [0,-(m*l*b)/den,m*g*l*(M+m)/den,-(M+m)*c/den]])
B = np.array([[0],[I/den],[0],[-m*l/den]])
```

```

Q_lqr = np.diag([2, 0.5, 1000, 5])
R_lqr = np.array([[0.01]])

P = solve_continuous_are(A, B, Q_lqr, R_lqr)

K_lqr = np.linalg.inv(R_lqr) @ B.T @ P
print("\nGain LQR optimal :")
print(K_lqr)
def dynamics(s,u):
    _,xd,th,thd=s; st,ct=np.sin(th),np.cos(th)
    den=M+m*(1-ct**2)
    xdd=(u-b*xd+m*L*thd**2*st-m*g*st*ct)/den
    thdd=(g*(M+m)*st-ct*(u-b*xd
        +m*L*thd**2*st))/(L*den)
    return s+ dt*np.array([xd,xdd,thd,thdd])

```

```
x_bins = np.linspace(-2, 2, 15)
xdot_bins = np.linspace(-6, 6, 15)

theta_bins = np.linspace(-0.8, 0.8, 25)
thetadot_bins = np.linspace(-5, 5, 25)

def discretize(state):

    x, xd, th, thd = state

    ix = np.digitize(x, x_bins)
    iv = np.digitize(xd, xdot_bins)
    it = np.digitize(th, theta_bins)
    iw = np.digitize(thd, thetadot_bins)

    return (ix, iv, it, iw)
```

```
K1=-14.14213562
K2=-32.52402555
K3=-358.37444058
K4=-42.49465821

K_set=[np.array([[K1 + ((i-1.5)*K_position)/10
                  ,K2 + ((j-1.5)*K_vitesse)/10
                  ,K3 + ((k-1.5)*K_theta)/10
                  ,K4 + ((l-1.5)*K_vitesseang)/10]])
        for i in range(4)
        for j in range(4)
        for k in range(4)
        for l in range(4)]+[K_lqr]

n_actions = len(K_set)
```

```

def reward(state, u):
    x, xd, th, thd = state
    cost = (2 * x**2+ 0.5 * xd**2+ 1000 * th**2+
            5 * thd**2+ 0.01 * u**2)
    r = -cost
    if abs(th) > 0.5:
        r -= 500
    return r+20
Qtable = {}
def get_Q(state):
    if state not in Qtable:
        Qtable[state]=[0 for i in
                        range(n_actions-1)]+[3]
    return Qtable[state]
alpha = 0.10
gamma = 0.99

epsilon = 0.6
epsilon_min = 0.01
epsilon_decay = 0.995

episodes = 1000
max_steps = 500

def choose_action(state):

    if np.random.rand() < epsilon:
        return np.random.randint(n_actions)

    return np.argmax(get_Q(state))

```

```

episode_rewards = []
for episode in range(episodes):
    state = np.array(
        [np.random.uniform(-0.2, 0.2),
         np.random.uniform(-0.8, 0.8),
         np.random.uniform(-0.2, 0.2),
         np.random.uniform(-1.0, 1.0)])
    cumulative_reward = 0
    for step in range(max_steps):
        s = discretize(state)
        action = choose_action(s)
        K = K_set[action]
        u = float((-K @ state).item())
        next_state = dynamics(state, u)
        r = reward(next_state, u)
        cumulative_reward += r
        s_next = discretize(next_state)
        q = get_Q(s)
        q_next = get_Q(s_next)
        q[action] += alpha * (r +
                               gamma * np.max(q_next) - q[action])
        state = next_state
        if abs(state[2]) > 0.5:
            break
    episode_rewards.append(cumulative_reward)
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
    if episode % 500 == 0:
        print(
            f"Episode {episode:4d}"
            f"|epsilon={epsilon:.3f}"
            f"|reward={cumulative_reward:.1f}")
print("\nApprentissage termine.")
policy_K = {}

```

```

for state in Qtable.keys():
    best_action = np.argmax(Qtable[state])
    policy_K[state] = K_set[best_action]
print("\nNombre d'etats appris :", len(policy_K))
def lqr_controller(state):
    u = float((-K_lqr @ state).item())
    return u
def hybrid_controller(state):
    s = discretize(state)
    if s not in Qtable:
        K = K_lqr
    else:
        action = np.argmax(Qtable[s])
        K = K_set[action]
    u = float((-K @ state).item())
    return u, K

```

```

def simulate_lqr(x0, T=10):
    n = int(T / dt)
    x = x0.copy()
    traj = []
    for _ in range(n):
        u = lqr_controller(x)
        x = dynamics(x, u)
        traj.append(x.copy())
    return np.array(traj)

def simulate_hybrid(x0, T=10):
    n = int(T / dt)
    x = x0.copy()
    traj = []
    gains = []
    for _ in range(n):
        u, K = hybrid_controller(x)
        x = dynamics(x, u)
        traj.append(x.copy())
        gains.append(K.copy())
    return np.array(traj), gains

x0 = np.array([0.0, 0.0, 0.90, 0.0])
traj_lqr = simulate_lqr(x0)
traj_hybrid, gains_used = simulate_hybrid(x0)
time = np.arange(len(traj_lqr)) * dt

```

```
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(time, traj_lqr[:, 2], label="LQR")
plt.plot(time, traj_hybrid[:, 2], "--", label="Hybride")
plt.title("Angle du pendule")
plt.ylabel("theta (rad)")
plt.grid()
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(time, traj_lqr[:, 0], label="LQR")
plt.plot(time, traj_hybrid[:, 0], "--", label="Hybride")
plt.title("Position chariot")
plt.ylabel("x (m)")
plt.grid()
plt.legend()

plt.tight_layout()
plt.show()
```