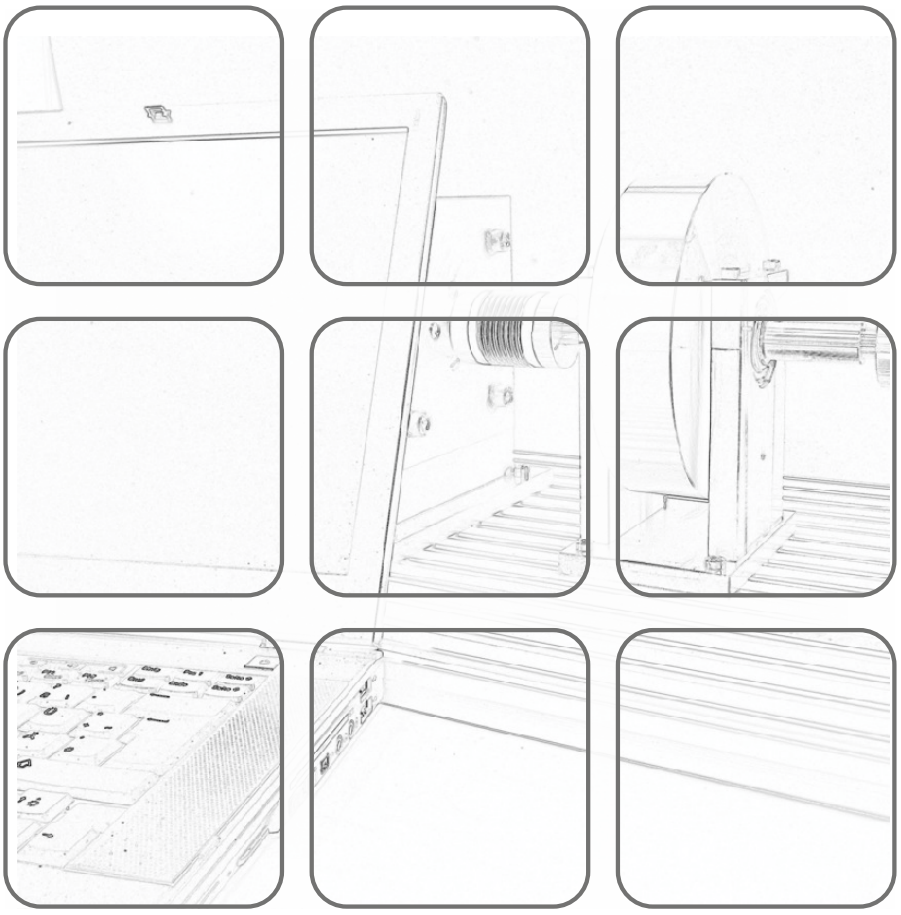
TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

FORTGESCHRITTENE METHODEN DER NICHTLINEAREN REGELUNG

Vorlesung und Übung
Wintersemester 2024/2025

A. Deutschmann-Olek, T. Glück, A. Kugi, M.N. Vu



Fortgeschrittene Methoden der nichtlinearen Regelung

Vorlesung und Übung
Wintersemester 2024/2025

A. Deutschmann-Olek, T. Glück, A. Kugi, M.N. Vu

TU Wien
Institut für Automatisierungs- und Regelungstechnik
Gruppe für komplexe dynamische Systeme

Gußhausstraße 27–29
1040 Wien
Telefon: +43 1 58801 – 37615
Internet: https://www.acin.tuwien.ac.at

Contents

1 Dissipativität und Passivität

Vereinfachend gesprochen, ist das Konzept der Dissipativität und Passivität die systemtheoretische Verallgemeinerung des Energieerhaltungsprinzips, welches besagt, dass in einem abgeschlossenen System Energie weder erzeugt noch vernichtet werden kann. Eine nähere Betrachtung des systemtheoretischen Konzeptes der Dissipativität wird jedoch zeigen, dass dies a priori mit dem Prinzip der Energieerhaltung nichts zu tun hat und lediglich bei gewissen physikalischen Systemen analoge Aussagen zulässt. Diese Analogie zu physikalischen Systemen trägt aber sicherlich zum Verständnis dieser Konzepte bei, weshalb im Folgenden zwei physikalische Systeme, ein Wärmeübertragungssystem und ein elektromechanisches System, diskutiert werden.

1.1 Glühsimulator

Abbildung 1.1 zeigt die schematische Darstellung eines so genannten Glühsimulators, der dazu verwendet wird, durch Ohmsches Erwärmen und freie bzw. erzwungene Konvektion (Pressluft oder Ventilator) für Metallproben vorgegebene Temperaturprofile abzufahren.

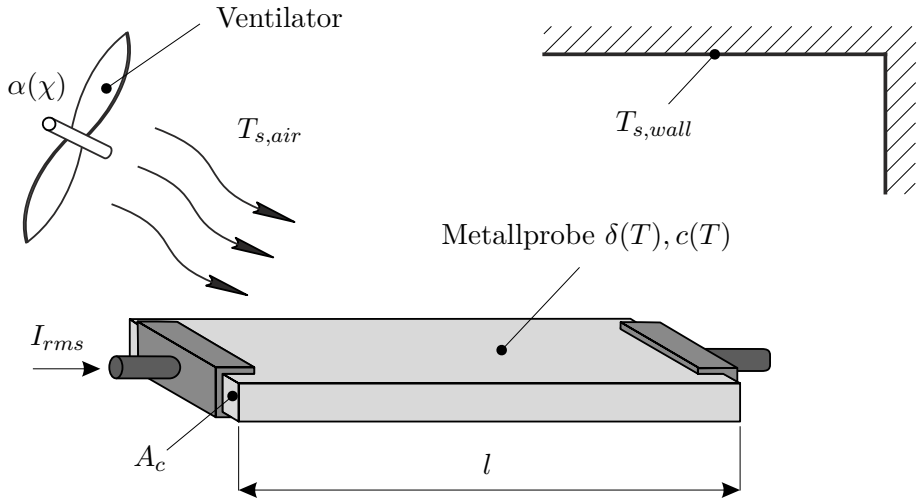


Figure 1.1: Glühsimulator.

Es ist naheliegend für dieses System die elektromechanischen Effekte zu vernachlässigen und die Änderung der im System gespeicherten Energie allein durch die Änderung der thermisch gespeicherten Energie zu erfassen. Das Energieerhaltungsprinzip besagt dann, dass die Änderung der thermisch gespeicherten Energie V der Beziehung

$$\frac{\mathrm{d}}{\mathrm{d}t}V = p_{in} - p_{out} \tag{1.1}$$

genügt, wobei p_{in} und p_{out} die Energieflüsse in das System und vom System beschreiben. Es wird angenommen, dass die Temperatur T in der Metallprobe zu jedem Zeitpunkt t gleichförmig verteilt ist, dass die Oberfläche der Probe sehr klein verglichen mit den umgebenden Wänden ist, und dass die Wärmeleitung vernachlässigt werden kann. Die in der Probe gespeicherte thermische Energie V lautet

$$V(T) = c(T)mT \tag{1.2}$$

mit der konstanten Probenmasse m und der spezifischen Wärmekapazität $c(T)$. Mit Hilfe des Ohmschen Gesetzes errechnet sich der Energiefluss in die Probe zu

$$p_{in} = I_{rms}^2 \delta(T) \frac{l}{A_c} \tag{1.3}$$

mit dem Effektivwert des durch die Probe fließenden Stromes I_{rms}, dem spezifischen Widerstand $\delta(T)$, der Länge der Probe l und der Probenquerschnittsfläche A_c. Die Energieflüsse von der Probe in die Umgebung werden einerseits durch die freie und erzwungene Konvektion

$$p_{out,1} = \alpha(\chi) A_s (T - T_{s,air}) \tag{1.4}$$

und andererseits durch die Wärmestrahlung

$$p_{out,2} = \varepsilon \sigma A_s \left(T^4 - T_{s,wall}^4 \right) \tag{1.5}$$

verursacht. Dabei bezeichnen A_s die Oberfläche der Metallprobe, $T_{s,air}$ und $T_{s,wall}$ die Temperaturen der umgebenden Luft und Wände, ε ist der Emissionsgrad, $\sigma = 5.67 \cdot 10^{-8}$ $\mathrm{Wm^{-2}K^{-4}}$ die Stefan-Boltzmann Konstante und $\alpha(\chi)$ ist der Konvektionskoeffizient, wobei χ im Falle eines Lüfters für die Drehwinkelgeschwindigkeit des Lüfters und im Falle von Druckluft für den Druck steht. Bei freier Konvektion ist $\alpha(\chi)$ konstant und liegt im Bereich von $2 - 25$ $\mathrm{Wm^{-2}K^{-1}}$. Das mathematische Modell des Glühsimulators erhält man einfach durch Einsetzen von (1.2) - (1.5) in (1.1) mit der Zustandsgröße T und den Eingangsgrößen $\mathbf{u}^{\mathrm{T}} = [I_{rms}, \chi, T_{s,air}, T_{s,wall}]$. Integriert man (1.1) entlang einer Lösungskurve vom Zeitpunkt $t_0 = 0$ zum Zeitpunkt t für gegebene Eingangsgrößen $\mathbf{u}(\tau)$, $0 \leq \tau \leq t$, dann erhält man

$$V(T(t)) - V(T(0)) = \int_0^t s(I_{rms}, \chi, T_{s,air}, T_{s,wall}, T) \mathrm{d}\tau \tag{1.6}$$

mit

$$s(I_{rms}, \chi, T_{s,air}, T_{s,wall}, T) = I_{rms}^2 \delta(T) \frac{l}{A_c} - \alpha(\chi) A_s (T - T_{s,air}) - \varepsilon \sigma A_s \left(T^4 - T_{s,wall}^4 \right) . \tag{1.7}$$

Gleichung (1.6) besagt, dass die zum Zeitpunkt t im System gespeicherte thermische Energie V gleich der zum Zeitpunkt $t_0 = 0$ gespeicherten Energie plus oder minus der in dieser Zeit mit der so genannten Versorgungsrate $s(I_{rms}, \chi, T_{s,air}, T_{s,wall}, T)$ dem System zu- oder abgeführten Energie ist.

1.2 Einfaches Elektromagnetventil

Abbildung 1.2 zeigt das Elektromagnetventil mit einem zylindrischen Gehäuse und einem zylindrischen Stössel mit der Masse m und dem Durchmesser D. Die aus N Windungen bestehende Spule mit einem gesamten Innenwiderstand R wird mit einer Spannung U_0 versorgt. Es wird angenommen, dass der magnetische Widerstand des Gehäuses und des Stössels Null ist, dass die Gleithülse die gleiche Permeabilität wie Luft besitzt und dass für die geometrischen Abmessungen gilt $h \ll D$ und $\delta \ll b$ (keine Streuflüsse).

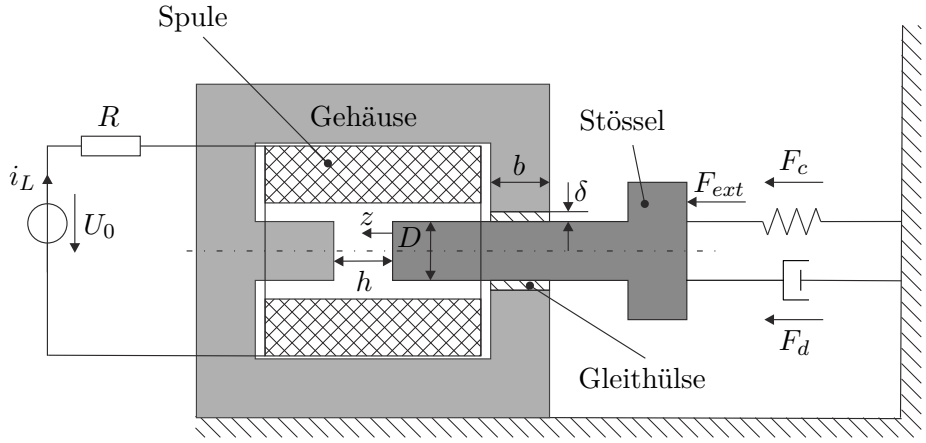


Figure 1.2: Einfaches Elektromagnetventil.

Auf analoge Art und Weise zu (1.1) gilt für die Änderung der im System gespeicherten Energie V die Beziehung

$$\frac{\mathrm{d}}{\mathrm{d}t}V = p_{in} - p_{out} - p_{diss} \tag{1.8}$$

mit den Energieflüssen p_{in} und p_{out} , die über die Systemgrenzen in das System bzw. vom System fließen und mit der in Wärme dissipierten Leistung p_{diss}.

Unter den obigen Voraussetzungen errechnet sich die im Magnetkreis gespeicherte Koenergie in der Form

$$\check{w}_L = \frac{1}{2}L(z)i_L^2 \tag{1.9}$$

mit der Ersatzinduktivität des magnetischen Kreises

$$L(z) = \frac{\mu_0 N^2 D^2 \pi (D+\delta)\pi b}{4(h-z)(D+\delta)\pi b + \delta D^2 \pi} \tag{1.10}$$

und der Permeabilität von Luft $\mu_0 = 4\pi \times 10^{-7}\,\mathrm{V\,s/(A\,m)}$.

Aufgabe 1.1. Rechnen Sie die Beziehung für die Induktivität $L(z)$ von (1.10) nach.

Da das betrachtete Elektromagnetventil magnetisch linear ist, sind die Ausdrücke für Energie $\hat{w}_L$ und Koenergie $\check{w}_L$ identisch. Die auf den Stössel wirkende Magnetkraft errechnet sich zu

$$F_{mag} = \frac{\partial}{\partial z}\check{w}_L = \frac{1}{2}\frac{\partial L(z)}{\partial z}i_L^2 \,. \tag{1.11}$$

Wie in Abbildung 1.2 gezeichnet, wirkt der Stössel gegen ein lineares Feder-Dämpfer System mit der Dämpfungskraft $F_d = dv,\ v = \dot{z},\ d > 0$, der Federkraft $F_c = cz(t),\ c > 0$ und einer externen Kraft F_{ext}. Das mathematische Modell des Elektromagnetventils lautet dann

$$\frac{\mathrm{d}}{\mathrm{d}t}z = v \tag{1.12}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}v = \frac{1}{m}\left(\frac{1}{2}\frac{\partial L(z)}{\partial z}i_L^2 - cz - dv + F_{ext}\right) \tag{1.13}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}i_L = \frac{1}{L(z)}\left(U_0 - Ri_L - \frac{\partial L(z)}{\partial z}i_L v\right) \tag{1.14}$$

mit den Zustandsgrößen $\mathbf{x}^{\mathrm{T}} = [z, v, i_L]$ und den Eingangsgrößen $\mathbf{u}^{\mathrm{T}} = [U_0, F_{ext}]$.

Die im System gespeicherte Energie setzt sich nun aus der magnetischen Energie (1.9), der kinetischen Energie des Stössels und der potenziellen Energie der Feder

$$V = \frac{1}{2}\left(L(z)i_L^2 + mv^2 + cz^2\right) \tag{1.15}$$

zusammen. Die Änderung der gespeicherten Energie V entlang einer Lösungskurve ergibt sich in der Form

$$\frac{\mathrm{d}}{\mathrm{d}t}V = \underbrace{U_0 i_L + F_{ext}v}_{p_{in}-p_{out}} - \underbrace{\left(dv^2 + Ri_L^2\right)}_{p_{diss}} . \tag{1.16}$$

Integriert man nun wieder (1.16) entlang einer Lösungskurve vom Zeitpunkt $t_0 = 0$ zum Zeitpunkt t für gegebene Eingangsgrößen $\mathbf{u}(\tau)$, $0 \le \tau \le t$, dann erhält man wegen $p_{diss} \ge 0$

$$V(\mathbf{x}(t)) - V(\mathbf{x}(t_0)) \le \int_{t_0}^{t} s(U_0, F_{ext}, i_L, v)\,\mathrm{d}\tau \tag{1.17}$$

mit der Versorgungsrate

$$s(U_0, F_{ext}, i_L, v) = U_0 i_L + F_{ext}v\ . \tag{1.18}$$

1.3 Systemtheoretisches Konzept

1.3.1 Dissipativität

Den nachfolgenden Betrachtungen liege ein nichtlineares dynamisches System der Form

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} &= \mathbf{f}(\mathbf{x}, \mathbf{u})\\ \mathbf{y} &= \mathbf{h}(\mathbf{x}, \mathbf{u})\end{aligned} \tag{1.19}$$

mit dem Zustand $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$, dem Stelleingang $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$ und dem Ausgang $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^p$ zu Grunde. Es sei angenommen, dass der Zustand $\mathbf{x}(t)$ zu jedem Zeitpunkt t eindeutig durch die Wahl der Eingangsgröße $\mathbf{u}(t)$ und des Anfangszustandes $\mathbf{x}(0) = \mathbf{x}_0$, bestimmt ist. Dies erlaubt es, die so genannte Versorgungsrate $s(\mathbf{u}, \mathbf{y}) : \mathcal{U} \times \mathcal{Y} \to \mathbb{R}$, eine

reellwertige Funktion, die für alle Anfangswerte $\mathbf{x}_0 \in \mathcal{X}$ und alle Eingangsgrößen $\mathbf{u}$ die Bedingung

$$\int_0^t |s(\mathbf{u}, \mathbf{y})| \mathrm{d}\tau < \infty \tag{1.20}$$

für alle Zeiten $t \geq 0$ erfüllt, einzuführen.

Definition 1.1. Das System (1.19) heißt *dissipativ bezüglich der Versorgungsrate* s, wenn eine nichtnegative Funktion $V(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$ so existiert, dass die so genannte *integrale Dissipativitätsungleichung*

$$V(\mathbf{x}(t)) - V(\mathbf{x}(0)) \leq \int_0^t s(\mathbf{u}(\tau), \mathbf{y}(\tau)) \mathrm{d}\tau \tag{1.21}$$

für alle Anfangswerte $\mathbf{x}(0) \in \mathcal{X}$ und alle Eingangsgrößen $\mathbf{u}(t)$ für alle Zeiten $t \geq 0$ erfüllt ist. Die Funktion $V(\mathbf{x})$ wird als *Speicherfunktion* bezeichnet. Falls in (1.21) das Gleichheitszeichen gilt, nennt man das System (1.19) *verlustlos bezüglich der Versorgungsrate* s.

Im Sinne dieser Definition ist der Glühsimulator von Abbildung 1.1 verlustlos bezüglich der Versorgungsrate (1.7) und das Elektromagnetventil von Abbildung 1.2 ist dissipativ bezüglich der Versorgungsrate (1.18). Wenn die Speicherfunktion $V(\mathbf{x})$ bezüglich $\mathbf{x}$ stetig differenzierbar ist, dann kann man die Änderung von $V(\mathbf{x})$ entlang einer Lösungskurve von (1.19) berechnen und man erhält die so genannte differenzielle Dissipativitätsungleichung

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}) \leq s(\mathbf{u}(t), \mathbf{y}(t)) \tag{1.22}$$

für alle Zeiten $t \geq 0$.

1.3.2 Passivität

Die Passivität kann als Spezialfall der Dissipativität aufgefasst werden. Zur Definition betrachte man wiederum das System (1.19), wobei nun die Dimension des Systemeingangs m gleich der Dimension des Ausgangs p ist.

Definition 1.2. Das System (1.19) mit $m = p$ nennt man *passiv*, wenn eine Konstante δ so existiert, dass die Ungleichung

$$\int_0^t \mathbf{y}^{\mathrm{T}} \mathbf{u} \mathrm{d}\tau \geq \delta \tag{1.23}$$

für alle zulässigen Eingangsgrößen $\mathbf{u}(t)$ und alle $t \geq 0$ erfüllt ist.
Wenn darüberhinaus für geeignete reelle Konstanten α, β die Ungleichung

$$\int_0^t \mathbf{y}^{\mathrm{T}} \mathbf{u} \mathrm{d}\tau \geq \delta + \alpha \int_0^t \mathbf{u}^{\mathrm{T}} \mathbf{u} \mathrm{d}\tau \quad \text{bzw.} \quad \int_0^t \mathbf{y}^{\mathrm{T}} \mathbf{u} \mathrm{d}\tau \geq \delta + \beta \int_0^t \mathbf{y}^{\mathrm{T}} \mathbf{y} \mathrm{d}\tau \tag{1.24}$$

für alle zulässigen Eingangsgrößen $\mathbf{u}(t)$ und alle $t \geq 0$ erfüllt ist, dann nennt man das System *α-eingangspassiv* bzw. *β-ausgangspassiv*.

Offensichtlich muss $\delta \leq 0$ gelten, denn die Ungleichung (1.23) muss auch für die Eingangsgröße $\mathbf{u}(t) = \mathbf{0}$ gültig sein.

Satz 1.1 (Verbindung Passivität und Dissipativität)**.** *Existiert nun für das System (1.19) mit $m = p$ eine nichtnegative Funktion $V(\mathbf{x}) : \mathcal{X} \to \mathbb{R}$ so, dass gilt (*integrale Passivitätsungleichung*)*

$$V(\mathbf{x}(t)) - V(\mathbf{x}(0)) \leq \int_0^t \mathbf{y}^{\mathrm{T}}\mathbf{u}\mathrm{d}\tau \tag{1.25}$$

für alle zulässigen Eingangsgrößen $\mathbf{u}(t)$, alle $\mathbf{x}(0)$ und alle $t \geq 0$, dann ist das System (1.19) vom Eingang $\mathbf{u}$ zum Ausgang $\mathbf{y}$ passiv. Offensichtlich ist dies gemäß Definition 1.1 äquivalent dazu, dass das System (1.19) bezüglich der speziellen bilinearen Versorgungsrate *$s(\mathbf{u}, \mathbf{y}) = \langle \mathbf{y}, \mathbf{u} \rangle = \mathbf{y}^{\mathrm{T}}\mathbf{u}$ dissipativ ist. Ist darüberhinaus das System (1.19) bezüglich der Versorgungsrate $s(\mathbf{u}, \mathbf{y}) = \mathbf{y}^{\mathrm{T}}\mathbf{u} - \alpha\|\mathbf{u}\|^2$ bzw. $s(\mathbf{u}, \mathbf{y}) = \mathbf{y}^{\mathrm{T}}\mathbf{u} - \beta\|\mathbf{y}\|^2$ für geeignete reelle Konstanten α, β dissipativ, so ist (1.19)* α-eingangspassiv *bzw.* β-ausgangspassiv. *Ein verlustloses passives System nennt man in diesem Zusammenhang auch ein* konservatives *System.*

Proof. Der Beweis des Satzes ist trivial, da wegen $V(\mathbf{x}) \geq 0$ aus (1.25) unmittelbar folgt

$$\int_0^t \mathbf{y}^{\mathrm{T}}\mathbf{u}\mathrm{d}\tau \geq -V(\mathbf{x}(0)) = \delta \ . \tag{1.26}$$

□

Mit dieser Definition erkennt man unmittelbar, dass das Elektromagnetventil von Abbildung 1.2 mit dem Eingang $\mathbf{u}^{\mathrm{T}} = [U_0, F_{ext}]$ und dem Ausgang $\mathbf{y}^{\mathrm{T}} = [i_L, v]$ passiv, ja sogar β-ausgangspassiv mit $0 < \beta < \min(d, R)$ ist, da für die dissipierte Leistung von (1.16) gilt $p_{diss} = dv^2 + Ri_L^2 \geq \beta\|\mathbf{y}\|^2$.

Die physikalische Interpretation der Passivitätsungleichung (1.25) lautet nun wie folgt: Gibt der Ausdruck $\mathbf{y}^{\mathrm{T}}\mathbf{u}$ eine Leistung an (z.B. geeignete Paare von Strömen und Spannungen bei elektrischen Systemen oder kollokierte Geschwindigkeiten und Kräfte bei mechanischen Systemen) und ist $V(\mathbf{x})$ die im System gespeicherte Energie, so besagt die Passivitätsungleichung (1.25), dass die Zunahme der im System gespeicherten Energie kleiner oder gleich der dem System zugeführten Energie ist.

Aufgabe 1.2. Zeigen Sie, dass der Integrator mit der Zustandsdarstellung

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}x &= u \\ y &= x \end{aligned} \tag{1.27}$$

passiv ist.

Aufgabe 1.3. Unter welchen Voraussetzungen an die Parameter σ_0, σ_1, σ_2, r_C, r_H und v_0 beschreibt das *LuGre-Reibmodell* (siehe z. B. Skriptum zur VO Regelungssysteme 2 [1.1]) ein passives System vom Eingang Δv zum Ausgang F_R. Zur Wiederholung soll das LuGre-Reibmodell nochmals in der Form

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} z &= \Delta v - \frac{\mathrm{abs}(\Delta v)}{\chi(\Delta v)} \sigma_0 z \\ F_R &= \sigma_0 z + \sigma_1 \frac{\mathrm{d}}{\mathrm{d}t} z + \sigma_2 \Delta v \end{aligned} \tag{1.28}$$

mit

$$\chi(\Delta v) = r_C + (r_H - r_C) \exp\left(-\left(\frac{\Delta v}{v_0} \right)^2 \right) \tag{1.29}$$

angeschrieben werden.

Aufgabe 1.4. Zeigen Sie, dass eine nichtlineare Kennlinie $y = \psi(u)$, die die Sektorbedingung $k_1 u^2 \leq \psi(u) u \leq k_2 u^2$ erfüllt, k_1-eingangspassiv und $\left(\frac{1}{k_2}\right)$-ausgangspassiv gemäß Definition 1.2 ist.

1.3.3 Eigenschaften Passiver Systeme

Passive Systeme haben nun die bemerkenswerte Eigenschaft, dass die Parallelschaltung und die Rückkopplung passiver Systeme, wie in Abbildung 1.3 dargestellt, wiederum passiv ist.

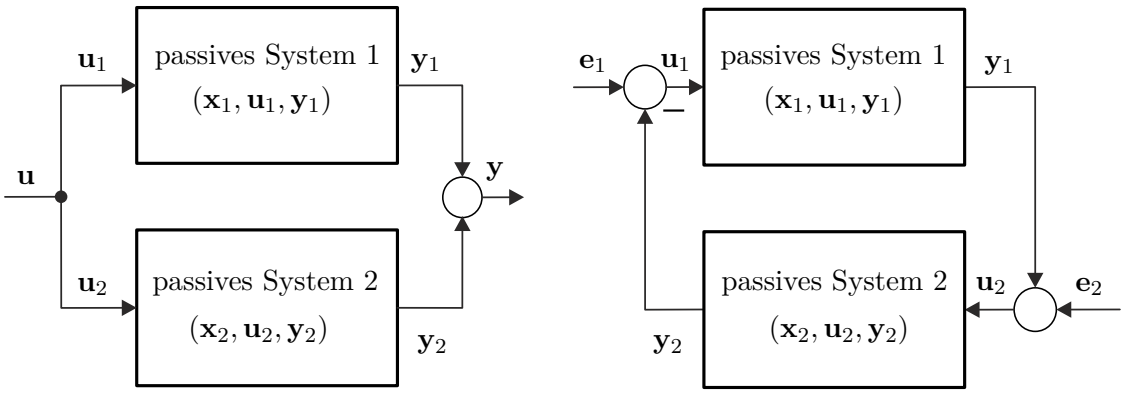


Figure 1.3: Parallelschaltung und Rückkopplung zweier passiver Systeme.

Proof. Um dies zu zeigen, nimmt man zwei passive Systeme der Form (1.19) mit $m = p$ an. Für diese existieren dann zwei nichtnegative Speicherfunktionen $V_1(\mathbf{x}_1)$

und $V_2(\mathbf{x}_2)$, die den Passivitätsungleichungen

$$\begin{aligned} V_1(\mathbf{x}_1(t)) - V_1(\mathbf{x}_1(0)) &\leq \int_0^t \mathbf{y}_1^{\mathrm{T}} \mathbf{u}_1 \, \mathrm{d}\tau \\ V_2(\mathbf{x}_2(t)) - V_2(\mathbf{x}_2(0)) &\leq \int_0^t \mathbf{y}_2^{\mathrm{T}} \mathbf{u}_2 \, \mathrm{d}\tau \end{aligned} \tag{1.30}$$

genügen. Für die Parallelschaltung nach Abbildung 1.3 gilt $\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{u}$, $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2$ und damit

$$V_1(\mathbf{x}_1(t)) + V_2(\mathbf{x}_2(t)) - V_1(\mathbf{x}_1(0)) - V_2(\mathbf{x}_2(0)) \leq \int_0^t \left(\mathbf{y}_1^{\mathrm{T}} + \mathbf{y}_2^{\mathrm{T}}\right) \mathbf{u} \mathrm{d}\tau \tag{1.31}$$

bzw.

$$V(\mathbf{x}(t)) - V(\mathbf{x}(0)) \leq \int_0^t \mathbf{y}^{\mathrm{T}} \mathbf{u} \mathrm{d}\tau \tag{1.32}$$

mit der nichtnegativen Speicherfunktion $V(\mathbf{x}) = V_1(\mathbf{x}_1) + V_2(\mathbf{x}_2)$ und dem Zustand $\mathbf{x}^{\mathrm{T}} = [\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}]$. □

Aufgabe 1.5. Zeigen Sie, dass der geschlossene Kreis der Rückkopplung zweier passiver Systeme (siehe Abbildung 1.3, rechtes Bild) vom Eingang ($\mathbf{e}_1$, $\mathbf{e}_2$) zum Ausgang ($\mathbf{y}_1$, $\mathbf{y}_2$) passiv ist.

Darüberhinaus ist auch die Hintereinanderschaltung zweier passiver Systeme gemäß Abbildung 1.4 passiv, sofern das Verbindungssystem energieerhaltend ist, d.h. folgende Zusammenschaltungsbedingung

$$\int_0^t \left(\mathbf{y}_1^{\mathrm{T}} \mathbf{u}_I + \mathbf{y}_2^{\mathrm{T}} \mathbf{y}_I\right) \mathrm{d}\tau = 0 \tag{1.33}$$

erfüllt ist.

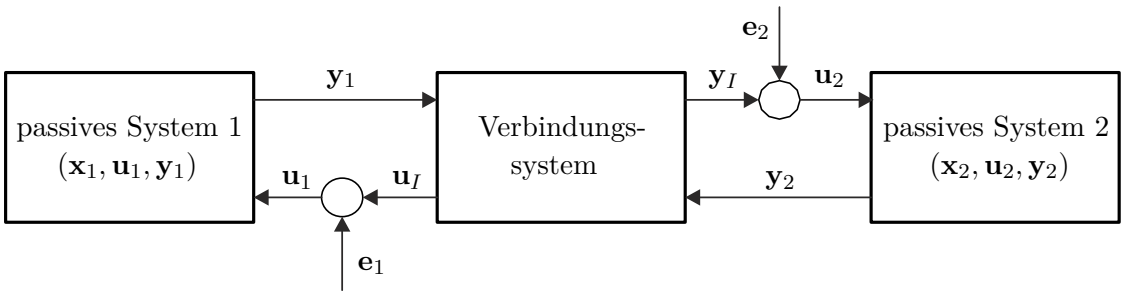


Figure 1.4: Hintereinanderschaltung passiver Systeme.

Man überzeugt sich leicht, dass dies der Fall ist, da die nachfolgende Passivitätsungleichung

$$V(\mathbf{x}(t)) - V(\mathbf{x}(0)) \leq \int_0^t \left(\mathbf{y}_1^{\mathrm{T}} \mathbf{e}_1 + \mathbf{y}_2^{\mathrm{T}} \mathbf{e}_2\right) \mathrm{d}\tau \tag{1.34}$$

mit $V(\mathbf{x}) = V_1(\mathbf{x}_1) + V_2(\mathbf{x}_2)$ und $\mathbf{x}^{\mathrm{T}} = [\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}]$ gilt. Gerade diese Eigenschaft wird bei gewissen passivitätsbasierten Reglerentwurfsverfahren genutzt, wobei das System 1 einer passiven Strecke und das System 2 einem passiven Regler entspricht. Für das Verbindungssystem wird in diesem Fall ein System der Form

$$\begin{bmatrix} \mathbf{u}_I \\ \mathbf{y}_I \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{U}_I(\mathbf{x}) \\ -\mathbf{U}_I^{\mathrm{T}}(\mathbf{x}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} \tag{1.35}$$

mit einer vorerst beliebigen quadratischen Matrix $\mathbf{U}_I(\mathbf{x})$ gewählt.

Aufgabe 1.6. Zeigen Sie, dass (1.35) die Zusammenschaltungsbedingung (1.33) erfüllt.

1.3.4 Passivität und Lyapunov-Stabilität

Es sei angenommen, dass das System (1.19) passiv mit einer stetig differenzierbaren, positiv definiten Speicherfunktion $V(\mathbf{x})$ ist. Dann folgt unmittelbar aus der Passivitätsungleichung (1.25) in ihrer differenziellen Form

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}) \leq \mathbf{y}^{\mathrm{T}} \mathbf{u}, \tag{1.36}$$

dass die Ruhelage $\mathbf{x} = \mathbf{0}$ des freien Systems (1.19), also für $\mathbf{u} = \mathbf{0}$, stabil im Sinne von Lyapunov ist mit der Lyapunovfunktion $V(\mathbf{x})$. Ob die Ruhelage asymptotisch stabil ist, muss von Fall zu Fall mit Hilfe des Invarianzprinzips von Krassovskii-LaSalle untersucht werden.

Für die Rückkopplung zweier passiver Systeme, wie sie im rechten Teil von Abbildung 1.3 gezeigt ist, kann die asymptotische Stabilität der Ruhelage des freien geschlossenen Kreises, also für $\mathbf{e}_1 = \mathbf{e}_2 = \mathbf{0}$, auf Eigenschaften der Teilsysteme zurückgeführt werden.

Satz 1.2. *Angenommen, die Ruhelage $\mathbf{x}_1 = \mathbf{0}$ des Teilsystems 1 ist asymptotisch stabil und α-eingangspassiv gemäß Definition 1.2 mit einer stetig differenzierbaren, positiv definiten Speicherfunktion $V_1(\mathbf{x}_1)$. Weiters sei das Teilsystem 2 nullzustandsermittelbar und β-ausgangspassiv gemäß Definition 1.2 mit einer stetig differenzierbaren, positiv definiten Speicherfunktion $V_2(\mathbf{x}_2)$. Die Ruhelage des geschlossenen Kreises $(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{0}, \mathbf{0})$ ist dann asymptotisch stabil, wenn $\alpha + \beta > 0$ gilt.*

Bevor dieser Satz gezeigt wird, sollen noch die Begriffe der Nullzustandsermittelbarkeit und Nullzustandsbeobachtbarkeit definiert werden.

Definition 1.3. Das System (1.19) heißt nullzustandsermittelbar (nullzustandsbeobachtbar), wenn aus $\mathbf{u}(t) = \mathbf{0}$ und $\mathbf{y}(t) = \mathbf{0}$ für alle Zeiten $t \geq 0$ folgt $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{0}$ ($\mathbf{x}(t) = \mathbf{0}$ für alle Zeiten $t \geq 0$).

Proof. Zum Beweis von Satz 1.2 wähle man als Lyapunovfunktion des geschlossenen Kreises $V(\mathbf{x}) = V_1(\mathbf{x}_1) + V_2(\mathbf{x}_2)$ und bilde deren zeitliche Ableitung

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}) \leq -(\alpha + \beta) \|\mathbf{y}_2\|^2 \,. \tag{1.37}$$

Da aber nach Satz 1.2 $\alpha + \beta > 0$ ist, folgt unmittelbar, dass die Ruhelage des geschlossenen Kreises $(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{0}, \mathbf{0})$ stabil im Sinne von Lyapunov ist. Aufgrund

der Nullzustandsermittelbarkeit des Teilsystems 2 und der asymptotischen Stabilität der Ruhelage $\mathbf{x}_1 = \mathbf{0}$ des Teilsystems 1 kann man zeigen, dass die größte positiv invariante Menge, die in $\mathcal{H} = \left\{ \mathbf{x} \in \mathcal{X} | \frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}) = 0 \right\}$ enthalten ist, der Ursprung $(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{0}, \mathbf{0})$ ist. Damit ist aber nach dem Invarianzprinzip von Krassovskii-LaSalle die Ruhelage des geschlossenen Kreises $(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{0}, \mathbf{0})$ asymptotisch stabil. □

Satz 1.2 wird im Zusammenhang mit dem Begriff der *absoluten Stabilität* benötigt, insbesondere zur Herleitung des *Kreis- und Popov-Kriteriums.*

1.4 Lineare passive Systeme

Für ein lineares zeitinvariantes System der Form

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} &= \mathbf{A}\mathbf{x} + \mathbf{b}u \\ y &= \mathbf{c}^{\mathrm{T}}\mathbf{x} + du \end{aligned} \tag{1.38}$$

lässt sich die Eigenschaft der Passivität auch an Hand der zugehörigen Übertragungsfunktion

$$G(s) = \frac{\hat{y}(s)}{\hat{u}(s)} = \mathbf{c}^{\mathrm{T}}(s\mathbf{E} - \mathbf{A})^{-1}\mathbf{b} + d \tag{1.39}$$

beurteilen. Ohne Einschränkung der Allgemeinheit werden hier nur Eingrößensysteme behandelt, für Mehrgrößensysteme sei auf die am Ende angeführte Literatur verwiesen. Gemäß Definition 1.2 ist das System (1.38) genau dann passiv, wenn folgende Ungleichung

$$\int_0^t y u \mathrm{d}\tau \geq 0 \tag{1.40}$$

erfüllt ist. Damit lässt sich folgender Satz für die Passivität linearer zeitinvarianter Eingrößensysteme angeben:

Satz 1.3. *Das lineare zeitinvariante System (1.38) mit der Übertragungsfunktion $G(s)$ von (1.39) ist*

(1) genau dann passiv, wenn gilt

$$\mathrm{Re}(G(\mathrm{I}\omega)) \geq 0 \quad \textit{für alle } \omega, \tag{1.41}$$

(2) genau dann α-eingangspassiv mit $\alpha > 0$, wenn gilt

$$\mathrm{Re}(G(\mathrm{I}\omega)) \geq \alpha > 0 \quad \textit{für alle } \omega \tag{1.42}$$

(3) und genau dann β-ausgangspassiv mit $\beta > 0$, wenn gilt

$$\mathrm{Re}(G(\mathrm{I}\omega)) \geq \beta |G(\mathrm{I}\omega)|^2 > 0 \quad \textit{für alle } \omega\ . \tag{1.43}$$

Man beachte, dass die überprüfung der Bedingungen (1.41) - (1.43) sehr einfach an Hand der Nyquist-Ortskurve von $G(s)$ möglich ist.

Proof. Zum Beweis dieses Satzes benötigt man das so genannte *Theorem von Parseval*. Bezeichnen $x(t)$ und $y(t)$ zwei quadratisch integrierbare Zeitfunktionen, also $x(t)$, $y(t) \in L_2(-\infty, \infty)$, und

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) \exp(-\mathrm{I}\omega t)\mathrm{d}t \quad \text{bzw.} \quad \hat{y}(\omega) = \int_{-\infty}^{\infty} y(t) \exp(-\mathrm{I}\omega t)\mathrm{d}t \tag{1.44}$$

seien die zugehörigen Fouriertransformierten, dann gilt für das innere Produkt

$$\int_{-\infty}^{\infty} x(t)y(t)\mathrm{d}t = \langle x, y\rangle = \langle \hat{x}, \hat{y}\rangle = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{x}(\omega)\hat{y}^*(\omega)\mathrm{d}\omega \ . \tag{1.45}$$

Aus (1.45) folgt dann unmittelbar die Beziehung

$$\|x\|_2 = \|\hat{x}\|_2 \ . \tag{1.46}$$

Um das Theorem von Parseval für den Beweis von Satz 1.3 anwenden zu können, wird der Abschneideoperator $(\ )_T$ in der Form

$$u_T(t) = \begin{cases} u(t) & \text{für} \quad t \le T \\ 0 & \text{für} \quad t > T \end{cases} \tag{1.47}$$

eingeführt. Weiters wird angenommen, dass die Zeitfunktionen $u(t)$ und $y(t)$ kausal sind, d.h. $u(t) = 0$ und $y(t) = 0$ für $t < 0$. Damit erhält man

$$\int_0^{\mathrm{T}} u(t)y(t)\mathrm{d}t = \int_{-\infty}^{\infty} u_T(t)y(t)\mathrm{d}t = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{u}_T(\omega)\hat{y}^*(\omega)\mathrm{d}\omega \tag{1.48}$$

bzw. mit $\hat{y}(\omega) = G(\mathrm{I}\omega)\hat{u}_T(\omega)$ ergibt sich

$$\begin{aligned} \int_0^{\mathrm{T}} u(t)y(t)\,\mathrm{d}t &= \frac{1}{2\pi}\int_{-\infty}^{\infty} G^*(\mathrm{I}\omega)\hat{u}_T(\omega)\hat{u}_T^*(\omega)\,\mathrm{d}\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} (\mathrm{Re}(G(\mathrm{I}\omega)) - \mathrm{I}\,\mathrm{Im}(G(\mathrm{I}\omega)))|\hat{u}_T(\omega)|^2\,\mathrm{d}\omega \ . \end{aligned} \tag{1.49}$$

Da die linke Seite von (1.49) rein reell ist, muss der Imaginärteil auf der rechten Seite verschwinden, und es gilt

$$\int_0^{\mathrm{T}} u(t)y(t)\mathrm{d}t = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{Re}(G(\mathrm{I}\omega))|\hat{u}_T(\omega)|^2\mathrm{d}\omega \ . \tag{1.50}$$

"⇐": Setzt man nun voraus, dass (1.42) gilt, dann folgt

$$\int_0^{\mathrm{T}} u(t)y(t)\mathrm{d}t \ge \frac{\alpha}{2\pi}\int_{-\infty}^{\infty} |\hat{u}_T(\omega)|^2\mathrm{d}\omega = \alpha\int_0^{\mathrm{T}} u^2(t)\mathrm{d}t \tag{1.51}$$

und damit nach Definition 1.2 die α-Eingangspassivität von (1.38).

"⇒": Umgekehrt, wenn das System (1.38) α-eingangspassiv ist, dann existiert ein $\alpha > 0$ so, dass die Ungleichung

$$\int_0^{\mathrm{T}} u(t)y(t)\mathrm{d}t \geq \alpha \int_0^{\mathrm{T}} u^2(t)\mathrm{d}t \tag{1.52}$$

erfüllt ist, bzw. mit Hilfe des Theorems von Parseval erhält man

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} (\mathrm{Re}(G(\mathrm{I}\omega)) - \alpha)|\hat{u}_T(\omega)|^2\mathrm{d}\omega \geq 0 \ . \tag{1.53}$$

Die Ungleichung (1.53) ist aber nur dann für alle Eingangsgrößen $u(t)$ gültig, wenn für alle ω gilt $\mathrm{Re}(G(\mathrm{I}\omega)) \geq \alpha$. Angenommen, es existiert ein ω_0 so, dass $\mathrm{Re}(G(\mathrm{I}\omega_0)) < \alpha$ ist, dann sieht man, dass für die Eingangsgröße $u(t) = U\sin(\omega_0 t)$ und hinreichend großes T die Ungleichung (1.53) nicht erfüllt ist. Damit ist aber Punkt (2) und für $\alpha = 0$ auch Punkt (1) von Satz 1.3 bewiesen.

Aufgabe 1.7. Beweisen Sie Punkt (3) von Satz 1.3.

□

Als einfaches Anwendungsbeispiel soll gezeigt werden, dass der PID-Regler

$$R(s) = V\frac{1+T_I s}{s}\frac{1+T_D s}{1+\alpha T_D s} \tag{1.54}$$

mit den positiven Parametern V, T_I, T_D und $0 < \alpha < 1$ passiv ist. Dazu berechne man einfach

$$\mathrm{Re}(R(\mathrm{I}\omega)) = \frac{V\left(T_I + T_D(1-\alpha) + \alpha T_D^2 T_I w^2\right)}{1+\alpha^2 T_D^2 w^2} > 0 \ . \tag{1.55}$$

Aufgabe 1.8. Zeigen Sie, dass ein PI-Regler passiv ist.

Aufgabe 1.9. Zeigen Sie, dass das lineare zeitinvariante System (1.38) mit der Übertragungsfunktion $G(s)$ von (1.39) passiv ist, wenn

$$|\arg(G(\mathrm{I}\omega))| \leq \frac{\pi}{2} \ . \tag{1.56}$$

Aufgabe 1.10. Betrachten Sie einen einschleifigen Standardregelkreis mit einer passiven Strecke $G(s)$ und einem α-eingangspassiven Regler $R(s)$ mit $\alpha > 0$. Zeigen Sie, dass der geschlossene Kreis BIBO-stabil ist.

Hinweis: Verwenden Sie dazu das Nyquistkriterium.

Aufgabe 1.11. Der Zusammenhang zwischen Strom $\hat{\imath}(x,s)$ und Spannung $\hat{u}(x,s)$ an der Stelle $x = 0$ und an der Stelle $x = l$ einer langen elektrischen Leitung mit dem Kapazitätsbelag c, dem Induktivitätsbelag l, dem Widerstandsbelag r und dem

Leitwertsbelag g lautet

$$\begin{bmatrix} \hat{u}(0,s) \\ \hat{\imath}(0,s) \end{bmatrix} = \begin{bmatrix} \cosh(\gamma(s)l) & Z_0(s)\sinh(\gamma(s)l) \\ \dfrac{1}{Z_0(s)}\sinh(\gamma(s)l) & \cosh(\gamma(s)l) \end{bmatrix} \begin{bmatrix} \hat{u}(l,s) \\ \hat{\imath}(l,s) \end{bmatrix}, \tag{1.57}$$

wobei $Z_0(s)$ den Wellenwiderstand und $\gamma(s)$ den Ausbreitungskoeffizienten

$$Z_0(s) = \sqrt{\frac{r+sl}{g+sc}} \quad \text{und} \quad \gamma(s) = \sqrt{(r+sl)(g+sc)} \tag{1.58}$$

bezeichnen. überprüfen Sie für verschiedene Lastimpedanzen $Z_L(s)$ mit

$$\hat{u}(l,s) = Z_L(s)\hat{\imath}(l,s) \tag{1.59}$$

die Passivität der Übertragungsfunktion $G(s) = \dfrac{\hat{u}(0,s)}{\hat{\imath}(0,s)}$.

1.5 Positive Reellheit

Bei linearen zeitinvarianten Systemen (1.38) wird an Stelle der Passivität sehr oft der Begriff der positiven Reellheit der zugehörigen übertragungsfunktion (1.39) verwendet. Ohne Beweis sei angemerkt, dass das System (1.38) genau dann passiv ist, wenn (1.39) positiv reell ist.

Satz 1.4. *Eine Übertragungsfunktion $G(s)$ ist genau dann positiv reell, wenn*

(1) $G(s)$ keine Pole in der rechten offenen s-Halbebene besitzt,

(2) $\mathrm{Re}(G(\mathrm{I}\omega)) \geq 0$ ist für alle ω, für die gilt, $\mathrm{I}\omega$ ist kein Pol von $G(s)$ und

(3) wenn $s = \mathrm{I}\omega_0$ ein Pol von $G(s)$ ist, dann ist dieser einfach und für endliches ω_0 muss das Residuum

$$\lim_{s \to \mathrm{I}\omega_0} (s - \mathrm{I}\omega_0) G(s) \tag{1.60}$$

positiv und reell sein. Ist ω_0 unendlich, dann muss der Grenzwert

$$\lim_{\omega \to \infty} \frac{G(\mathrm{I}\omega)}{\mathrm{I}\omega} \tag{1.61}$$

positiv und reell sein.

Man nennt $G(s)$ streng positiv reell, wenn $G(s-\delta)$ für ein geeignetes $\delta > 0$ positiv reell ist.

Aufgabe 1.12. Zeigen Sie, dass die Bedingungen

(1) die Graddifferenz zwischen Zähler- und Nennerpolynom von $G(s)$ sind $-1, 0$ oder 1 und

(2) $G(s)$ hat keine Nullstellen in der rechten offenen s-Halbebene

notwendig dafür sind, dass $G(s)$ positiv reell ist.

Aufgabe 1.13. Sind die nachfolgenden Übertragungsfunktionen

$$G_1(s) = -(s-3),\ G_2(s) = \frac{1}{s^2+2s+1},\ G_3(s) = \frac{s+1}{s^2+1},\ G_4(s) = \frac{s+10}{(s+1)(s+2)} \tag{1.62}$$

positiv reell?

Wie im nachfolgenden Satz gezeigt wird, hängt die positive Reellheit einer Übertragungsfunktion $G(s)$ eng mit der Lösbarkeit eines Gleichungssystems zusammen. Für den Beweis dieses Satzes sei auf die am Ende angeführte Literatur verwiesen.

Satz 1.5 (Kalman-Yakubovich-Popov (KYP)-Lemma)**.** *Gegeben ist das System (1.38), wobei angenommen wird, dass das Paar* $(\mathbf{A}, \mathbf{b})$ *erreichbar und das Paar* $\left(\mathbf{c}^{\mathrm{T}}, \mathbf{A}\right)$ *beobachtbar ist. Die Übertragungsfunktion (1.39) ist genau dann positiv reell (passiv), wenn ein Skalar* w*, ein Vektor* $\mathbf{m}$ *und eine positiv definite Matrix* $\mathbf{P}$ *so existieren, dass nachfolgende Bedingungen*

$$\begin{aligned} \mathbf{P}\mathbf{A} + \mathbf{A}^{\mathrm{T}}\mathbf{P} &= -\mathbf{m}\,\mathbf{m}^{\mathrm{T}} \\ \mathbf{P}\mathbf{b} &= \mathbf{c} - \mathbf{m}w \\ w^2 &= 2d \end{aligned} \tag{1.63}$$

erfüllt sind. Die Übertragungsfunktion (1.39) ist darüberhinaus genau dann streng positiv reell nach Satz 1.4, wenn Skalare w *und* $\varepsilon > 0$*, ein Vektor* $\mathbf{m}$ *und eine positiv definite Matrix* $\mathbf{P}$ *so existieren, dass nachfolgende Bedingungen*

$$\begin{aligned} \mathbf{P}\mathbf{A} + \mathbf{A}^{\mathrm{T}}\mathbf{P} &= -\mathbf{m}\,\mathbf{m}^{\mathrm{T}} - \varepsilon\mathbf{P} \\ \mathbf{P}\mathbf{b} &= \mathbf{c} - \mathbf{m}w \\ w^2 &= 2d \end{aligned} \tag{1.64}$$

erfüllt sind.

Aufgabe 1.14. Angenommen w, $\mathbf{m}$, $\mathbf{P} > \mathbf{0}$ und $\varepsilon > 0$ seien Lösungen von (1.64). Zeigen Sie, dass dann im Falle $d \neq 0$ die Riccati–Gleichung

$$\mathbf{P}\left(\frac{\varepsilon}{2}\mathbf{E} + \mathbf{A}\right) + \left(\frac{\varepsilon}{2}\mathbf{E} + \mathbf{A}^{\mathrm{T}}\right)\mathbf{P} + (\mathbf{c} - \mathbf{P}\mathbf{b})\frac{1}{2d}\left(\mathbf{c}^{\mathrm{T}} - \mathbf{b}^{\mathrm{T}}\mathbf{P}\right) = \mathbf{0} \tag{1.65}$$

erfüllt ist.

Als Anwendung des KYP Lemmas betrachte man den geschlossenen Regelkreis von Abbildung 1.5 mit der nichtlinearen passiven Strecke im Vorwärtszweig und dem streng positiv reellen Regler im Rückwärtszweig.

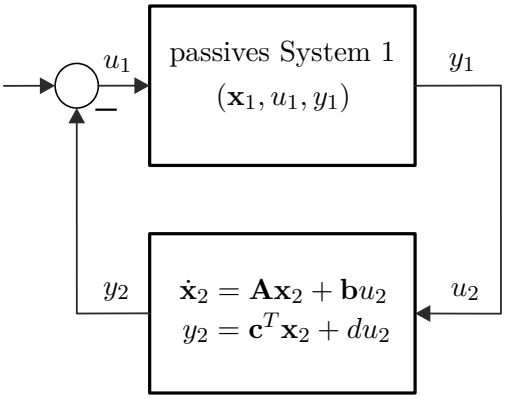


Figure 1.5: Passives System mit linearem Regler.

Angenommen das passive nichtlineare System habe eine stetig differenzierbare, positiv definite Speicherfunktion $V_1(\mathbf{x}_1)$, die der differenziellen Passivitätsungleichung (siehe (1.36))

$$\frac{\mathrm{d}}{\mathrm{d}t}V_1(\mathbf{x}_1) = -W_1(\mathbf{x}_1) + y_1u_1 \leq y_1u_1 \ , \tag{1.66}$$

mit der positiv semidefiniten Funktion $W_1(\mathbf{x}_1)$ genügt. Für das Weitere sei der streng positiv reelle Regler durch folgende Minimalrealisierung

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}_2 &= \mathbf{A}\mathbf{x}_2 + \mathbf{b}u_2 \\ y_2 &= \mathbf{c}^{\mathrm{T}}\mathbf{x}_2 + du_2 \end{aligned} \tag{1.67}$$

beschrieben. Aufgrund des KYP Lemmas Satz 1.5 findet man für das System (1.67) Skalare w und $\varepsilon > 0$, einen Vektor $\mathbf{m}$ und eine positiv definite Matrix $\mathbf{P}$ so, dass (1.64) erfüllt ist. Damit ergibt sich die *Lyapunov-Funktion des geschlossenen Kreises* von Abbildung 1.5 zu

$$V_e(\mathbf{x}_1, \mathbf{x}_2) = V_1(\mathbf{x}_1) + \frac{1}{2}\mathbf{x}_2^{\mathrm{T}}\mathbf{P}\mathbf{x}_2 \ . \tag{1.68}$$

Um dies zu zeigen, berechnet man die zeitliche Änderung von (1.68) entlang der Lösungskurve und berücksichtigt die Zusammenschaltungsbedingung $u_1 = -y_2$ und $u_2 = y_1$ gemeinsam mit (1.64) und (1.66)

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}V_e(\mathbf{x}_1,\mathbf{x}_2) &= -W_1(\mathbf{x}_1) + y_1u_1 + \frac{1}{2}\underbrace{\dot{\mathbf{x}}_2^{\mathrm{T}}\mathbf{P}\mathbf{x}_2}_{(\mathbf{x}_2^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}+u_2\mathbf{b}^T)\mathbf{P}\mathbf{x}_2} + \frac{1}{2}\underbrace{\mathbf{x}_2^{\mathrm{T}}\mathbf{P}\dot{\mathbf{x}}_2}_{\mathbf{x}_2^{\mathrm{T}}\mathbf{P}(\mathbf{A}\mathbf{x}_2+\mathbf{b}u_2)} \\
&= -W_1(\mathbf{x}_1) + y_1u_1 + \frac{1}{2}\mathbf{x}_2^{\mathrm{T}}\underbrace{\left(\mathbf{A}^{\mathrm{T}}\mathbf{P}+\mathbf{P}\mathbf{A}\right)}_{-\mathbf{m}\mathbf{m}^{\mathrm{T}}-\varepsilon\mathbf{P}}\mathbf{x}_2 + \mathbf{x}_2^{\mathrm{T}}\underbrace{\mathbf{P}\mathbf{b}}_{\mathbf{c}-\mathbf{m}w}u_2 \\
&= -W_1(\mathbf{x}_1) - \underbrace{y_1\mathbf{c}^{\mathrm{T}}\mathbf{x}_2}_{=} - dy_1^2 - \frac{1}{2}\mathbf{x}_2^{\mathrm{T}}\mathbf{m}\mathbf{m}^{\mathrm{T}}\mathbf{x}_2 - \frac{1}{2}\varepsilon\mathbf{x}_2^{\mathrm{T}}\mathbf{P}\mathbf{x}_2 + \underbrace{\mathbf{x}_2^{\mathrm{T}}\mathbf{c}y_1}_{=} - \mathbf{x}_2^{\mathrm{T}}\mathbf{m}wy_1 \\
&= -W_1(\mathbf{x}_1) - \frac{1}{2}\varepsilon\mathbf{x}_2^{\mathrm{T}}\mathbf{P}\mathbf{x}_2 - \frac{1}{2}y_1^2\underbrace{(2d)}_{w^2} - \frac{1}{2}\mathbf{x}_2^{\mathrm{T}}\mathbf{m}\mathbf{m}^{\mathrm{T}}\mathbf{x}_2 - \mathbf{x}_2^{\mathrm{T}}\mathbf{m}wy_1 \\
&= -W_1(\mathbf{x}_1) - \frac{1}{2}\varepsilon\mathbf{x}_2^{\mathrm{T}}\mathbf{P}\mathbf{x}_2 - \frac{1}{2}\left(\mathbf{m}^{\mathrm{T}}\mathbf{x}_2 + wy_1\right)^{\mathrm{T}}\left(\mathbf{m}^{\mathrm{T}}\mathbf{x}_2 + wy_1\right) \leq 0.
\end{aligned}
\tag{1.69}
$$

Dies zeigt unmittelbar die Stabilität des geschlossenen Kreises von Abbildung 1.5.

1.6 Kanonische Form Passiver Systeme

Bevor eine kanonische Form für passive Systeme vorgestellt wird, soll gezeigt werden, dass die wohlbekannten Euler-Lagrange Gleichungen passiv sind.

1.6.1 Hamiltonsche Systeme

Betrachtet man ein endlich-dimensionales Lagrangesches System mit n Freiheitsgraden und den generalisierten Koordinaten $\mathbf{q} \in \mathbb{R}^n$, dann folgen bekannterweise die Bewegungsgleichungen aus den Euler-Lagrange Gleichungen in der Form

$$
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_k}\right) - \frac{\partial L}{\partial q_k} = \tau_k \ , \quad k = 1,\ldots,n \tag{1.70}
$$

mit der Lagrangefunktion $L(\mathbf{q},\mathbf{v})$, den generalisierten Geschwindigkeiten $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q} = \mathbf{v}$ und den generalisierten Kräften τ_k, $k = 1,\ldots,n$. Bei einfachen Lagrangeschen Systemen entspricht die Langrangefunktion der Differenz aus kinetischer und potenzieller Energie

$$
L(\mathbf{q},\mathbf{v}) = T(\mathbf{q},\mathbf{v}) - V(\mathbf{q}) \ . \tag{1.71}
$$

Es sei angenommen, dass sich die generalisierten Kräfte $\boldsymbol{\tau}$ aus externen Kräften $\boldsymbol{\tau}_e$ (Stell- und Störeingänge im regelungstechnischen Sinne) und dissipativen Kräften $\boldsymbol{\tau}_d^{\mathrm{T}} = -\left(\frac{\partial}{\partial\mathbf{v}}R\right)(\mathbf{v})$ mit der *Rayleighsche Dissipationsfunktion* $R(\mathbf{v})$ und

$$
\left(\frac{\partial}{\partial\mathbf{v}}R\right)(\mathbf{v})\mathbf{v} \geq 0 \tag{1.72}
$$

zusammensetzen. Damit ergibt sich (1.70) zu

$$
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_k}\right) - \frac{\partial L}{\partial q_k} + \frac{\partial}{\partial v_k}R = \tau_{e,k} \ , \quad k = 1,\ldots,n \ . \tag{1.73}
$$

Definition 1.4. Man bezeichnet das Lagrangesche System (1.73) *voll gedämpft*, wenn die Rayleighsche Dissipationsfunktion $R(\mathbf{v})$ folgender Ungleichung

$$\left(\frac{\partial}{\partial \mathbf{v}} R\right)(\mathbf{v})\mathbf{v} \geq \sum_{k=1}^{n} \beta_k v_k^2, \qquad \beta_k > 0, \quad k = 1, \ldots, n \tag{1.74}$$

genügt. Ist ein $\beta_k = 0$, dann spricht man auch von einem *nicht voll gedämpften* Lagrangeschen System.

Mit Hilfe der generalisierten Impulskoordinaten

$$p_k = \frac{\partial L}{\partial v_k}, \quad k = 1, \ldots, n \tag{1.75}$$

und der *Legendre-Transformation* $(q_k, v_k) \to (q_k, p_k)$ erhält man direkt aus den Euler-Lagrange Gleichungen (1.70) die *äquivalenten Hamiltonschen Gleichungen*

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} q_k &= \frac{\partial H}{\partial p_k} \\ \frac{\mathrm{d}}{\mathrm{d}t} p_k &= -\frac{\partial H}{\partial q_k} + \tau_k, \quad k = 1, \ldots, n \end{aligned} \tag{1.76}$$

mit der Hamiltonfunktion

$$H(\mathbf{q}, \mathbf{p}) = \sum_{k=1}^{n} p_k v_k - L(\mathbf{q}, \mathbf{v}) \ . \tag{1.77}$$

Der Satz über implizite Funktionen besagt, dass die generalisierten Geschwindigkeiten v_k aus (1.75) genau dann lokal berechnet werden können, wenn die Matrix $\left[\frac{\partial^2}{\partial v_i \partial v_j} L\right]$ regulär ist. Man spricht dann auch von einer *nichtdegenerierten Lagrangefunktion* L.

Proof. Zum Beweis betrachte man die kurzen Ableitungen

$$\frac{\partial H}{\partial p_k} = v_k + \sum_{j=1}^{n} \left(p_j \frac{\partial v_j}{\partial p_k} - \underbrace{\frac{\partial L}{\partial v_j}}_{=p_j} \frac{\partial v_j}{\partial p_k} \right) = v_k = \frac{\mathrm{d}}{\mathrm{d}t} q_k \tag{1.78}$$

und

$$\frac{\partial H}{\partial q_k} = \sum_{j=1}^{n} \left(p_j \frac{\partial v_j}{\partial q_k} - \underbrace{\frac{\partial L}{\partial v_j}}_{=p_j} \frac{\partial v_j}{\partial q_k} \right) - \frac{\partial L}{\partial q_k} = \tau_k - \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial v_k}\right) = \tau_k - \frac{\mathrm{d}}{\mathrm{d}t} p_k \ . \tag{1.79}$$

□

Wenn die kinetische Energie $T(\mathbf{q}, \mathbf{v})$ in (1.71) die Form

$$T(\mathbf{q}, \mathbf{v}) = \frac{1}{2} \mathbf{v}^{\mathrm{T}} \mathbf{D}(\mathbf{q}) \mathbf{v} \tag{1.80}$$

mit der positiv definiten Massenmatrix $\mathbf{D}(\mathbf{q})$ hat, dann entspricht die Hamiltonfunktion

$$H(\mathbf{q},\mathbf{p}) = \sum_{k=1}^{n} p_k v_k - \frac{1}{2}\mathbf{v}^{\mathrm{T}}\mathbf{D}(\mathbf{q})\mathbf{v} + V(\mathbf{q}) = \frac{1}{2}\mathbf{v}^{\mathrm{T}}\mathbf{D}(\mathbf{q})\mathbf{v} + V(\mathbf{q}) \tag{1.81}$$

der im System gespeicherten Energie. Berechnet man die zeitliche Änderung der Hamiltonfunktion (1.81)

$$\frac{\mathrm{d}}{\mathrm{d}t}H(\mathbf{q},\mathbf{p}) = \sum_{k=1}^{n}\left[\frac{\partial H}{\partial q_k}\frac{\partial H}{\partial p_k} + \underbrace{\frac{\partial H}{\partial p_k}}_{v_k}\left(-\frac{\partial H}{\partial q_k} - \frac{\partial}{\partial v_k}R + \tau_{e,k}\right)\right] \leq \sum_{k=1}^{n} v_k\tau_{e,k}, \tag{1.82}$$

dann sieht man, dass das Lagrangesche System gemäß Definition 1.2 passiv ist mit der Eingangsgröße $\boldsymbol{\tau}_e$, der Ausgangsgröße $\mathbf{v} = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}$ und der Speicherfunktion $H(\mathbf{q},\mathbf{p})$. Ist darüberhinaus das Lagrangesche System gemäß Definition 1.4 voll gedämpft, dann ist das Lagrangesche System wegen (1.74) sogar β-ausgangspassiv mit $\beta = \min_k(\beta_k)$, $k = 1,\ldots,n$, da gilt

$$\frac{\mathrm{d}}{\mathrm{d}t}H(\mathbf{q},\mathbf{p}) \leq \sum_{k=1}^{n} v_k\tau_{e,k} - \sum_{k=1}^{n}\beta_k v_k^2 \leq \sum_{k=1}^{n} v_k\tau_{e,k} - \min_k(\beta_k)\|\mathbf{v}\|_2^2\,. \tag{1.83}$$

Man sagt dann auch, dass v_k der zur generalisierten Kraft $\tau_{e,k}$ *kollokierte Ausgang* ist. D.h., die Paarung $(\tau_{e,k},\, v_k)$ beschreibt einen Energieeingang in das System, wie z.B. zusammengehörende Strom und Spannungen, Kräfte und Geschwindigkeiten oder Momente und Drehwinkelgeschwindigkeiten. Im Rahmen der Netzwerkstheorie werden solche Paarungen von Strom und Spannung, die einen Energieeingang bilden, auch als *Tor* (im Englischen port) bezeichnet. Die Generalisierung der Hamiltonschen Gleichungen (1.76) in Kombination mit dem Torkonzept führt direkt zur Klasse der *Port-Hamiltonschen Systeme*.

1.6.2 Port-Hamiltonsche Systeme

Ein finit-dimensionales Port-Hamiltonsches System lässt sich in der Form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\mathbf{J}(\mathbf{x}) - \mathbf{S}(\mathbf{x}))\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \mathbf{G}_u(\mathbf{x})\mathbf{u} \tag{1.84}$$

mit dem Zustand $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ und dem Stelleingang $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$ formulieren. Dabei bezeichnet $V(\mathbf{x})$, $V(\mathbf{0}) = 0$, eine stetig differenzierbare positiv definite Speicherfunktion und die Einträge der Matrizen $\mathbf{G}_u(\mathbf{x})$, $\mathbf{J}(\mathbf{x}) = -\mathbf{J}^{\mathrm{T}}(\mathbf{x})$ und $\mathbf{S}(\mathbf{x}) = \mathbf{S}^{\mathrm{T}}(\mathbf{x}) \geq \mathbf{0}$ seien glatte Funktionen in $\mathbf{x}$. Wählt man als Ausgang $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^m$ den *kollokierten Ausgang*

$$\mathbf{y} = \mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}}, \tag{1.85}$$

dann erkennt man unmittelbar aus der differenziellen Passivitätsungleichung

$$\frac{\mathrm{d}}{\mathrm{d}t}V = \mathbf{y}^{\mathrm{T}}\mathbf{u} - \left(\frac{\partial V}{\partial \mathbf{x}}\right)\mathbf{S}(\mathbf{x})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} \leq \mathbf{y}^{\mathrm{T}}\mathbf{u}, \tag{1.86}$$

dass das System (1.84) passiv ist mit der Speicherfunktion $V(\mathbf{x})$. Die Darstellung in der Form von (1.84) erlaubt mehr als nur die einfache Feststellung der Passivität – sie ermöglicht, falls die Speicherfunktion $V(\mathbf{x})$ gleich der im System gespeicherten Gesamtenergie ist, einen tieferen Einblick in die Energieflüsse des Systems im Inneren und mit der Systemumgebung: Die schiefsymmetrische Matrix $\mathbf{J}(\mathbf{x})$ ist nämlich mit den Energieflüssen im Systeminneren verbunden, die symmetrische, positiv semidefinite Matrix $\mathbf{S}(\mathbf{x})$ umfasst das Verhalten der dissipativen Effekte und $\mathbf{G}_u(\mathbf{x})$ beschreibt den Energieaustausch des Systems mit der Systemumgebung über die Systemtore. Wenn (1.84) keine dissipativen Elemente enthält, also $\mathbf{S}(\mathbf{x}) = \mathbf{0}$ ist, dann ist das System verlustlos bezüglich der Versorgungsrate $\mathbf{y}^{\mathrm{T}}\mathbf{u}$.

Eine perfekte Aktuator/Sensor Kollokation bringt den Vorteil mit sich, dass eine einfache (zustandsabhängige) Rückführung des kollokierten Ausgangs (1.85) der Form

$$\mathbf{u} = -\mathbf{K}(\mathbf{x})\mathbf{y} = -\mathbf{K}(\mathbf{x})\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}}, \tag{1.87}$$

mit der positiv definiten Matrix $\mathbf{K}(\mathbf{x}) > 0$ für alle $\mathbf{x} \in \mathcal{X}$ bei stabilen Strecken die Stabilität im geschlossenen Kreis erhält, da gilt

$$\frac{\mathrm{d}}{\mathrm{d}t}V = -\left(\frac{\partial V}{\partial \mathbf{x}}\right)\Big(\mathbf{S}(\mathbf{x}) + \mathbf{G}_u(\mathbf{x})\mathbf{K}(\mathbf{x})\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\Big)\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} \leq 0\ . \tag{1.88}$$

In der Literatur wird diese Art der Rückführung (1.87) im Zusammenhang mit Port-Hamiltonschen Systemen als *damping injection* bezeichnet oder bei allgemeinen nichtlinearen Systemen mit affinem Eingang als *Jurdjevic-Quinn Rückführung.*

Beispiel 1.1 (Port-Hamiltonsche Darstellung des Elektromagnetventils (1.14)). Um das mathematische Modell des Elektromagnetventils (1.14) in Port-Hamiltonsche Darstellung (1.84) zu bringen, führt man die neuen Zustandsgrößen $\mathbf{x}^{\mathrm{T}} = [z, p = mv, \psi_L = L(z)i_L]$ ein. Die im Magnetventil gespeicherte Energie gemäß (1.15) formuliert im neuen Zustand $[z, p, \psi_L]$

$$V = \frac{1}{2}\left(\frac{1}{L(z)}\psi_L^2 + \frac{1}{m}p^2 + cz^2\right) \tag{1.89}$$

wird in weiterer Folge als Speicherfunktion verwendet. Mit

$$\frac{\partial V}{\partial \mathbf{x}} = \begin{bmatrix} cz - \frac{1}{2}\frac{\partial L(z)}{\partial z}\frac{\psi_L^2}{L^2(z)} & \frac{p}{m} & \frac{\psi_L}{L(z)} \end{bmatrix} \tag{1.90}$$

und den Systemgleichungen (1.14) im transformierten Zustand

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}z &= \frac{p}{m} \\ \frac{\mathrm{d}}{\mathrm{d}t}p &= \left(\frac{1}{2}\frac{\partial L(z)}{\partial z}\frac{\psi_L^2}{L^2(z)} - cz - d\frac{p}{m} + F_{ext}\right) \\ \frac{\mathrm{d}}{\mathrm{d}t}\psi_L &= U_0 - R\frac{\psi_L}{L(z)} \end{aligned} \tag{1.91}$$

ergibt sich unmittelbar die Port-Hamiltonsche Darstellung (1.84) zu

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} z \\ p \\ \psi_L \end{bmatrix} = \left(\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathbf{J}(\mathbf{x})} - \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & R \end{bmatrix}}_{\mathbf{S}(\mathbf{x})} \right) \left(\frac{\partial V}{\partial \mathbf{x}} \right)^{\mathrm{T}} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}}_{\mathbf{G}_u(\mathbf{x})} \underbrace{\begin{bmatrix} U_0 \\ F_{ext} \end{bmatrix}}_{\mathbf{u}} . \tag{1.92}$$

Der zugehörige kollokierte Ausgang gemäß (1.85) lautet

$$\mathbf{y} = \mathbf{G}_u^{\mathrm{T}}(\mathbf{x}) \left(\frac{\partial V}{\partial \mathbf{x}} \right)^{\mathrm{T}} = \begin{bmatrix} \psi_L \\ \dfrac{\psi_L}{L(z)} \\ \dfrac{p}{m} \end{bmatrix} = \begin{bmatrix} i_L \\ v \end{bmatrix} . \tag{1.93}$$

Aufgabe 1.15. Stellen Sie die mathematischen Modelle des Balkens mit rollender Kugel und des Krans mit einem Schwenkarm aus dem Skriptum zur VO Regelungssysteme 2 [1.1] als Port-Hamiltonsche Systeme dar.

Aufgabe 1.16. Stellen Sie die unterschiedlichen Gleichstrommaschinen vom Abschnitt 1.7 aus dem Skriptum zur VO Regelungssysteme 2 [1.1] als Port-Hamiltonsche Systeme dar.

1.7 Passivitätsbasierter Reglerentwurf

Ein mit der Port-Hamiltonschen Struktur (1.84) unmittelbar verbundenes Reglerentwurfsverfahren ist die so genannte *IDA-PBC (Interconnection and Damping Assignment Passivity-Based Control)*. Dazu sei folgender Satz formuliert:

Satz 1.6 (IDA-PBC). *Gegeben ist das nichtlineare System*

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}) + \mathbf{G}_u(\mathbf{x})\mathbf{u} \tag{1.94}$$

mit dem Zustand $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ *und dem Stelleingang* $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$ *mit* $m < n$. *Von der Matrix* $\mathbf{G}_u(\mathbf{x})$ *wird vorausgesetzt, dass diese für alle* $\mathbf{x} \in \mathcal{X}$ *spaltenregulär ist, d.h.* $\operatorname{rang}(\mathbf{G}_u(\mathbf{x})) = m$. *Im Weiteren bezeichne* $\mathbf{G}_u^{\perp}(\mathbf{x})$ *den* Linksannihilator *von* $\mathbf{G}_u(\mathbf{x})$, *d.h.* $\mathbf{G}_u^{\perp}(\mathbf{x})\mathbf{G}_u(\mathbf{x}) = \mathbf{0}$, *und* $V_d(\mathbf{x})$ *sei die Speicherfunktion des geschlossenen Kreises und habe an der gewünschten Ruhelage* $\mathbf{x} = \mathbf{x}_d$ *ein striktes Minimum, d.h.*

$$V_d(\mathbf{x}) > V_d(\mathbf{x}_d) \quad \textit{für alle} \quad \mathbf{x} \neq \mathbf{x}_d, \quad \left(\frac{\partial V_d}{\partial \mathbf{x}} \right)(\mathbf{x}_d) = \mathbf{0} \quad \textit{und} \quad \left(\frac{\partial^2 V_d}{\partial \mathbf{x}^2} \right)(\mathbf{x}_d) > 0 \ . \tag{1.95}$$

*Damit ist $V_d(\mathbf{x}) - V_d(\mathbf{x}_d)$ positiv definit und eignet sich als Lyapunovfunktion für den geschlossenen Kreis. Angenommen die Matrizen $\mathbf{J}_d(\mathbf{x}) = -\mathbf{J}_d^{\mathrm{T}}(\mathbf{x})$, $\mathbf{S}_d(\mathbf{x}) = \mathbf{S}_d^{\mathrm{T}}(\mathbf{x}) \geq 0$, der Linksannihilator $\mathbf{G}_u^{\perp}(\mathbf{x})$ und die Speicherfunktion $V_d(\mathbf{x})$ genügen der Bedingung (*PBC matching equation*)*

$$\mathbf{G}_u^{\perp}(\mathbf{x})\mathbf{f}(\mathbf{x}) = \mathbf{G}_u^{\perp}(\mathbf{x})(\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}}, \tag{1.96}$$

dann ergibt sich mit der Zustandsrückführung

$$\mathbf{u} = \boldsymbol{\beta}(\mathbf{x}) = \left(\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\mathbf{G}_u(\mathbf{x})\right)^{-1}\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\left\{(\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} - \mathbf{f}(\mathbf{x})\right\} \tag{1.97}$$

eingesetzt in (1.94) ein geschlossener Kreis in Port-Hamiltonscher Form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} \tag{1.98}$$

mit der stabilen gewünschten Ruhelage des geschlossenen Kreises $\mathbf{x} = \mathbf{x}_d$. Wenn die Menge $\{\mathbf{x}_d\}$ die größte positiv invariante Menge von

$$\left\{\mathbf{x} \in \mathbb{R}^n \middle| \left(\frac{\partial V_d}{\partial \mathbf{x}}\right)\mathbf{S}_d(\mathbf{x})\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} = 0\right\} \tag{1.99}$$

ist, dann ist $\mathbf{x} = \mathbf{x}_d$ sogar asymptotisch stabil.

Proof. Setzt man die rechten Seiten von (1.98) und (1.94) mit (1.97) gleich, d.h.

$$\mathbf{f}(\mathbf{x}) + \mathbf{G}_u(\mathbf{x})\boldsymbol{\beta}(\mathbf{x}) = (\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}}, \tag{1.100}$$

und multipliziert man mit $\mathbf{G}_u^{\perp}(\mathbf{x})$ von links, so erhält man unmittelbar die PBC matching equation (1.96). Die Zustandsrückführung (1.97) folgt direkt aus (1.100) durch Multiplikation mit der Pseudoinversen $\left(\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\mathbf{G}_u(\mathbf{x})\right)^{-1}\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})$ von links. Man beachte, dass die zuvor angenommene Spaltenregularität von $\mathbf{G}_u(\mathbf{x})$ die Regularität der Pseudoinversen garantiert. Im nächsten Schritt soll gezeigt werden, dass (1.94) mit (1.97) tatsächlich (1.98) entspricht, also gilt

$$\begin{aligned}\boldsymbol{\Psi} = \mathbf{f}(\mathbf{x}) + \mathbf{G}_u(\mathbf{x})\left(\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\mathbf{G}_u(\mathbf{x})\right)^{-1}\mathbf{G}_u^{\mathrm{T}}(\mathbf{x})\left\{(\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} - \mathbf{f}(\mathbf{x})\right\} \\ -(\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d(\mathbf{x}))\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} = \mathbf{0}\ .\end{aligned} \tag{1.101}$$

Dazu multipliziere man (1.101) mit der regulären Matrix

$$\mathbf{T}(\mathbf{x}) = \begin{bmatrix} \mathbf{G}_u^{\mathrm{T}}(\mathbf{x}) \\ \mathbf{G}_u^{\perp}(\mathbf{x}) \end{bmatrix} \tag{1.102}$$

und zufolge der PBC matching condition (1.96) folgt $\mathbf{T}(\mathbf{x})\boldsymbol{\Psi} = \mathbf{0}$ und damit unmittelbar $\boldsymbol{\Psi} = \mathbf{0}$. □

Die Schwierigkeit dieser Reglerentwurfsmethode besteht offensichtlich darin, die PBC matching equation (1.96), welche ein *System partieller Differentialgleichungen* darstellt, zu lösen.
Dazu sei erwähnt, dass

- die Matrizen $\mathbf{J}_d(\mathbf{x}) = -\mathbf{J}_d^{\mathrm{T}}(\mathbf{x})$ und $\mathbf{S}_d(\mathbf{x}) = \mathbf{S}_d^{\mathrm{T}}(\mathbf{x}) \geq 0$ frei zu wählen sind,
- die Speicherfunktion des geschlossenen Kreises $V_d(\mathbf{x})$ abgesehen von der Bedingung (1.95) ebenfalls frei gewählt werden kann,
- und der Linksannihilator $\mathbf{G}_u^{\perp}(\mathbf{x})$ mit jeder regulären $(n-m)\times(n-m)$ Matrix $\boldsymbol{\Lambda}(\mathbf{x})$ von links multipliziert werden kann, d.h. $\tilde{\mathbf{G}}_u^{\perp}(\mathbf{x}) = \boldsymbol{\Lambda}(\mathbf{x})\mathbf{G}_u^{\perp}(\mathbf{x})$, ohne die PBC matching equation (1.96) zu ändern. Die Matrix $\boldsymbol{\Lambda}(\mathbf{x})$ stellt somit einen weiteren Entwurfsfreiheitsgrad dar.

In den letzten Jahren haben sich im Wesentlichen folgende Varianten des IDA-PBC Entwurfsverfahrens durchgesetzt:

- *Non-Parametrized IDA-PBC:* In diesem Fall wird die Struktur der Zusammenschaltung in Form der Matrizen $\mathbf{J}_d(\mathbf{x}) = -\mathbf{J}_d^{\mathrm{T}}(\mathbf{x})$ und $\mathbf{S}_d(\mathbf{x}) = \mathbf{S}_d^{\mathrm{T}}(\mathbf{x}) \geq 0$ vorgegeben. Mit bekanntem $\mathbf{G}_u^{\perp}(\mathbf{x})$ resultiert die PBC matching equation (1.96) zu einer partiellen Differentialgleichung für die Speicherfunktion $V_d(\mathbf{x})$. Aus der Familie aller Lösungen müssen dann jene extrahiert werden, die die Bedingung (1.95) erfüllen. In der Literatur, siehe beispielsweise [1.2], findet man auch Bedingungen für die Existenz einer Lösung der zugrundeliegenden partiellen Differentialgleichung (1.96).
- *Algebraic IDA-PBC:* In diesem Fall wird die Speicherfunktion $V_d(\mathbf{x})$ unter der Bedingung (1.95) festgelegt und die PBC matching equation (1.96) degeneriert zu einer algebraischen Gleichung für die Bestimmung der Matrizen $\mathbf{J}_d(\mathbf{x}) = -\mathbf{J}_d^{\mathrm{T}}(\mathbf{x})$ und $\mathbf{S}_d(\mathbf{x}) = \mathbf{S}_d^{\mathrm{T}}(\mathbf{x}) \geq 0$.
- *Parametrized IDA-PBC:* Hier wird die Speicherfunktion $V_d(\mathbf{x})$ auf eine bestimmte Klasse eingeschränkt, beispielsweise bei mechanischen Systemen, dass die gewünschte potenzielle Energie nur von den generalisierten Lagekoordinaten abhängt und die gewünschte kinetische Energie eine quadratische Form in den generalisierten Geschwindigkeiten ist. Diese spezielle Form von $V_d(\mathbf{x})$ impliziert eine neue PBC matching equation mit Einschränkungen bezüglich der Wahl von $\mathbf{J}_d(\mathbf{x}) = -\mathbf{J}_d^{\mathrm{T}}(\mathbf{x})$ und $\mathbf{S}_d(\mathbf{x}) = \mathbf{S}_d^{\mathrm{T}}(\mathbf{x}) \geq 0$.

Beispiel 1.2. Als Anwendungsbeispiel betrachte man eine *permanentmagnetisch erregte Synchronmaschine* in dq-Darstellung

$$\begin{aligned} L_d \frac{\mathrm{d}}{\mathrm{d}t} i_d &= -R_s i_d + \omega L_q i_q + u_d \\ L_q \frac{\mathrm{d}}{\mathrm{d}t} i_q &= -R_s i_q - \omega(L_d i_d + \Phi) + u_q \\ J \frac{\mathrm{d}}{\mathrm{d}t} \omega &= p((L_d - L_q) i_d i_q + \Phi i_q) - \tau_l \end{aligned} \tag{1.103}$$

mit den Statorströmen i_d und i_q sowie der Drehwinkelgeschwindigkeit des Rotors ω als Zustandsgrößen, den Statorspannungen u_d und u_q als Stellgrößen und dem Lastmoment τ_l. Im Weiteren bezeichnet J das Trägheitsmoment, R_s den Statorwicklungswiderstand, L_d und L_q die Statorinduktivitäten, p die Polpaarzahl und Φ den Fluss des Permanentmagneten im Rotor. Es sei an dieser Stelle erwähnt, dass für den Fall eines *gleichförmigen Luftspaltes* gilt $L_d = L_q = L$ und sich damit das mathematische Modell (1.103) entsprechend vereinfacht.

Wählt man nun als Zustandsgrößen $\mathbf{x}^{\mathrm{T}} = [x_1, x_2, x_3] = [L_d i_d, L_q i_q, J\omega/p]$, dann lässt sich (1.103) in Form eines Port-Hamiltonschen Systems

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\mathbf{J}(\mathbf{x}) - \mathbf{S})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \mathbf{G}_u \mathbf{u} + \mathbf{g}_d \tau_l \tag{1.104}$$

mit der Energiefunktion als Speicherfunktion

$$V(\mathbf{x}) = \frac{1}{2L_d} x_1^2 + \frac{1}{2L_q} x_2^2 + \frac{p}{2J} x_3^2 \tag{1.105}$$

und

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} 0 & 0 & x_2 \\ 0 & 0 & -(x_1 + \Phi) \\ -x_2 & x_1 + \Phi & 0 \end{bmatrix}, \qquad \mathbf{S} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.106}$$

sowie

$$\mathbf{G}_u = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{g}_d = \begin{bmatrix} 0 \\ 0 \\ -1/p \end{bmatrix} \quad \text{und} \quad \mathbf{u} = \begin{bmatrix} u_d \\ u_q \end{bmatrix} \tag{1.107}$$

schreiben.

Aufgabe 1.17. Zeigen Sie die Gültigkeit von (1.104).

Es soll nun mit Hilfe der *Non-Parametrized IDA-PBC* eine Zustandsrückführung gemäß

Satz 1.6 so entworfen werden, dass der stationäre Arbeitspunkt

$$\mathbf{x}_d^{\mathrm{T}} = [0, x_{2,d}, x_{3,d}] \quad \text{mit} \quad x_{2,d} = \frac{\bar{\tau}_l L_q}{\Phi p} \tag{1.108}$$

für ein konstantes Moment $\bar{\tau}_l$ und eine gewünschte Drehwinkelgeschwindigkeit $\omega_d = x_{3,d}p/J$ stabilisiert wird. Die Struktur des geschlossenen Kreises $\mathbf{J}_d(\mathbf{x})$ und $\mathbf{S}_d$ wird nun entsprechend einer Maschine mit gleichförmigem Luftspalt gewählt, d.h., es gilt $L_d = L_q = L$.

Aufgabe 1.18. Zeigen Sie, dass für $L_d = L_q = L$ die Matrizen $\mathbf{J}_d(\mathbf{x})$ und $\mathbf{S}_d$ des zu (1.103) zugehörigen Port-Hamiltonschen Systems folgende Struktur aufweisen

$$\mathbf{J}_d(\mathbf{x}) = \begin{bmatrix} 0 & \frac{Lp}{J}x_3 & 0 \\ -\frac{Lp}{J}x_3 & 0 & -\Phi \\ 0 & \Phi & 0 \end{bmatrix} \quad \text{und} \quad \mathbf{S}_d = \mathbf{S}\ . \tag{1.109}$$

Die PBC matching equation (1.96) lautet dann

$$(\mathbf{J}(\mathbf{x}) - \mathbf{S})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \mathbf{G}_u\boldsymbol{\beta}(\mathbf{x}) + \mathbf{g}_d\bar{\tau}_l = (\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d)\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} \tag{1.110}$$

bzw. mit dem Linksannihilator von $\mathbf{G}_u$

$$\mathbf{G}_u^{\perp} = [0, 0, 1] \tag{1.111}$$

und den Größen $V_a(\mathbf{x}) = V_d(\mathbf{x}) - V(\mathbf{x})$ sowie

$$\mathbf{J}_a(\mathbf{x}) = \mathbf{J}_d(\mathbf{x}) - \mathbf{J}(\mathbf{x}) = \begin{bmatrix} 0 & \frac{Lp}{J}x_3 & -x_2 \\ -\frac{Lp}{J}x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{bmatrix} \tag{1.112}$$

ergibt sich

$$\mathbf{G}_u^{\perp}(\mathbf{J}(\mathbf{x}) - \mathbf{S})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \mathbf{G}_u^{\perp}\mathbf{g}_d\bar{\tau}_l = \mathbf{G}_u^{\perp}(\mathbf{J}(\mathbf{x}) + \mathbf{J}_a(\mathbf{x}) - \mathbf{S})\left(\left(\frac{\partial V_a}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}}\right) \tag{1.113}$$

bzw.

$$-\mathbf{G}_u^{\perp}\mathbf{J}_a(\mathbf{x})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} + \mathbf{G}_u^{\perp}\mathbf{g}_d\bar{\tau}_l = \mathbf{G}_u^{\perp}(\mathbf{J}_d(\mathbf{x}) - \mathbf{S})\left(\frac{\partial V_a}{\partial \mathbf{x}}\right)^{\mathrm{T}}\ . \tag{1.114}$$

Die Auswertung von (1.114) resultiert in folgender partieller Differentialgleichung

$$-\frac{x_2 x_1}{L_d} + \frac{x_2 x_1}{L_q} - \frac{1}{p}\bar{\tau}_l = \Phi\frac{\partial V_a}{\partial x_2}\ , \tag{1.115}$$

deren allgemeine Lösung sich wie folgt

$$V_a(x_1, x_2, x_3) = \alpha_1\left(\frac{1}{2}x_2^2 x_1\left(\frac{L_d - L_q}{L_d L_q \Phi}\right) - \frac{x_2}{\Phi p}\bar{\tau}_l\right) + \psi(x_1, x_3) \tag{1.116}$$

mit dem positiven Parameter α_1 und einer noch zu wählenden Funktion $\psi(x_1, x_3)$ darstellen lässt. Damit besitzt die Speicherfunktion des geschlossenen Kreises $V_d = V + V_a$ folgende Struktur

$$V_d = \frac{1}{2L_d}x_1^2 + \frac{1}{2L_q}x_2^2 + \frac{p}{2J}x_3^2 + \frac{1}{2}\alpha_1 x_2^2 x_1\left(\frac{L_d - L_q}{L_d L_q \Phi}\right) - \alpha_1 \frac{x_2}{\Phi p}\bar{\tau}_l + \psi(x_1, x_3)\ . \quad (1.117)$$

Die Aufgabe besteht nun darin, die Funktion $\psi(x_1, x_3)$ so festzulegen, dass die Bedingungen (1.95) erfüllt werden. Man kann sich nun einfach überzeugen, dass der Ansatz

$$\psi(x_1, x_3) = -\frac{1}{2}\alpha_1\left(\frac{L_d - L_q}{L_d L_q \Phi}\right)x_1 x_{2,d}^2 + \frac{\alpha_2}{2}x_1^2 - \frac{p}{2J}x_3^2 + \frac{\alpha_3}{2}(x_3 - x_{3,d})^2 - \frac{1}{2L_q}x_{2,d}^2 \quad (1.118)$$

mit den positiven Entwurfsparametern α_1, α_2 und α_3 diese Bedingungen erfüllt. Dazu berechnet man für

$$V_d = \left(\frac{1}{2L_d} + \frac{\alpha_2}{2}\right)x_1^2 + \left(\frac{1}{2L_q} + \frac{\alpha_1}{2}x_1\left(\frac{L_d - L_q}{L_d L_q \Phi}\right)\right)\left(x_2^2 - x_{2,d}^2\right) - \frac{\alpha_1}{L_q}x_2 x_{2,d} + \frac{\alpha_3}{2}(x_3 - x_{3,d})^2 \quad (1.119)$$

vorerst den Gradienten und wertet diesen an der Stelle $\mathbf{x} = \mathbf{x}_d$ (siehe (1.108)) aus

$$\left(\frac{\partial}{\partial \mathbf{x}}V_d\right)^{\mathrm{T}}(\mathbf{x}_d) = \begin{bmatrix} \left(\frac{1}{L_d} + \alpha_2\right)x_{1,d} \\ \left(\frac{1}{L_q} + \alpha_1 x_{1,d}\left(\frac{L_d - L_q}{L_d L_q \Phi}\right)\right)x_{2,d} - \frac{\alpha_1}{L_q}x_{2,d} \\ 0 \end{bmatrix}\ . \quad (1.120)$$

Offensichtlich ist für $\alpha_1 = 1$ die Forderung $\left(\frac{\partial}{\partial \mathbf{x}}V_d\right)^{\mathrm{T}}(\mathbf{x}_d) = \mathbf{0}$ erfüllt. Um zu gewährleisten, dass $\mathbf{x}_d$ ein striktes lokales Minimum von V_d ist, muss im Weiteren

$$\left(\frac{\partial^2}{\partial \mathbf{x}^2}V_d\right)(\mathbf{x}_d) = \begin{bmatrix} \frac{1}{L_d} + \alpha_2 & \left(\frac{L_d - L_q}{L_d L_q \Phi}\right)x_{2,d} & 0 \\ \left(\frac{L_d - L_q}{L_d L_q \Phi}\right)x_{2,d} & \frac{1}{L_q} & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \quad (1.121)$$

positiv definit sein, was durch geeignete Wahl der Parameter $\alpha_2 > 0$ und $\alpha_3 > 0$ mit

$$\frac{1}{L_d} + \alpha_2 > 0 \quad \text{und} \quad \left(\frac{1}{L_d} + \alpha_2\right)\frac{1}{L_q} - \left(\frac{L_d - L_q}{L_d L_q \Phi}\right)^2 x_{2,d}^2 > 0 \quad (1.122)$$

sichergestellt wird. Die Zustandsrückführung errechnet sich dann gemäß (1.97) in der Form

$$\boldsymbol{\beta}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\left\{(\mathbf{J}_d(\mathbf{x}) - \mathbf{S}_d)\left(\frac{\partial V_d}{\partial \mathbf{x}}\right)^{\mathrm{T}} - (\mathbf{J}(\mathbf{x}) - \mathbf{S})\left(\frac{\partial V}{\partial \mathbf{x}}\right)^{\mathrm{T}} - \mathbf{g}_d \Phi p x_{2,d}\right\}\ . \quad (1.123)$$

Aufgabe 1.19. Bestimmen Sie die expliziten Ausdrücke des Zustandsregelgesetzes (1.123).

1.8 Literatur

[1.1] A. Kugi, *Skriptum zur VO Regelungssysteme 2 (SS 2019)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2019. [Online]. Available: https://www.acin.tuwien.ac.at/master/regelungssysteme-2/.

[1.2] P. Tabuada and G. Pappas, "From Nonlinear to Hamiltonian via Feedback," *IEEE Transactions on Automatic Control*, vol. 48, no. 8, pp. 1439–1442, 2003.

[1.3] O. Föllinger, *Nichtlineare Regelung I + II*. München: Oldenbourg, 1993.

[1.4] H. K. Khalil, *Nonlinear Systems (3rd Edition)*. New Jersey: Prentice Hall, 2002.

[1.5] A. Kugi, *Non-linear Control Based on Physical Models* (Lecture Notes in Control and Information Sciences 260). London: Springer, 2001.

[1.6] A. Kugi and K. Schlacher, "Analyse und Synthese nichtlinearer dissipativer Systeme: Ein Überblick (Teil 1)," *at – Automatisierungstechnik*, vol. 50, pp. 63–69, 2002.

[1.7] A. Kugi and K. Schlacher, "Analyse und Synthese nichtlinearer dissipativer Systeme: Ein Überblick (Teil 2)," *at – Automatisierungstechnik*, vol. 50, pp. 103–111, 2002.

[1.8] R. Lozano, B. Brogliato, O. Egeland, and B. Maschke, *Dissipative Systems Analysis and Control*. London: Springer, 2000.

[1.9] R. Ortega and E. García-Canseco, "Interconnection and Damping Assignment Pasivity-Based Control: A Survey," *European Journal of Control*, vol. 10, pp. 432–450, 2004.

[1.10] R. Ortega, A. van der Schaft, F. Castaños, and A. Astolfi, "Control by Interconnection and Standard Passivity-Based Control of Port-Hamiltonian Systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 11, pp. 2527–2542, 2008.

[1.11] V. Petrović, R. Ortega, and A. Stanković, "Interconnection and Damping Assignment Approach to Control of PM Synchronous Motors," *IEEE Transactions on Control Systems Technology*, vol. 9, no. 6, pp. 811–820, 2001.

[1.12] R. Sepulchre, M. JankovicHEREHEREHERE, and P. KokotovicHEREHERE-HERE, *Constructive Nonlinear Control*. London: Springer, 1997.

[1.13] E. Slotine and W. Li, *Applied Nonlinear Control*. New Jersey: Prentice Hall, 1991.

[1.14] M. Vidyasagar, *Nonlinear Systems Analysis*. New Jersey: Prentice Hall, 1993.

2 Iterative learning control

A large number of physical or industrial processes operate in an repetitive fashion, whereby the same task is performed over and over again under identical or at least very similar conditions. *Iterative learning control* (ILC) is based on the notion that the performance of such repetitive processes can be improved by learning from previous trials (i.e., iterations). Typical application examples include various robotic pick-and-place tasks, switched operation of electronic or optical systems or batched manifacturing processes.

Learning on observed information from previous trials is a very general idea and holds true for many every-day activities. For example, a basketball player throwing a ball repeatedly towards the hoop from a fixed position can improve his or her success rate over time. By observing the trajectories of the ball, the player obtains information on how to subsequently modify the throwing motion such that the future outcome improves. Iterative learning strategies can thus be seen as an *adaptive open-loop control scheme* that refines its input signals during operation through repetition and learning. This way, iterative learning strategies achieve high control performance in the presence of large uncertainty - either due to any inevitable *model-plant mismatch* or under the effect of *external disturbances* - as long as its effect is (almost) identical in each trial. Conversely, a non-learning controller is not able to leverage this additional information and thus reproduces the same residual control error in each subsequent iteration.

While learning can be applied to a large variety of problems, ILC specifically considers *tracking problems*, i.e., a system $\mathbf{G}$ is operated in a repetitive fashion whereby the input $\mathbf{u}_j(t) \in \mathcal{U}$ during the j-th iteration yields the corresponding output $\mathbf{y}_j(t) \in \mathcal{Y}$ as

$$\mathbf{y}_j(t) = \mathbf{G}[\mathbf{x}_j(0), \mathbf{u}_j(t)]. \tag{2.1}$$

Within such a setting, ILC methods typically assume that:

(i.) every iteration ends within an identical time interval $t \in [0, t_f]$,

(ii.) every iteration starts from an (almost) identical initial state $\mathbf{x}_j(0) \approx \mathbf{x}(0)$,

(iii.) there exists a unique input $\mathbf{u}_d(t)$ that yields the desired output $\mathbf{y}_d(t)$.

The main goal is now to design an operator $\mathbf{\Psi} : \mathcal{U} \times \mathcal{Y} \to \mathcal{U}$ that produces an updated input

$$\mathbf{u}_{j+1}(t) = \mathbf{\Psi}(\mathbf{u}_j(t), \mathbf{e}_j(t)) \tag{2.2}$$

based on the previous input and the resulting tracking error $\mathbf{e}_j(t) = \mathbf{y}_d(t) - \mathbf{y}_j(t)$ such that the output sequence asymptotically converges to the desired output $\mathbf{y}_d(t)$, i.e.,

$$\lim_{j \to \infty} \mathbf{G}[\mathbf{x}(0), \mathbf{u}_j(t)] = \mathbf{y}_d(t). \tag{2.3}$$

This iterative learning process is illustrated in Fig. 2.1. In some sense, ILC can be seen as a method to increase the robustness of feedforward control against model uncertainties and repetitive disturbances by determining it online through a fixed-point iteration (2.2), whose properties depend on the chosen operator $\boldsymbol{\Psi}$. For simplicity, we will restrict ourselves to linear operators $\boldsymbol{\Psi}$, wherefore the learning law (2.2) can be written as

$$\mathbf{u}_{j+1}(t) = \mathbf{L}_u \mathbf{u}_j(t) + \mathbf{L}_e \mathbf{e}_j(t) = \mathbf{Q}(\mathbf{u}_j(t) + \mathbf{L}\mathbf{e}_j(t)) \tag{2.4}$$

with the linear operators $\mathbf{L}_u$, $\mathbf{L}_e$, $\mathbf{Q}$, and $\mathbf{L}$, respectively. Correspondingly, we will mainly restrict ourselves to linear, time-invariant systems of the form

$$\dot{\mathbf{x}}_j(t) = \mathbf{A}\mathbf{x}_j(t) + \mathbf{B}\mathbf{u}_j(t) \tag{2.5}$$
$$\mathbf{y}_j(t) = \mathbf{C}\mathbf{x}_j(t) + \mathbf{D}\mathbf{u}_j(t) \tag{2.6}$$

with the state vector $\mathbf{x}_j^{\mathrm{T}}(t) \in \mathbb{R}^n$, the input vector $\mathbf{u}_j^{\mathrm{T}}(t) \in \mathbb{R}^l$ and the output vector $\mathbf{y}_j^{\mathrm{T}}(t) \in \mathbb{R}^m$.

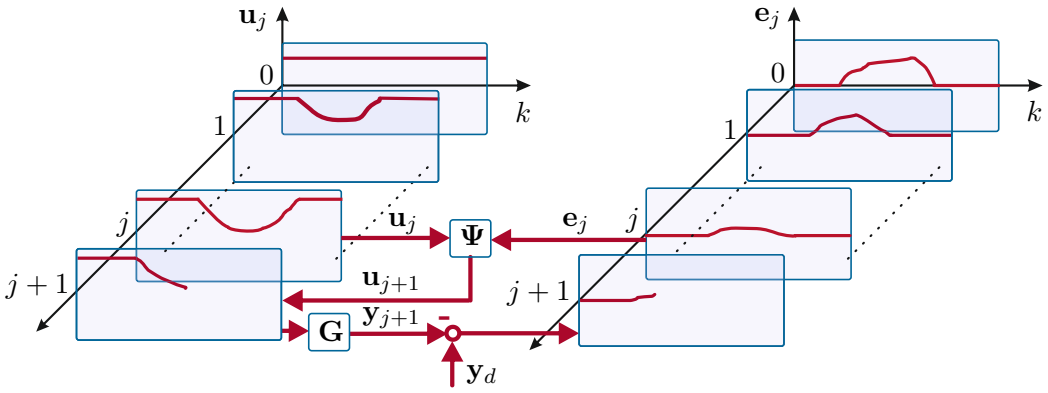


Figure 2.1: Graphical illustration of iterative learning control (ILC).

There are a number of different concepts in control theory that aim to include some element of learning. Most notably, *adaptive control* is using information from the past to increase the performance of the closed-loop system by continuously modifying model or (feedback) control parameters. All uncertainty is thus represented in a parametric form. Conversely, ILC modifies the feedforward inputs applied to the system and is thus not limited to parametric uncertainties. However, this entails that the learned input signal is specific to the desired output $\mathbf{y}_d(t)$. One property that is used to distinguish ILC [2.1–2.5] from other learning concepts [2.6] is the so-called *identical initialization condition*, i.e., that every iteration starts from the exact same initial state $\mathbf{x}_j(0) = \mathbf{x}(0)$. In a more relaxed formulation, $\mathbf{x}_j(0)$ may be a stochastic quantity, but at least is independent of previous iterations. *Repetitive control* on the other hand assumes that the initial state of the current iteration is given by the terminal state of the previous one such that the sequence of all iterations can be seen as a continuously operated system.

2.1 Fixed-point iterations

Calculating zeros of a function

$$\boldsymbol{\gamma}(\mathbf{z}) = \mathbf{0} \tag{2.7}$$

with $\boldsymbol{\gamma}(\mathbf{z}) : \mathbb{R}^N \to \mathbb{R}^N$ and $\mathbf{z}^{\mathrm{T}} = \begin{bmatrix} z_1 & \dots & z_N \end{bmatrix}$ can always be recast as a fixed-point problem

$$\boldsymbol{\psi}(\mathbf{z}) = \mathbf{z}. \tag{2.8}$$

A *fixed-point iteration* is a sequence $\mathbf{z}_0, \mathbf{z}_1, \dots$ given by

$$\mathbf{z}_{j+1} = \boldsymbol{\psi}(\mathbf{z}_j) \ , \quad j = 0, 1, 2, \dots \ . \tag{2.9}$$

that converges to a solution $\mathbf{z}_\infty$ of (2.8) depending on the chosen function $\boldsymbol{\psi}$. Note that there is no unique fixed-point formulation for a given zero-point problem, i.e., the choices

- $\boldsymbol{\psi}(\mathbf{z}) = \mathbf{z} - \boldsymbol{\gamma}(\mathbf{z})$
- $\boldsymbol{\psi}(\mathbf{z}) = \mathbf{z} + 2\boldsymbol{\gamma}(\mathbf{z})$
- $\boldsymbol{\psi}(\mathbf{z}) = \mathbf{z} - \left(\frac{\partial}{\partial \mathbf{z}}\boldsymbol{\gamma}\right)^{-1}(\mathbf{z})\boldsymbol{\gamma}(\mathbf{z})$

are equally valid. However, the choice of $\boldsymbol{\psi}$ crucially determines the convergence properties of the fixed-point iteration. Therefore, we define:

Definition 2.1 (Convergence)**.** The iteration (2.9) is

- *locally convergent (LC)* to $\mathbf{z}_\infty$ if there exists a $\delta > 0$ such that the iteration (2.9) exists and converges to $\mathbf{z}_\infty$ for all starting points $\|\mathbf{z}_0 - \mathbf{z}_\infty\| < \delta$,
- *globally convergent (GC)* if the iteration (2.9) converges to $\mathbf{z}_\infty$ for all $\mathbf{z}_0$.

Definition 2.2 (Stability)**.** A fixed-point $\mathbf{z}_\infty$ is

- *stable (S)* (in a Lyapunov sense), if for every $\varepsilon > 0$ there exists a $\delta > 0$ such that if $\|\mathbf{z}_0 - \mathbf{z}_\infty\| < \delta$, the sequence $\{\mathbf{z}_j\}$ exists and $\|\mathbf{z}_j - \mathbf{z}_\infty\| < \varepsilon$ for all $j \geq 1$,
- *attractive (A)*, if there exists a $\delta > 0$ such that $\|\mathbf{z}_0 - \mathbf{z}_\infty\| < \delta$ implies that $\{\mathbf{z}_j\}$ exists and $\lim_{j\to\infty} \mathbf{z}_j = \mathbf{z}_\infty$,
- *globally attractive (GA)* if $\delta = \infty$ above,
- *asymptotically stable (AS)*, if stable and attractive and *globally asymptotically stable (GAS)*, if stable and globally attractive.

Attractivity and convergence are equivalent concepts, whereby the following relations [2.7] hold true

$$GAS \implies GA \iff GC \implies A \impliedby AS \ . \tag{2.10}$$

In the following we will consider the special case of linear fixed-point iterations

$$\mathbf{z}_{j+1} = \mathbf{\Psi}\mathbf{z}_j \ , \quad j = 0, 1, 2, \ldots . \tag{2.11}$$

Definition 2.3 (Stability and asymptotic stability of linear fixed-point iterations)**.** A linear iteration (2.11) is stable, if

$$\sup_{j \geq 1} \|\mathbf{\Psi}^j\| < \infty \ , \tag{2.12}$$

and asymptotically stable if

$$\lim_{j \to \infty} \|\mathbf{\Psi}^j\| = 0 \ . \tag{2.13}$$

Definition 2.4 (Spectral radius)**.** The spectrum of a matrix $\mathbf{\Gamma}$ denotes the set of all its eigenvalues, i.e.,

$$\sigma(\mathbf{\Gamma}) = \{\lambda \in \mathbb{C} \mid \det(\lambda \mathbf{I} - \mathbf{\Gamma}) = 0\} \tag{2.14}$$

and

$$\rho(\mathbf{\Gamma}) = \max_{\lambda \in \sigma(\mathbf{\Gamma})} |\lambda| \tag{2.15}$$

denotes the spectral radius of $\mathbf{\Gamma}$.

Satz 2.1. *A linear iteration (2.11) is stable iff $\rho(\mathbf{\Psi}) \leq 1$ and all eigenvalues at 1 are distinct eigenvalues, i.e., their algebraic multiplicity equals 1. The iteration is further asymptotically stable iff $\rho(\mathbf{\Psi}) < 1$.*

Definition 2.5 (BIBO stability)**.** A linear iteration

$$\mathbf{z}_{j+1} = \mathbf{\Psi}\mathbf{z}_j + \mathbf{\Lambda}\mathbf{v}_j \ , \quad \mathbf{z}_0 = \mathbf{0} \tag{2.16}$$

is called bounded-input-bounded-output (BIBO) stable if every bounded input sequence $\{\mathbf{v}_j\}$ results in a bounded output sequence $\{\mathbf{z}_j\}$.

Satz 2.2. *A linear iteration $\mathbf{z}_{j+1} = \mathbf{\Psi}\mathbf{z}_j + \mathbf{\Lambda}\mathbf{v}_j$, $\mathbf{z}_0 = \mathbf{0}$ is BIBO stable iff $\rho(\mathbf{\Psi}) < 1$.*

Lemma 2.1. *For a bounded sequence $\{\mathbf{z}_j\}$ and $\varepsilon > 0 \in \mathbb{R}$ with*

$$\|\mathbf{z}_{j+1}\| \leq \rho \|\mathbf{z}_j\| + \varepsilon \ , \quad 0 \leq \rho < 1 \tag{2.17}$$

it follows that

$$\limsup_{j\to\infty} \|\mathbf{z}_j\| \leq \frac{1}{1-\rho}\varepsilon \ . \tag{2.18}$$

Proof. Iterative application of the iteration (2.17) yields

$$\begin{aligned}
\|\mathbf{z}_1\| &\leq \rho\|\mathbf{z}_0\| + \varepsilon \\
\|\mathbf{z}_2\| &\leq \rho^2\|\mathbf{z}_0\| + (1+\rho)\varepsilon \\
&\vdots \\
\|\mathbf{z}_j\| &\leq \rho^j\|\mathbf{z}_0\| + \sum_{j=0}^{j-1}\rho^j\varepsilon = \rho^j\|\mathbf{z}_0\| + \frac{1-\rho^j}{1-\rho}\varepsilon.
\end{aligned} \tag{2.19}$$

Since $0 \leq \rho < 1$, one directly obtains (2.18) for $j \to \infty$. □

While asymptotic stability ensures that an iteration (2.11) eventually converges, this is not necessarily happening in a monotonic way. Since the notion of iterative learning somewhat suggests that one is successively progressing towards a desired solution, i.e., that the control performance improves with every iteration, monotonicity is an important property of ILC algorithms.

Definition 2.6 (Largest singular value)**.** The largest singular value of a matrix $\mathbf{\Psi}$ is given by

$$\bar{\sigma}(\mathbf{\Psi}) = \sqrt{\rho(\mathbf{\Psi}^{\mathrm{T}}\mathbf{\Psi})} \ . \tag{2.20}$$

Our main interest in the largest singular value of a matrix stems from the fact that it is the induced norm of a matrix mapping two spaces equiped with the Euclidean norm $\|z\| = \sqrt{\mathbf{z}^{\mathrm{T}}\mathbf{z}}$, i.e.,

$$\|\mathbf{\Psi}\mathbf{z}_j\| \leq \|\mathbf{\Psi}\|_{i,2}\|\mathbf{z}_j\| = \bar{\sigma}(\mathbf{\Psi})\|\mathbf{z}_j\| \ . \tag{2.21}$$

The largest singular value can thus be seen as an upper bound of the gain or amplification of the mapping given by the matrix $\mathbf{\Psi}$.

Satz 2.3 (Monotone convergence of linear iterations)**.** *A linear iteration $\mathbf{z}_{j+1} = \mathbf{\Psi}\mathbf{z}_j$ converges monotonically towards $\mathbf{0}$ in the l_2-norm, i.e., it holds true that*

$$\|\mathbf{z}_{j+1}\| \leq \beta\|\mathbf{z}_j\| \quad \textit{and thus} \quad \|\mathbf{z}_{j+1}\| \leq \beta^j\|\mathbf{z}_0\| \tag{2.22}$$

for $0 \leq \beta < 1$, if

$$\bar{\sigma}(\mathbf{\Psi}) < 1 \ . \tag{2.23}$$

Note that since $\rho(\mathbf{\Psi}) \leq \bar{\sigma}(\mathbf{\Psi})$, monotonic convergence unsurprisingly implies convergence.

2.2 Signals, systems, and the frequency domain

Integral transforms are essential tools to analyze temporal signals and their interaction with dynamical systems. Here, we consider a *signal* as a real-valued (measureable) function that maps the real numbers $\mathbb{R}$ to $\mathbb{R}^n$. The set of all signals

$$\mathcal{S} = \{\mathbf{f} : \mathbb{R} \to \mathbb{R}^n\} \tag{2.24}$$

can intuitively be conceived as a set that contains all signals that can possibly occur in an engineering system - and many more. Since the sum of two signals $\mathbf{f}$ and $\mathbf{g}$ and the product of a signal with a scalar are again contained in this set, signals form a natural *linear vector space*.

Definition 2.7. A *linear vector space* over a field $\mathbb{K}$ is a set $\mathcal{S}$ with a binary operation $+ : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ (addition) and a binary operation $\cdot : \mathbb{K} \times \mathcal{S} \to \mathcal{S}$ (multiplication) that fulfils

1. The set $\mathcal{S}$ and the operation $+$ are a commutative group.
2. Multiplication with scalars $a, b \in \mathbb{K}$ for $\mathbf{f}, \mathbf{g} \in \mathcal{S}$ satisfies
   - $a(\mathbf{f} + \mathbf{g}) = a\mathbf{f} + a\mathbf{g}$
   - $(a + b)\mathbf{f} = a\mathbf{f} + b\mathbf{f}$
   - $(ab)\mathbf{f} = a(b\mathbf{f})$
   - $0\mathbf{f} = \mathbf{0}$ and $1\mathbf{f} = \mathbf{f}$.

To measure the size of a signal, one typically equips vector spaces with a suitable norm.

Definition 2.8. A *normed linear vector space* $\mathcal{S}$ is a linear vector space equipped with real-valued function $\|\cdot\|_p : \mathcal{S} \to \mathbb{R}$ that adheres to the following properties:

1. $\|\mathbf{f}\|_p \geq 0$
2. $\|\mathbf{f}\|_p = 0 \iff \mathbf{f} = 0$
3. $\|a\mathbf{f}\|_p = |a|\|\mathbf{f}\|_p$
4. $\|\mathbf{f} + \mathbf{g}\|_p \leq \|\mathbf{f}\|_p + \|\mathbf{g}\|_p$

Note that a vector space can be equipped with different norms. The resulting normed vector spaces are different and may contain different elements.

We will now introduce typical signal spaces that we will require to analyze ILC algorithms. Since we consider signals with values in some $\mathbb{R}^n$, this implicitly requires a notion of size in this underlying vector space, for which we simply use the Euclidean norm. One of the most intuitive candidates for an ILC setting is the finite-horizon 2-norm defined by

$$\|\mathbf{f}\|_{2,[0,t_f]} = \left\{ \int_0^{t_f} \|\mathbf{f}(t)\| \, \mathrm{d}t \right\}^{\frac{1}{2}} = \left\{ \int_0^{t_f} \mathbf{f}(t)^{\mathrm{T}} \mathbf{f}(t) \, \mathrm{d}t \right\}^{\frac{1}{2}}. \tag{2.25}$$

The set of signals for which this norm is finite is known as the finite-horizon Lebesque 2-space

$$\mathcal{L}_2([0,t_f];\mathbb{R}^n) = \Big\{\mathbf{f} \in \mathcal{S} : \|\mathbf{f}\|_{2,[0,t_f]} < \infty\Big\}. \tag{2.26}$$

For simplicity, we will simply write $\mathcal{L}_2([0,t_f])$ in cases where the dimension of the vector-valued signals is not important. Any signal that is continuous on $[0,t_f]$ is bounded and thus contained in $\mathcal{L}_2([0,t_f])$ but signals like $\frac{1}{|2t-t_f|}$ are not. In order to address stability issues or to apply frequency-domain methods, one must consider signals over infinite time intervals. Extending the considered horizon to infinity on both sides yields the usual infinite-horizon Lebesque 2-space

$$\mathcal{L}_2(\mathbb{R}) = \{\mathbf{f} \in \mathcal{S} : \|\mathbf{f}\|_2 < \infty\} \tag{2.27}$$

with the corresponding norm

$$\|\mathbf{f}\|_2 = \left[\int_{-\infty}^{\infty} \|\mathbf{f}(t)\|\,\mathrm{d}t\right]^{\frac{1}{2}}. \tag{2.28}$$

The spaces $\mathcal{L}_2([0,\infty))$ and $\mathcal{L}_2((-\infty,0])$ can be defined analogously.

For conveniance, we will restrict ourselves to *Hilbert spaces*, i.e., *complete* spaces whose norm is generated by an *inner product* $\|\mathbf{f}\|_2 = \sqrt{\langle \mathbf{f},\mathbf{f}\rangle}$.

Definition 2.9 (Inner product)**.** A mapping $\mathcal{S}\times\mathcal{S}\to\mathbb{K}$ that assigns a scalar to each two elements of a vector space is called an *inner product* if the conditions

1. $\langle \mathbf{f}+\mathbf{g},\mathbf{h}\rangle = \langle \mathbf{f},\mathbf{h}\rangle + \langle \mathbf{g},\mathbf{h}\rangle$
2. $\langle \mathbf{f},\mathbf{g}\rangle = \langle \mathbf{g},\mathbf{f}\rangle^*$
3. $\langle a\mathbf{f},\mathbf{g}\rangle = a\langle \mathbf{f},\mathbf{g}\rangle$
4. $\langle \mathbf{f},\mathbf{f}\rangle > 0$ and $\langle \mathbf{f},\mathbf{f}\rangle = 0 \iff \mathbf{f} = 0$

hold true for $\mathbf{f},\mathbf{g},\mathbf{h}\in\mathcal{S}$ and $a\in\mathbb{K}$.

The space $\mathcal{L}_2(\mathbb{R})$ is a Hilbert space with inner product defined by

$$\langle \mathbf{f},\mathbf{g}\rangle = \int_{-\infty}^{\infty} \mathbf{g}(t)^{\mathrm{T}}\mathbf{f}(t)\,\mathrm{d}t. \tag{2.29}$$

Two signals $\mathbf{f}$ and $\mathbf{g}$ are called orthogonal if $\langle \mathbf{f},\mathbf{g}\rangle = 0$ analogous to orthogonality in $\mathbb{R}^n$. The spaces $\mathcal{L}_2([0,\infty))$, $\mathcal{L}_2((-\infty,0])$, and $\mathcal{L}_2([0,t_f])$ are all Hilbert spaces in their own right, with the inner product integral taken over the appropriate time interval, e.g., for $\mathcal{L}_2([0,t_f])$ we have

$$\langle \mathbf{f},\mathbf{g}\rangle_{[0,t_f]} = \int_0^{t_f} \mathbf{g}(t)^{\mathrm{T}}\mathbf{f}(t)\,\mathrm{d}t. \tag{2.30}$$

The Fourier transform

The Fourier transform is a unique mapping of certain (real-valued) functions of time to complex-valued functions of a single real variable ω using

$$\hat{\mathbf{f}}(j\omega) = \mathcal{F}\{\mathbf{f}\} = \int_{-\infty}^{\infty} \mathbf{f}(t)\, \mathrm{e}^{-j\omega t}\, \mathrm{d}t. \tag{2.31}$$

Note that the Fourier transform in (2.31) is performed for each component individually and a vector-valued signal thus simply yields a vector of all transformed components. To ensure that such a function $\hat{\mathbf{f}}(j\omega)$ exists and is reasonable well behaved, classical Fourier analysis assumes that each component of $\mathbf{f}(t)$ is absolutely integrable over the real line

$$\int_{-\infty}^{\infty} |f_i(t)|\, \mathrm{d}t < \infty, \tag{2.32}$$

i.e., each component is a $\mathcal{L}_1(-\infty, \infty)$ function. With some mathematical effort, the Fourier transform can be extended to $\mathcal{L}_2(\mathbb{R})$ (and functions beyond that such as the Dirac delta function). To motivate this, one can introduce the inner product of the transformed signals

$$\langle \hat{\mathbf{f}}, \hat{\mathbf{g}} \rangle = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{\mathbf{g}}(j\omega)^H\, \hat{\mathbf{f}}(j\omega)\, \mathrm{d}\omega, \tag{2.33}$$

which again introduces an Lebesque 2-space $\mathcal{L}_2(\mathbb{R})$ in the frequency domain. Here, $(\cdot)^H$ denotes the conjugate transpose or Hermitian transpose.

Satz 2.4 (Parseval's theorem). *For $\mathbf{f}, \mathbf{g} \in \mathcal{L}_2(\mathbb{R})$ and $\hat{\mathbf{f}}(j\omega) = \mathcal{F}\{\mathbf{f}\}, \hat{\mathbf{g}}(j\omega) = \mathcal{F}\{\mathbf{g}\} \in \mathcal{L}_2(\mathbb{R})$ it follows that*

$$\langle \mathbf{f}, \mathbf{g} \rangle = \langle \hat{\mathbf{f}}, \hat{\mathbf{g}} \rangle \tag{2.34}$$

and thus

$$\|\mathbf{f}\|_2 = \|\hat{\mathbf{f}}\|_2. \tag{2.35}$$

The Fourier transform is thus a mapping between two Lebesque 2-spaces that preserves the inner product and the norm, i.e., a Hilbert space *isomorphism*. Finally, the inverse transform is given by

$$\mathbf{f}(t) = \mathcal{F}^{-1}\{\hat{\mathbf{f}}\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{\mathbf{f}}(j\omega)\, \mathrm{e}^{j\omega t}\, \mathrm{d}\omega. \tag{2.36}$$

The bilateral Laplace transform

For reasons that are quite obvious to a control engineer, one is interested to extend the Fourier transform from the imaginary axis $j\omega$ to the complex plane $s = \sigma + j\omega$, which directly yields the bilateral Laplace transform

$$\hat{\mathbf{f}}(s) = \mathcal{B}\{\mathbf{f}\} = \int_{-\infty}^{\infty} \mathbf{f}(t)\, \mathrm{e}^{-st}\, \mathrm{d}t \tag{2.37}$$

that is equivalent to the Fourier transform for $\sigma = 0$. In general, the Laplace transform will not converge for arbitrary values of s. From

$$\hat{\mathbf{f}}(\sigma + j\omega) = \int_{-\infty}^{\infty} \mathbf{f}(t)\mathrm{e}^{-\sigma t}\, \mathrm{e}^{-j\omega t}\, \mathrm{d}t \tag{2.38}$$

we see that the region of convergence (ROC) of the Laplace transform is directly linked to the convergence of $\mathbf{f}(t)\mathrm{e}^{-\sigma t}$.

Satz 2.5. *The ROC of a signal has the following properties:*

1. *The ROC consists of strips parallel to the $j\omega$-axis in the s-plane.*
2. *If $\mathbf{f}(t)$ is of finite duration and is absolutely integrable, then the ROC is the entre s-plane.*
3. *If $\mathbf{f}(t)$ is right-sided (i.e., $\mathbf{f}(t) = 0$ for $t < \bar{t}$) and if the line $\mathrm{Re}\{s\} = \sigma_0$ is in the ROC, then all values of s with $\mathrm{Re}\{s\} > \sigma_0$ are also included in the ROC.*
4. *If $\mathbf{f}(t)$ is left-sided (i.e., $\mathbf{f}(t) = 0$ for $t > \bar{t}$) and if the line $\mathrm{Re}\{s\} = \sigma_0$ is in the ROC, then all values of s with $\mathrm{Re}\{s\} < \sigma_0$ are also included in the ROC.*
5. *If the Laplace transform $\hat{\mathbf{f}}(s)$ is rational, then its ROC is bounded by poles or extends to infinity. No poles are contained in the ROC.*

In particular, a signal that is contained in $\mathcal{L}_2([0,\infty))$ (and assuming that it is absolutely integrable for simplicity) is thus bounded and analytic (i.e., holomorphic) for $\mathrm{Re}\{s\} > 0$, which is the definition of the Hardy 2-space $\mathcal{H}_2$. Conversely, a signal that is contained in $\mathcal{L}_2((-\infty,0])$ is bounded and analytic for $\mathrm{Re}\{s\} < 0$, which is the definition of the complementary Hardy 2-space $\mathcal{H}_2^\perp$. Since for every $\hat{\mathbf{f}} \in \mathcal{H}_2$ it follows that $\lim_{\sigma\to+0} \hat{\mathbf{f}} \in \mathcal{L}_2(\mathbb{R})$ and analogous $\hat{\mathbf{f}} \in \mathcal{H}_2^\perp$ implies that $\lim_{\sigma\to-0} \hat{\mathbf{f}} \in \mathcal{L}_2(\mathbb{R})$, we regard $\mathcal{H}_2$ and $\mathcal{H}_2^\perp$ closed subspaces of the frequency-domain $\mathcal{L}_2(\mathbb{R})$. Since any time signal can be decomposed as

$$\mathbf{f}(t) = \mathbf{f}_1(t) + \mathbf{f}_2(t) \quad \text{with} \quad \mathbf{f}_1 \in \mathcal{L}_2([0,\infty)), \mathbf{f}_2 \in \mathcal{L}_2((-\infty,0]) \tag{2.39}$$

and $\langle \mathbf{f}_1, \mathbf{f}_2 \rangle = 0$, one can see from $\mathcal{L}_2(\mathbb{R}) = \mathcal{H}_2 \cup \mathcal{H}_2^\perp$ and Parseval's theorem that

$$\langle \hat{\mathbf{f}}_1, \hat{\mathbf{f}}_2 \rangle = 0 \quad \text{and} \quad \mathcal{H}_2 \cap \mathcal{H}_2^\perp = 0, \tag{2.40}$$

which justifies the already used nomenclature $\mathcal{H}_2$ and $\mathcal{H}_2^\perp$, respectively. A graphical illustrations of these connections is given in Fig. 2.2. Finally, the inverse transformation to 2.37 is given by the contour integral

$$\mathbf{f}(t) = \mathcal{B}^{-1}\left\{\hat{\mathbf{f}}\right\} = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} \hat{\mathbf{f}}(s)\, \mathrm{e}^{st}\, \mathrm{d}s \tag{2.41}$$

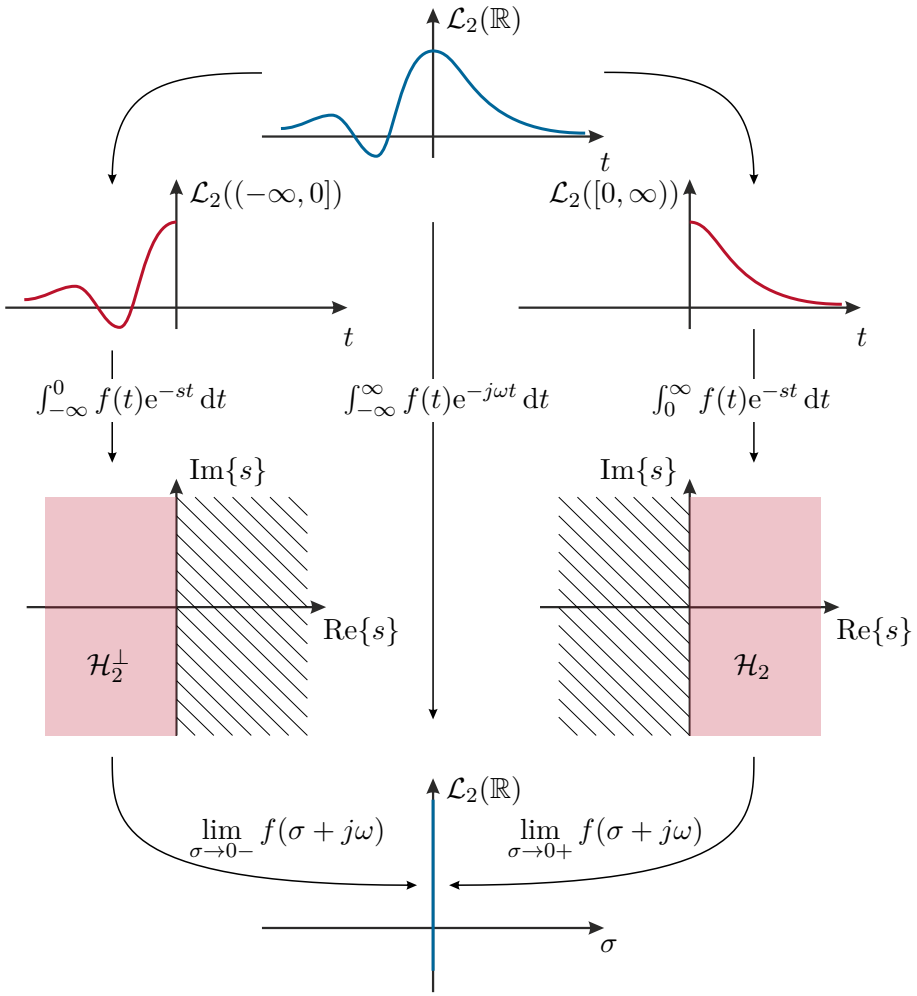


Figure 2.2: On $\mathcal{L}_2((-\infty, 0])$, $\mathcal{L}_2([0, \infty))$, $\mathcal{H}_2$, and $\mathcal{H}_2^{\perp}$.

where the line of integration $\mathrm{Re}\{s\} = \sigma$ is within the corresponding ROC of $\hat{\mathbf{f}}(s)$.

By definition, the bilateral Laplace transform shares most of its properties with the unilateral Laplace transform. For convenience, the most common properties are summarized in Table 2.1. Selected pairs of (scalar) time-domain signals $f(t)$ and their corresponding frequency-domain functions $\hat{f}(s)$ are listed in Table 2.2. In the following, we will not make a notational distinction between time-domain signals and their frequency-domain representations and drop the hat notation unless it is necessary to avoid confusion. In general, the argument or the context sufficiently determines whether one is dealing with a time-domain or frequency-domain signal, which are anyhow isomorphic under the used integral transformations.

Property	**Expression**	**ROC**
Linearity	$\mathcal{B}\{af(t)+bg(t)\} = a\hat{f}(s)+b\hat{g}(s)$	Intersection of ROCs
Time Shifting	$\mathcal{B}\{f(t-t_0)\} = e^{-st_0}\hat{f}(s)$	Same as $\hat{f}(s)$
Frequency Shifting	$\mathcal{B}\{e^{at}f(t)\} = F(s-a)$	$\mathrm{Re}(s)$ shifted by a
Time Reversal	$\mathcal{B}\{f(-t)\} = F(-s)$	Reflect ROC about $\mathrm{Re}(s)$
Differentiation (in t)	$\mathcal{B}\{f'(t)\} = s\hat{f}(s)$	Same as $\hat{f}(s)$
Differentiation (in s)	$\mathcal{B}^{-1}\left\{-\frac{d}{ds}\hat{f}(s)\right\} = tf(t)$	Same as $\hat{f}(s)$
Integration	$\mathcal{B}\left\{\int_{-\infty}^{t} f(\tau)\,d\tau\right\} = \frac{1}{s}\hat{f}(s)$	ROC extends to include $s=0$
Convolution	$\mathcal{B}\{f(t)*g(t)\} = \hat{f}(s)\hat{g}(s)$	Intersection of ROCs

Table 2.1: Selected properties of the bilateral Laplace transform.

Signal $f(t)$	**Bilateral Laplace Transform** $\hat{f}(s)$	**ROC**
$\delta(t)$	1	all s
$h(t)$	$\frac{1}{s}$	$\mathrm{Re}(s) > 0$
$-h(-t)$	$\frac{1}{s}$	$\mathrm{Re}(s) < 0$
$\frac{t^{n+1}}{(n-1)!}h(t)$	$\frac{1}{s^n}$	$\mathrm{Re}(s) > 0$
$-\frac{t^{n+1}}{(n-1)!}h(-t)$	$\frac{1}{s^n}$	$\mathrm{Re}(s) < 0$
$e^{-at}h(t)$	$\frac{1}{s+a}$	$\mathrm{Re}(s) > -a$
$-e^{-at}h(-t)$	$\frac{1}{s+a}$	$\mathrm{Re}(s) < -a$
$\frac{t^{n+1}}{(n-1)!}e^{-at}h(t)$	$\frac{1}{(s+a)^n}$	$\mathrm{Re}(s) > -a$
$-\frac{t^{n+1}}{(n-1)!}e^{-at}h(-t)$	$\frac{1}{(s+a)^n}$	$\mathrm{Re}(s) < -a$
$e^{-at}\sin(bt)h(t)$	$\frac{b}{(s+a)^2+b^2}$	$\mathrm{Re}(s) > -a$
$e^{-at}\cos(bt)h(t)$	$\frac{s+a}{(s+a)^2+b^2}$	$\mathrm{Re}(s) > -a$
$e^{-a\lvert t\rvert}$	$\frac{2a}{a^2-s^2}$	$-a < \mathrm{Re}(s) < -a$
$e^{-a^2t^2}$	$\frac{\sqrt{\pi}}{a}e^{\frac{s^2}{4a^2}}$	all s

Table 2.2: Selected transform pairs of the bilateral Laplace transform and their associated ROC with the Dirac delta function $\delta(t)$ and the Heaviside step function $h(t)$.

Input-output representation of linear systems

The input-output behavior of a dynamical system can be seen as a mapping from one signal space, the input space $\mathcal{U}$, to another space, the output space $\mathcal{Y}$:

$$\mathbf{G} : \mathbf{u} \in \mathcal{U} \mapsto \mathbf{y} = \mathbf{G}\mathbf{u} \in \mathcal{Y} \tag{2.42}$$

For the case $\mathcal{U} = \mathcal{L}_2(\mathbb{R}; \mathbb{R}^l)$ and $\mathcal{Y} = \mathcal{L}_2(\mathbb{R}; \mathbb{R}^m)$, the input-output behavior of any linear system can be represented by the integral operator

$$\mathbf{y}(t) = \int_{-\infty}^{\infty} \mathbf{G}(t, \tau)\mathbf{u}(t)\, \mathrm{d}\tau \tag{2.43}$$

with the kernel function $\mathbf{G}(t, \tau) \in \mathbb{R}^{m \times l}$ that is usually referred to as the systems impulse response matrix. The system is said to be causal iff $\mathbf{G}(t, \tau) = 0$ for all $\tau > t$ and time-invariant if $\mathbf{G}(t, \tau) = \mathbf{G}(t - \tau, 0)$ for all t, τ. Linear time-invariant (LTI) systems are thus represented by a convolution-type integral

$$\mathbf{y}(t) = \int_{-\infty}^{\infty} \mathbf{G}(t - \tau)\mathbf{u}(t)\, \mathrm{d}\tau \tag{2.44}$$

where $\mathbf{G}(t-\tau, 0)$ is denoted as $\mathbf{G}(t-\tau)$ for simplicity. A system is called *causal* if $\mathbf{G}(t) = 0$ for $t < 0$ and *anticausal* if $\mathbf{G}(t) = 0$ for $t > 0$. If a system is neither causal nor anticausal, it is called *non-causal* or *acausal*. By applying the Laplace transform (2.37) to (2.44), one obtains

$$\mathbf{y}(s) = \mathbf{G}(s)\mathbf{u}(s) \tag{2.45}$$

with the transfer function matrix

$$\mathbf{G}(s) = \int_{-\infty}^{\infty} \mathbf{G}(t)\, \mathrm{e}^{-st}\, \mathrm{d}t. \tag{2.46}$$

The following properties of a dynamic system are directly related to the ROC of the transfer function matrix:

Satz 2.6 (Causality and stability)**.** *For an LTI system $\mathbf{G}(s)$ with corresponding ROC it holds true that:*

1. *If it is causal then its ROC is a right-half plane. If $\mathbf{G}(s)$ is rational, then causality is equivalent to the ROC being the right-half plane to the right of the rightmost pole.*
2. *If it is anticausal then its ROC is a left-half plane. If $\mathbf{G}(s)$ is rational, then anticausality is equivalent to the ROC being the left-half plane to the left of the leftmost pole.*
3. *It is stable iff the ROC includes $\mathrm{Re}\{s\} = 0$.*

Note that if one assumes a system to be rational and causal, then this is equivalent to the usual stability criterion that all poles have negative real part, i.e., they lie in the left-half of the s-plane. Unlike feedback control, ILC methods do not need to be causal

in time. In fact, strictly causal ILC algorithms are known to be inferior to noncausal algorithms since they lack the ability to *plan ahead*. This ability is closely linked to so-called *adjoint systems* that regularly appear in (optimation-based) feedforward control methods.

Aufgabe 2.1. Show that the system

$$G(s) = \frac{\exp(s)}{s+1} \quad \text{with } \mathrm{Re}\{s\} > -1 \tag{2.47}$$

is not causal although the ROC is right to the rightmost pole.

Aufgabe 2.2. Specify all possible ROCs for the system

$$G(s) = \frac{s-1}{(s+1)(s-2)} \tag{2.48}$$

and determine the corresponding impulse response functions.

The $\mathcal{L}_\infty$ norm

That a linear time-invariant system indeed maps some $\mathcal{L}_2(\mathbb{R};\mathbb{R}^l)$ to $\mathcal{L}_2(\mathbb{R};\mathbb{R}^m)$ is only true if $\mathbf{Gu} \in \mathcal{L}_2(\mathbb{R};\mathbb{R}^m)$ for any $\mathbf{u} \in \mathcal{L}_2(\mathbb{R};\mathbb{R}^l)$. Due to Parseval's theorem 2.4, one can evaluate the norm of a signal either in the time or frequency domain. The 2-norm of the output is thus

$$\begin{aligned}\|\mathbf{Gu}\|_2^2 = \langle \mathbf{Gu}, \mathbf{Gu}\rangle &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \|\mathbf{G}(j\omega)\mathbf{u}(j\omega)\|\,\mathrm{d}\omega \\ &\leq \frac{1}{2\pi}\int_{-\infty}^{\infty} \bar{\sigma}(\mathbf{G}(j\omega))^2\|\mathbf{u}(j\omega)\|\,\mathrm{d}\omega \\ &\leq \sup_\omega \bar{\sigma}(\mathbf{G}(j\omega))^2 \frac{1}{2\pi}\int_{-\infty}^{\infty} \|\mathbf{u}(j\omega)\|\,\mathrm{d}\omega. \end{aligned} \tag{2.49}$$

Using the definition of the supremum norm or $\mathcal{L}_\infty$-norm

$$\|\mathbf{G}\|_\infty = \sup_\omega \bar{\sigma}(\mathbf{G}(j\omega)) \tag{2.50}$$

yields

$$\|\mathbf{Gu}\|_2 \leq \|\mathbf{G}\|_\infty \|\mathbf{u}\|_2, \tag{2.51}$$

i.e., the $\mathcal{L}_\infty$-norm is the induced operator norm of a mapping between two $\mathcal{L}_2$-spaces. Consequently, a sufficient condition for $\mathbf{Gu} \in \mathcal{L}_2(\mathbb{R};\mathbb{R}^m)$ is $\sup_\omega \bar{\sigma}(\mathbf{G}(j\omega)) < \infty$. Note that if $\mathbf{G}(s)$ is a rational transfer matrix, this is true if and only if $\mathbf{G}(s)$ has no poles on the imaginary axis.

Systems in state-space representation

For a given state-space representation of a LTI system with measurement noise

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)\ , \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{2.52a}$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) + \mathbf{w}(t), \tag{2.52b}$$

which is the main type of system we will investigate in this chapter, one obtains a corresponding input-output representation (2.1) as

$$\mathbf{y}(t) = \mathbf{C}\mathbf{\Phi}(t)\mathbf{x}_0 + \mathbf{D}\mathbf{u}(t) + \int_0^t \mathbf{C}\mathbf{\Phi}(t-\tau)\mathbf{B}\mathbf{u}(\tau)\,\mathrm{d}\tau + \mathbf{w}(t) \tag{2.53}$$

with $\mathbf{\Phi}(t) = \exp(\mathbf{A}t)$. If $\mathbf{x}_0 = 0$, the system is said to be relaxed at $t = 0$. It is often convenient to assume that the system is relaxed in the infinitely remote past, wherefore $\mathbf{C}\mathbf{\Phi}(t)\mathbf{x}_0$ vanishes and (2.53) can be extended to an infinite time horizon

$$\mathbf{y} = \mathbf{G}\mathbf{u} + \mathbf{w} \tag{2.54}$$

and $\mathbf{G}(t) = \mathcal{B}^{-1}\{\mathbf{G}(s)\} = \mathbf{C}\mathbf{\Phi}(t-\tau)\mathbf{B} + \mathbf{D}\delta(t)$ using the Dirac-delta function $\delta(t)$. Note that (2.54) is more general than (2.52), since it can describe dynamic systems that do not have a state-space representation or whose state-space is infinite-dimensional, i.e., distributed-parameter systems described by partial differential equations. Further, the noise term $\mathbf{w}$ can be arbitrary and may thus be used to also include process noise which is not present in (2.52).

Adjoint systems

For a given system $\mathbf{G} : \mathcal{L}_2(\mathbb{R};\mathbb{R}^l) \to \mathcal{L}_2(\mathbb{R};\mathbb{R}^m)$, the *adjoint system* is defined as the linear system $\mathbf{G}^\dagger : \mathcal{L}_2(\mathbb{R};\mathbb{R}^m) \to \mathcal{L}_2(\mathbb{R};\mathbb{R}^l)$ such that

$$\langle \mathbf{G}\mathbf{w}, \mathbf{y}\rangle = \langle \mathbf{w}, \mathbf{G}^\dagger\mathbf{y}\rangle. \tag{2.55}$$

It is easy to show that this uniquely defines $\mathbf{G}^\dagger$ and that $(\mathbf{G}^\dagger)^\dagger = \mathbf{G}$. Furthermore, if a real-valued $\mathbf{G}$ has a state-space representation $(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D})$, then $\mathbf{G}^\dagger$ has a realization $(-\mathbf{A}^\mathrm{T}, -\mathbf{C}^\mathrm{T}, \mathbf{B}^\mathrm{T}, \mathbf{D}^\mathrm{T})$ and the transfer matrix

$$\mathbf{G}^\dagger(s) = \mathbf{G}^\mathrm{T}(-s). \tag{2.56}$$

> *Aufgabe* 2.3. Determine the adjoint $\mathbf{G}^\dagger$ in the Hilbert space $\mathcal{L}_2([0,t_f])$ using (2.55).

2.3 Frequency-domain ILC methods on infinite time horizons

Following the previous section, we consider a linear LTI system that is relaxed in the infinitely remote past and performs the same task over and over again. Each iteration is then described by

$$\mathbf{y}_j = \mathbf{G}\mathbf{u}_j + \mathbf{w}_j \tag{2.57}$$

with the input and output quantities $\mathbf{y}_j(t) \in \mathbb{R}^l$ and $\mathbf{u}_j(t) \in \mathbb{R}^m$, respectively, as well as a exogenous disturbance $\mathbf{w}_j(t) \in \mathbb{R}^l$. We assume that the system $\mathbf{G}$ is inherently BIBO-stable and thus $\mathbf{G}(j\omega) < \infty$ for all ω. Since ILC can be seen as an adaptive open-loop control strategy, this assumption is quite natural. Unstable plants thus have to be stabilized before applying ILC methods. By using a linear ILC law (2.4) on an infinite time horizon, i.e.,

$$\mathbf{u}_{j+1} = \mathbf{Q}(\mathbf{u}_j + \mathbf{L}\mathbf{e}_j) \tag{2.58}$$

with

$$\mathbf{Q}\mathbf{u} = \int_{-\infty}^{\infty} \mathbf{Q}(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau \quad \text{and} \quad \mathbf{L}\mathbf{u} = \int_{-\infty}^{\infty} \mathbf{L}(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau, \tag{2.59}$$

we want to iteratively track a desired output $\mathbf{y}_d$ that exists and is uniquely defined by $\mathbf{y}_d = \mathbf{G}\mathbf{u}_d$. Following a signal processing nomenclature, $\mathbf{Q}$ and $\mathbf{L}$ are usually referred to as *Q-filter* and *L-filter* or *learning filter*, respectively.

2.3.1 Analysis of ILC laws

To analyze stability and convergence of the learning law (2.58), we assume a deterministic input-output behavior with $\mathbf{w}_j = 0$. Using the output error $\mathbf{e}_j = \mathbf{y}_d - \mathbf{y}_j$ and the learning law (2.58) yields

$$\mathbf{u}_{j+1} = \mathbf{\Psi}\mathbf{u}_j + \mathbf{\Lambda}\mathbf{y}_d\ , \tag{2.60}$$

with $\mathbf{\Psi} = \mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})$ and $\mathbf{\Lambda} = \mathbf{Q}\mathbf{L}$. The asymptotic input $\mathbf{u}_{j+1} = \mathbf{u}_j = \mathbf{u}_\infty$ of the input iteration (2.60) is given by

$$\mathbf{u}_\infty = (\mathbf{I} - \mathbf{\Psi})^{-1}\mathbf{\Lambda}\mathbf{y}_d\ . \tag{2.61}$$

By introducing the input error $\bar{\mathbf{u}}_j = \mathbf{u}_j - \mathbf{u}_\infty$, one obtains the input error iteration

$$\bar{\mathbf{u}}_{j+1} = \mathbf{\Psi}\bar{\mathbf{u}}_j\ , \tag{2.62}$$

for which the follow theorem assures stability:

Satz 2.7 (Asymptotic stability of the ILC law)**.** *The input error iteration* (2.62) *of the ILC law* (2.58) *is asymptotically stable if*

$$\sup_\omega \rho\Big(\mathbf{Q}(j\omega)(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega))\Big) < 1 \tag{2.63}$$

and $\mathbf{u}_j$ *converges to* $\mathbf{u}_\infty$*.*

For practical reasons, we are usually much more interested in making stability assertions for the output error $\mathbf{e}_j$. Assuming that there exists a formal inverse $\mathbf{G}^{-1}$, we can rewrite the input-output behavior as $\mathbf{u}_j = \mathbf{G}^{-1}(\mathbf{y}_d - \mathbf{e}_j)$ which yields the output iteration

$$\mathbf{e}_{j+1} = \mathbf{G}\mathbf{\Psi}\mathbf{G}^{-1}\mathbf{e}_j + (\mathbf{I} - \mathbf{G}\mathbf{Q}\mathbf{G}^{-1})\mathbf{y}_d\ . \tag{2.64}$$

It is immediately clear from this equation that perfect tracking, i.e., a vanishing asymptotic output error $\mathbf{e}_\infty = \mathbf{0}$, is only possible for $\mathbf{Q} = \mathbf{I}$ and one may be inclined to question other choices for $\mathbf{Q}$. We will get back to this issue later. The asymptotic output error $\mathbf{e}_\infty$ is given by

$$\begin{aligned} \mathbf{e}_\infty &= \mathbf{y}_d - \mathbf{G}\mathbf{u}_\infty \\ &= \Big(\mathbf{I} - \mathbf{G}(\mathbf{I} - \mathbf{\Psi})^{-1}\mathbf{\Lambda}\Big)\mathbf{y}_d \ . \end{aligned} \tag{2.65}$$

Using $\bar{\mathbf{e}}_j = \mathbf{e}_j - \mathbf{e}_\infty$ yields

$$\bar{\mathbf{e}}_{j+1} = \mathbf{G}\mathbf{\Psi}\mathbf{G}^{-1}\bar{\mathbf{e}}_j \ . \tag{2.66}$$

Since

$$\rho\Big(\mathbf{G}\mathbf{\Psi}\mathbf{G}^{-1}\Big) = \rho(\mathbf{\Psi}), \tag{2.67}$$

it follows that:

Satz 2.8 (Asymptotic stability of the output iteration)**.** *The output iteration* (2.64) *of the ILC law* (2.58) *is asymptotically stable iff the input iteration is stable, i.e.,*

$$\sup_\omega \rho\Big(\mathbf{Q}(j\omega)(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega))\Big) < 1 \tag{2.68}$$

and $\mathbf{e}_j$ *then converges to the asymptotic tracking error*

$$\mathbf{e}_\infty = \Big(\mathbf{I} - \mathbf{G}(\mathbf{I} - \mathbf{\Psi})^{-1}\mathbf{\Lambda}\Big)\mathbf{y}_d = \Big(\mathbf{I} - \mathbf{G}\mathbf{\Psi}\mathbf{G}^{-1}\Big)^{-1}\Big(\mathbf{I} - \mathbf{G}\mathbf{Q}\mathbf{G}^{-1}\Big)\mathbf{y}_d \ . \tag{2.69}$$

Aufgabe 2.4. Show the equivalence of both expressions in (2.69).

If one wants to avoid (potentially) large transient errors during the learning process, monotonic convergence of the learning law has to be ensured.

Satz 2.9 (Monotonic convergence of the input iteration)**.** *The input iteration* (2.60) *of the ILC law* (2.58) *converges monotonically to* $\mathbf{u}_\infty$*, i.e., it holds that*

$$\|\mathbf{u}_{j+1} - \mathbf{u}_\infty\|_2 \leq \alpha\|\mathbf{u}_j - \mathbf{u}_\infty\|_2 \tag{2.70}$$

for $0 \leq \alpha < 1$ *if*

$$\|\mathbf{\Psi}\|_\infty = \sup_\omega \ \bar{\sigma}\Big(\mathbf{Q}(j\omega)(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega))\Big) = \alpha < 1 \ . \tag{2.71}$$

Satz 2.10 (Monotonic convergence of the output iteration)**.** *The output iteration* (2.64) *of the ILC law* (2.58) *converges monotonically to* $\mathbf{e}_\infty$*, i.e., it holds that*

$$\|\mathbf{e}_{j+1} - \mathbf{e}_\infty\|_2 \leq \beta\|\mathbf{e}_j - \mathbf{e}_\infty\|_2 \tag{2.72}$$

for $0 \leq \alpha < 1$ *if*

$$\|\mathbf{G}\boldsymbol{\Psi}\mathbf{G}^{-1}\|_\infty = \sup_\omega \ \bar{\sigma}\Big(\mathbf{G}(j\omega)\mathbf{Q}(j\omega)(\mathbf{I}-\mathbf{L}(j\omega)\mathbf{G}(j\omega))\mathbf{G}^{-1}(j\omega)\Big) = \beta < 1 \ . \quad (2.73)$$

Note that monotonic convergence of the input iteration *does not* imply monotonic convergence of the output iteration and vice versa.

2.3.2 Deterministic design methods

All types of ILC algorithms try to perform some kind of approximate inversion of the input-output behavior of the system. Since this inversion has to be performed online based on measurements, it is inherently constrained by the presence of disturbances and measurement noise. Ideally, an ILC algorithm is able to iteratively separate repeating disturbances and effects of a model-plant mismatch from non-repeating disturbances and measurement noise. Nevertheless, most ILC designs are developed within a deterministic framework with $\mathbf{w} = \mathbf{0}$ which we will adopt in this section. Note that we introduce ILC as a pure open-loop control strategy here, henceforth ILC algorithms are not able to respond to non-repeating (and unanticipated) disturbances. For the performance of the complete control concept, it is thus advisable to incorporate feedback control methods. For simplicity, we thus assume that the system $\mathbf{G}$ already incorporates a suitable feedback control strategy and thus $\mathbf{G}$ is BIBO-stable.

We know from the previous section that $\mathbf{Q} = \mathbf{I}$ is a necessary condition to achieve perfect tracking. One thus only wants do deviate from this ideal if necessary to achieve a stable ILC algorithm that is sufficiently *robust* to model variations. We will thus start with three typical design strategies to obtain suitable learning operators before considering the relation between Q-filtering and robustness. As we will see later, good learning filters usually avoid to learn at high frequencies due to the presence of measurement noise. By assuming $\mathbf{Q} = \mathbf{I}$, however, $\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega) \to \mathbf{I}$ for $\omega \to \infty$, which would always violate the stability condition in Theorem 2.7. We will therefore consider a finite frequency range up to some $\bar{\omega}$ and assume $\mathbf{Q} = \mathbf{0}$ above, which is always possible.

PD-type ILC laws

As the name implies, PD-type learning laws combine proportional and derivative action, i.e,

$$\mathbf{L}\mathbf{e}_j = \mathbf{K}_p\mathbf{e}_j(t) + \mathbf{K}_d\frac{\mathrm{d}\mathbf{e}_j(t)}{\mathrm{d}t} \quad (2.74)$$

with the proportional and derivative gain matrices $\mathbf{K}_p$ and $\mathbf{K}_d$, respectively, that are tunable parameters that have to be chosen such that a desired performance is reached. These heuristic designs are arguably the most widely used ILC laws in the literature since they do not require an accurate model but rely on intensive tuning of the gain matrices. Since this can be a quite tedious task for MIMO systems, PD-type ILC laws are typically used for SISO systems only, which yields the learning filter

$$\mathbf{L}(s) = k_p + k_d s \quad (2.75)$$

in the frequency domain. While there are no tuning guidelines, it is usually suggested to start with small gain values and increase them until a growth of (usually) high-frequency components indicate that the learning law left the stable parameter regime [2.8].

The fact that such simple ILC laws perform quite well for a large class of systems is somewhat surprising. However, this can be explained by observing that a PD-type laws (2.74) is the exact inverse of a first-order lag (i.e., P-T_1) element. Since most technical systems exhibit some kind of low-pass behavior, PD-type laws can be seen as an approximate inverse of their behavior. Note that the learning filter (2.75) yields infinite gains for $\omega \to \infty$, which is problematic in the presence of measurement noise.

Beispiel 2.1. Consider a system described by the scalar transfer function

$$G(s) = \frac{s + 1/2}{s^2 + s + 3} \tag{2.76}$$

together with the PD-type law (2.75). To evaluate stability for different parameter choices of k_p and k_d, the absolute value of $I - L(j\omega)G(j\omega)$ is plotted in Fig. 2.3 together with the resulting convergence behavior of the ILC law. Note that the ILC law ultimately reaches the precision of the numerical simulation. Observing that the inverse of the system's transfer function is

$$G^{-1}(s) = 1/2 + s + \frac{11}{2(2s+1)}, \tag{2.77}$$

one could try to use $k_p = 1/2$ and $k_d = 1$, which is in fact unstable according to Theorem 2.7.

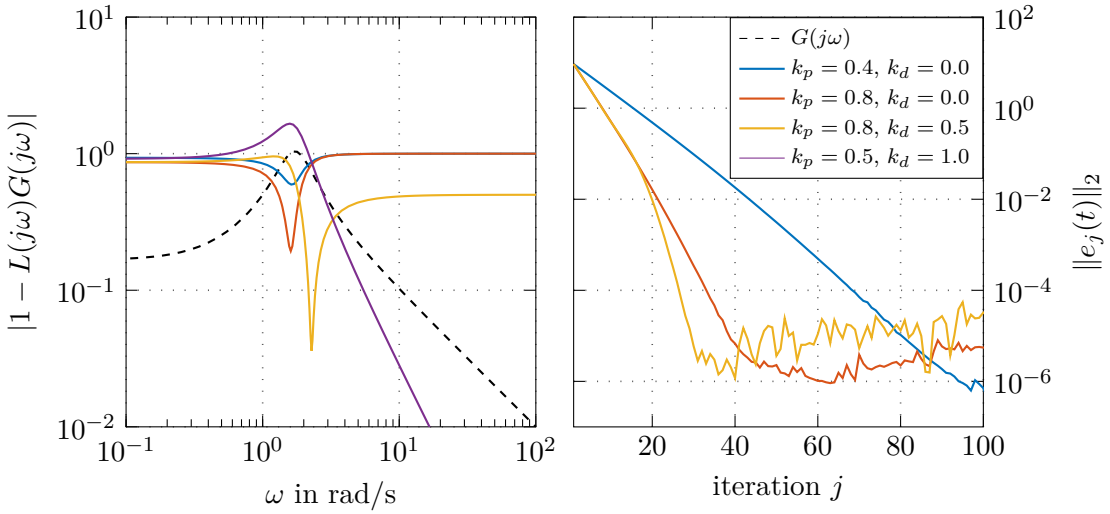


Figure 2.3: Stability and convergence behavior of a PD-type learning law for different parameters k_p and k_d.

Inversion-based ILC law

A particularly intuitive choice is to use $\mathbf{L} = \mathbf{G}^{-1}$, which yields

$$\mathbf{u}_{j+1} = \mathbf{Q}\Big(\mathbf{u}_j + \mathbf{G}^{-1}\mathbf{e}_j\Big) = \mathbf{Q}\Big(\mathbf{u}_j + \mathbf{G}^{-1}(\mathbf{y}_d - \mathbf{G}\mathbf{u}_j)\Big) = \mathbf{Q}\Big(\mathbf{G}^{-1}\mathbf{y}_d\Big). \tag{2.78}$$

The input $\mathbf{u}_j$ thus remains unchanged after the first iteration (dead-beat behavior) with a resulting output error

$$\mathbf{e}_{j+1} = \mathbf{e}_\infty = \mathbf{y}_d - \mathbf{G}\mathbf{Q}\Big(\mathbf{G}^{-1}(\mathbf{y}_d - \mathbf{y}_0)\Big). \tag{2.79}$$

Using $\mathbf{Q} = \mathbf{I}$ further implies $\mathbf{e}_\infty = \mathbf{0}$. There are several caveats to this result: First of all, $\mathbf{G}$ is never known exactly which limits the usability of inversion-based ILC laws and mandates an additional $\mathbf{Q}$-filter. Further, $\mathbf{G}(s)$ is strictly proper for any physical process, wherefore its inverse $\mathbf{G}^{-1}(s)$ is unbound for $s = j\omega \to \infty$ and thus $\mathbf{e}_\infty$ is undefined for arbitrary $\mathbf{y}_d - \mathbf{y}_0 \in \mathcal{L}_2(\mathbb{R})$. Finally, $\mathbf{G}^{-1}$ is unstable for non-minimum phase systems although this can be alleviated by (non-causal) stable-inversion methods [2.9].

Pseudo-Inversion-based ILC law

Using an inversion-based ILC law is essentially determining an exact feedforward input signal from output data. Unlike for feedforward purposes, however, we cannot demand certain levels of regularity of the output signals due to the presence of stochastic disturbances and measurement noise. The obvious solution is to regularize the system inversion by using a pseudo-inverse learning filter [2.10], i.e.,

$$\mathbf{L}(s) = \Big(\alpha\mathbf{I} + \mathbf{G}^\dagger(s)\mathbf{G}(s)\Big)^{-1}\mathbf{G}^\dagger(s) \tag{2.80}$$

with the regularization parameter $\alpha > 0$ and the transfer matrix of the adjoint system $\mathbf{G}^\dagger(s)$. The regularization parameter separates regions where the learning law is approaching an inversion-based law (i.e., $\mathbf{L}(s) \approx \mathbf{G}^{-1}(s)$) from regions where learning is almost prohibited (i.e., $\mathbf{L}(s) \approx \mathbf{0}$) depending on whether $\|\mathbf{G}^\dagger(s)\mathbf{G}(s)\|$ is much larger or smaller than α, respectively. For systems with low-pass behavior where $\mathbf{G}(s)$ is strictly proper, the learning filter (2.80) thus naturally avoids learning in the high-frequency region, i.e., $\lim\limits_{s\to\infty} \mathbf{L}(s) = \mathbf{0}$. It still remains to be shown that (2.80) results in a stable learning iteration. Using

$$\mathbf{I} - \mathbf{L}\mathbf{G} = \mathbf{I} - \Big(\alpha\mathbf{I} + \mathbf{G}^\dagger\mathbf{G}\Big)^{-1}\mathbf{G}^\dagger\mathbf{G} = \Big(\mathbf{I} + \frac{1}{\alpha}\mathbf{G}^\dagger\mathbf{G}\Big)^{-1} \tag{2.81}$$

it follows with the definition of the adjoint system that

$$\rho(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega)) = \rho\left(\Big(\mathbf{I} + \frac{1}{\alpha}(\mathbf{G}(-j\omega))^T\mathbf{G}(j\omega)\Big)^{-1}\right) \tag{2.82}$$

$$= \rho\left(\Big(\mathbf{I} + \frac{1}{\alpha}(\mathbf{G}(j\omega))^H\mathbf{G}(j\omega)\Big)^{-1}\right). \tag{2.83}$$

Assuming that $\mathbf{G}(j\omega)\mathbf{u} \neq \mathbf{0}$ for any $\mathbf{u} \neq \mathbf{0}$, the matrix $\frac{1}{\alpha}(\mathbf{G}(j\omega))^H\mathbf{G}(j\omega)$ is positive definite and thus all of its eigenvalues are strictly positive. This implies the asymptotic stability of the learning law since $\rho(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega)) < 1$ for all ω.

Beispiel 2.2. Consider again the plant (2.76). According to Figure 2.3, the magnitude of $G(j\omega)$ is above $1 \cdot 10^{-1}$ for the essential part of its dynamic behavior. The resulting Pseudo-Inversion-based learning filter (2.80) for different choices of the regularization parameter $\alpha = \{1 \cdot 10^{-1}, 5 \cdot 10^{-2}, 1 \cdot 10^{-2}, 1 \cdot 10^{-3}\}$ is illustrated in Figure 2.4. As one can see, the learning filter (2.80) is an approximation of the exact inverse $G^{-1}(j\omega)$. While $L(j\omega)$ deviates (significantly) over the whole frequency range for high values of α, small values of α only introduce a roll-off for high-frequency components. Since asymptotic convergence is equivalent to monotonic convergence for SISO plants according to Theorem 2.7 and Theorem 2.9, the learning law (2.80) always converges exponentially. Note that this holds only true for infinite time horizons and that the convergence rate in Theorem (2.9) is in general not equivalent (but determined by) the regularization parameter α.

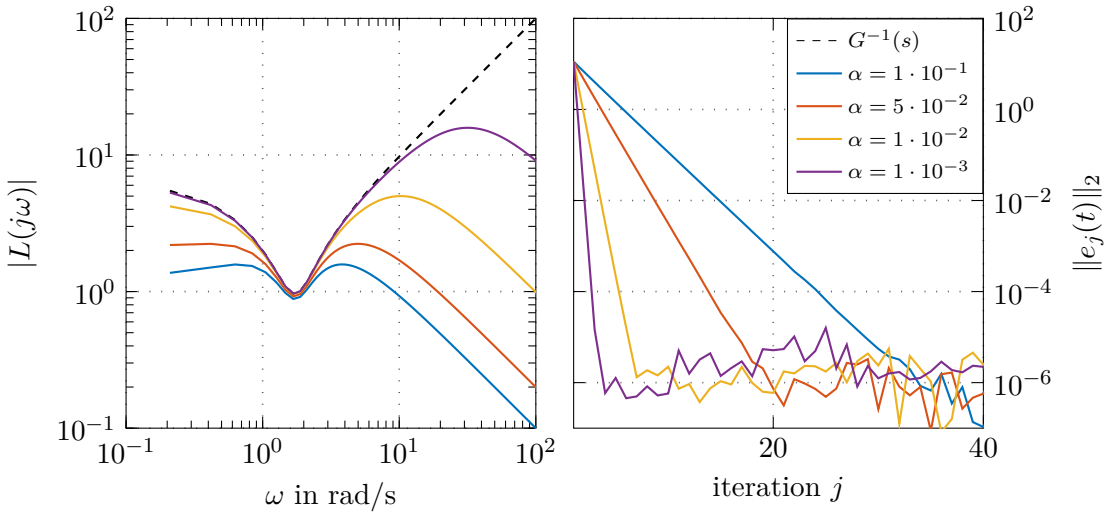


Figure 2.4: Learning filter and convergence behavior of the Pseudo-Inversion-based learning filter (2.80) for different regularization parameters α.

2.3.3 Stochastically optimal learning laws

We already mentioned in the previous section that the approximate inversion due to the learning filter $\mathbf{L}$ is inherently constrained by the presence of stochastic quantities, namely (non-repeating) process disturbances and measurement noise $\mathbf{w}_j(t) \in \mathbb{R}^l$ in

$$\mathbf{y}_j(t) = \mathbf{G}\mathbf{u}_j(t) + \mathbf{w}_j(t) \tag{2.84}$$

that is given by a zero-mean wide-sense stationary (WSS) process with identical stochastic properties in each iteration. To analyze this problem in a stochastic framework, we

introduce the following notation: For two vector-valued stochastic signals $\mathbf{a}_k(t)$ and $\mathbf{b}_l(t)$ with the iteration indices k and l, their cross-correlation function is given by $\mathbf{R}^{a_k b_l}(\tau) = \mathrm{E}\Big\{\mathbf{a}_k(t+\tau)(\mathbf{b}_l(t))^{\mathrm{T}}\Big\}$ where $\mathrm{E}\{\cdot\}$ denotes the expectation operator. The corresponding power spectral density (PSD) reads as $\mathbf{S}^{a_k b_l}(s) = \mathcal{B}\Big\{\mathbf{R}^{a_k b_l}(\tau)\Big\}$. In case $k = l$, the common index is written as subscript, i.e., $\mathbf{R}_k^{ab}(\tau) = \mathbf{R}^{a_k b_k}(\tau)$ and $\mathbf{S}_k^{ab}(s) = \mathbf{S}^{a_k b_k}(s)$.

For generality, we use a *iteration-varying* learning law (again without $\mathbf{Q}$-filter)

$$\mathbf{u}_{j+1}(t) = \mathbf{u}_j(t) + \mathbf{L}_j \mathbf{e}_j(t). \tag{2.85}$$

Our goal now is to design a learning filter $\mathbf{L}_j$ in a stochastically optimal way, i.e. $\mathbf{u}_{j+1}(t)$ should be determined such that it minimizes the expected value of the mean-square output error $\mathrm{E}\Big\{(\mathbf{e}_{j+1}(t))^{\mathrm{T}}\mathbf{e}_{j+1}(t)\Big\}$. Following section 2.3.1, the output error $\mathbf{e}_j(t) = \mathbf{y}_d(t) - \mathbf{y}_j(t)$ of the iteration j is given by

$$\mathbf{e}_j(t) = \mathbf{G}\boldsymbol{\nu}_j(t) - \mathbf{w}_j(t) \tag{2.86}$$

using the input error $\boldsymbol{\nu}_j(t) = \mathbf{u}_d(t) - \mathbf{u}_j(t)$. Together with the learning law, the evolution of the input error is described by

$$\boldsymbol{\nu}_{j+1}(t) = \boldsymbol{\nu}_j(t) - \mathbf{L}_j \mathbf{e}_j(t) \tag{2.87}$$

and the output error of the iteration $j+1$ yields

$$\mathbf{e}_{j+1}(t) = (\mathbf{I} - \mathbf{G}\mathbf{L}_j)\mathbf{e}_j(t) + \mathbf{w}_j(t) - \mathbf{w}_{j+1}(t). \tag{2.88}$$

The problem of learning in a stochastically optimal sense can be written as the optimization problem

$$\min_{\mathbf{L}_j(t)} \ \mathrm{E}\Big\{(\mathbf{e}_{j+1}(t))^{\mathrm{T}}\mathbf{e}_{j+1}(t)\Big\}. \tag{2.89}$$

Since we only consider LTI systems and the stochastic disturbance $\mathbf{w}_j$ is WSS, the optimization problem (2.89) can be treated in the Laplace domain by applying the bilateral Laplace transform, which yields

$$\begin{aligned}\min_{\mathbf{L}_j(t)} \ \mathrm{E}\Big\{(\mathbf{e}_{j+1}(t))^{\mathrm{T}}\mathbf{e}_{j+1}(t)\Big\} &= \min_{\mathbf{L}_j(t)} \mathrm{Tr}\left\{\mathbf{R}_{j+1}^{ee}(0)\right\} \\ &= \min_{\mathbf{L}_j(s)} \frac{1}{j2\pi}\mathrm{Tr}\left\{\int_{-\infty}^{\infty} \mathbf{S}_{j+1}^{ee}(j\omega)\,\mathrm{d}\omega\right\},\end{aligned} \tag{2.90}$$

where $\mathrm{Tr}\{\cdot\}$ denotes the trace operator. To obtain the output error PSD $\mathbf{S}_{j+1}^{ee}(s)$, one can use (2.88) together with (2.86), which yields

$$\mathbf{e}_{j+1}(t) = (\mathbf{G} - \mathbf{G}\mathbf{L}_j\mathbf{G})\boldsymbol{\nu}_j(t) + \mathbf{G}\mathbf{L}_j\mathbf{w}_j(t) - \mathbf{w}_{j+1}(t). \tag{2.91}$$

In general, the stochastic quantities $\boldsymbol{\nu}_j$, $\mathbf{w}_j$ and $\mathbf{w}_{j+1}$ will be correlated. We thus make the following assumptions:

A1 different instances of the disturbance are uncorrelated, i.e., $\mathrm{E}\Big\{\mathbf{w}_i(t+\tau)(\mathbf{w}_j(t))^{\mathrm{T}}\Big\} = \mathbf{0}$ for $i \neq j$

A2 the input error $\boldsymbol{\nu}_j$ is uncorrelated with the exogenous disturbance of the current and the following iteration, i.e., $\mathrm{E}\Big\{\boldsymbol{\nu}_j(t+\tau)(\mathbf{w}_i(t))^{\mathrm{T}}\Big\} = \mathbf{0}$ for $i \in \{j, j+1\}$. Together with A1, this is equivalent to $\mathrm{E}\Big\{\boldsymbol{\nu}_0(t+\tau)(\mathbf{w}_j(t))^{\mathrm{T}}\Big\} = \mathbf{0}$

A3 the stochastic properties of of the disturbance do not change over iterations, i.e., $\mathrm{E}\Big\{\mathbf{w}_j(t+\tau)(\mathbf{w}_j(t))^{\mathrm{T}}\Big\} = \mathbf{S}^{ww}$.

Using these assumption, we obtain

$$\mathbf{S}^{ee}_{j+1} = (\mathbf{G} - \mathbf{G}\mathbf{L}_j\mathbf{G})\,\mathbf{S}^{\nu\nu}_j\,(\mathbf{G} - \mathbf{G}\mathbf{L}_j\mathbf{G})^{\dagger} + \mathbf{G}\mathbf{L}_j\,\mathbf{S}^{ww}\,(\mathbf{L}_j)^{\dagger}\mathbf{G}^{\dagger} + \mathbf{S}^{ww} \tag{2.92}$$

The optimization problem (2.90) can be solved by variational calculus. The resulting learning filter

$$\mathbf{L}_k(s) = \mathbf{S}^{\nu\nu}_j(s)\mathbf{G}^{\dagger}(s)\Big[\mathbf{G}(s)\mathbf{S}^{\nu\nu}_j(s)\mathbf{G}^{\dagger}(s) + \mathbf{S}^{ww}(s)\Big]^{-1} \tag{2.93}$$

is an iterative version of the well-known (non-causal) Wiener filter, which is used to *estimate* the input deviation that optimally *explains* the measured output error. A common problem of Wiener-filter-based approaches is that the optimal solution (2.93) requires knowledge of the input error PSD $\mathbf{S}^{\nu\nu}_j(s)$, which is typically handled using a-priori knowledge of the problem.

Due to the learning process, the measured output error $\mathbf{e}_j(t)$ will be increasingly dominated by stochastic disturbances. A stochastically optimal learning law will therefore reduce its learning action with increasing iterations. Such a behavior is intrinsic to (2.93) due to the decreasing input error PSD $\mathbf{S}^{\nu\nu}_j(s)$. From (2.86), (2.87) and (2.93) we obtain the iterative relation

$$\mathbf{S}^{\nu\nu}_{j+1}(s) = (\mathbf{I} - \mathbf{L}_j(s)\mathbf{G}(s))\mathbf{S}^{\nu\nu}_j(s). \tag{2.94}$$

By starting from an initial PSD $\mathbf{S}^{\nu\nu}_0(s)$ and the corresponding learning filter (2.93), one can iterate forward in time to obtain a stochastically optimal ILC method.

Satz 2.11 (Stochastically optimal learning)**.** *If $\mathbf{S}^{ww}(s)$ and the initial input error PSD $\mathbf{S}^{\nu\nu}_0(s)$ are positive definite and the system's transfer matrix $\mathbf{G}(s)$ does not exhibit transmission zeros on the imaginary axis, the learning filter (2.93) together with (2.94) yields a stable learning law that ensures convergence to the optimal error PSDs*

$$\mathbf{S}^{\nu\nu}_{\infty}(s) = \mathbf{0}, \quad \mathbf{S}^{\eta\eta}_{\infty}(s) = \mathbf{S}^{ww}(s). \tag{2.95}$$

The fact that one can iteratively construct a noise-less representation of the unknown desired input $\mathbf{u}_{\infty}$ from noisy output measurements and an output error containing only noise sounds like a very attractive solution. However, there is a severe limitation to this seemingly nice result: By design, the learning filter is asymptotically vanishing and thus the learning process comes to a halt. Since the forward iteration (2.94) is independent

of actual measurements, this is the case even if some unforeseen disturbance (i.e., not represented in the stochastic model) is causing large output errors.

Since iteration-varying learning laws furthermore increase the effort of implementation, one may thus prefer to accept sub-optimal performance by using a fixed input error PSD $\mathbf{S}_j^{\nu\nu}(s) = \mathbf{S}^{\nu\nu}(s)$.

Satz 2.12 (Stochastically sub-optimal learning)**.** *If $\mathbf{S}^{ww}(s)$ and the chosen input error PSD $\mathbf{S}^{\nu\nu}(s)$ are positive definite and the system's transfer matrix $\mathbf{G}(s)$ does not exhibit transmission zeros on the imaginary axis, the iteration-invariant learning filter*

$$\mathbf{L}(s) = \mathbf{S}^{\nu\nu}(s)\mathbf{G}^{\dagger}(s)\Big[\mathbf{G}(s)\mathbf{S}^{\nu\nu}(s)\mathbf{G}^{\dagger}(s) + \mathbf{S}^{ww}(s)\Big]^{-1} \tag{2.96}$$

yields a stable learning law that converges to a positive definite asymptotic output error PSD $\mathbf{S}_{\infty}^{ee}(s)$ given by the solution of

$$(\mathbf{I}-\mathbf{G}(s)\mathbf{L}(s))\mathbf{S}_{\infty}^{ee}(s)(\mathbf{I}-\mathbf{G}(s)\mathbf{L}(s))^{\dagger} - \mathbf{S}_{\infty}^{ee}(s) + \mathbf{G}\mathbf{L}(s)\mathbf{S}^{ww}(s) + \mathbf{S}^{ww}(s)\mathbf{L}^{\dagger}(s)\mathbf{G}^{\dagger}(s) = \mathbf{0}. \tag{2.97}$$

This sub-optimal learning law is structurally similar to pseudo-inversion-based learning laws. For the special case of $\mathbf{S}^{\nu\nu}(s) = \sigma_{\nu}\mathbf{I}$ and $\mathbf{S}^{ww}(s) = \sigma_{w}\mathbf{I}$, it is easy to show that (2.96) is identical to (2.80) with $\alpha = \frac{\sigma_w}{\sigma_\nu}$. Stochastically optimal learning filters thus regularize the system inversion according to the expected signal-to-noise ratio.

2.3.4 Q-filtering and robustness

One of the main advantages of ILC over other control approaches is its ability to achieve (almost) perfect tracking in the presence of external disturbances and model-plant mismatch. While we have investigated external disturbances in the previous section, we may now shift our attention to an inevitable model-plant mismatch. Assuming that the plant can be described by $\mathbf{G}(s) = \mathbf{G}_0(s)\Delta\mathbf{G}(s)$ with the nominal (design) model $\mathbf{G}_0(s)$ and the unknown deviation $\Delta\mathbf{G}(s)$, it is expected that $\rho(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}_0(s)\Delta\mathbf{G}(s))$ will in general be larger than one for some ω since we designed the learning filter $\mathbf{L}(s)$ such that $\rho(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}_0(s)) < 1$. However, it is clear that the asymptotic stability criterion

$$\sup_{\omega} \rho\Big(\mathbf{Q}(j\omega)(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}(j\omega))\Big) < 1 \tag{2.98}$$

can always be met by choosing $\mathbf{Q}(s)$ sufficiently small. For example, for a known $\Delta\mathbf{G}(s)$ one could always use $\mathbf{Q}(s) = \kappa\mathbf{I}$ with $\kappa > 1/\sup_{\omega} \rho(\mathbf{I} - \mathbf{L}(j\omega)\mathbf{G}_0(j\omega)\Delta\mathbf{G}(j\omega))$.

Using a spectrally uniform $\mathbf{Q}$-filter is usually not recommended, since $\Delta\mathbf{G}(s)$ is typically small in those parts of the frequency range within which we want to learn. To avoid unnecessary large asymptotic tracking errors, a spectrally selective $\mathbf{Q}$-filter is thus usually advisable. The most common case in practice is that the (identified) nominal model fits quite accurately up to a certain frequency ω_c and starts to deteriorate beyond that frequency, which motivates the use of low-pass filters.

Beispiel 2.3. Consider the plant (2.76) again with an additional model-plant mismatch given by

$$G_0(s) = \frac{s+1/2}{s^2+s+3} \quad \text{and} \quad \Delta G(s) = \frac{1}{1+s/30+(s/15)^2} \tag{2.99}$$

and a pseudo-inverse learning law $L(s)$ with $\alpha = 1 \cdot 10^{-2}$ for the nominal model $G_0(s)$. Simulation scenarios for different choices of $Q(s)$ are illustrated in Figure 2.5. For $Q(s) = 1$, we can see that $|1 - L(j\omega)G(j\omega)| > 1$ for $\omega > 10$, which leads to high-frequency oscillations that build up over time. Note that the error seems to converge initially but diverges after 20 iterations. Using a simple first-order low-pass filter

$$Q(s) = \frac{10}{s+10} \tag{2.100}$$

stabilizes the learning iteration, but at the cost of a significantly higher asymptotic output error. Interestingly, for the more aggressive choice (see left-hand side of Figure 2.5)

$$Q(s) = \frac{10^2}{(s+10)(-s+10)}, \tag{2.101}$$

the asymptotic output error is vastly improved. The main reason for this is that simple first-order low-pass filter introduces a phase shift to the learned signal, which in turn results in a slight temporal mismatch between $y_d(t)$ and $y_\infty(t)$ that dominates all other error sources. The latter choice of $Q(s)$ is a so-called *zero-phase* filter.

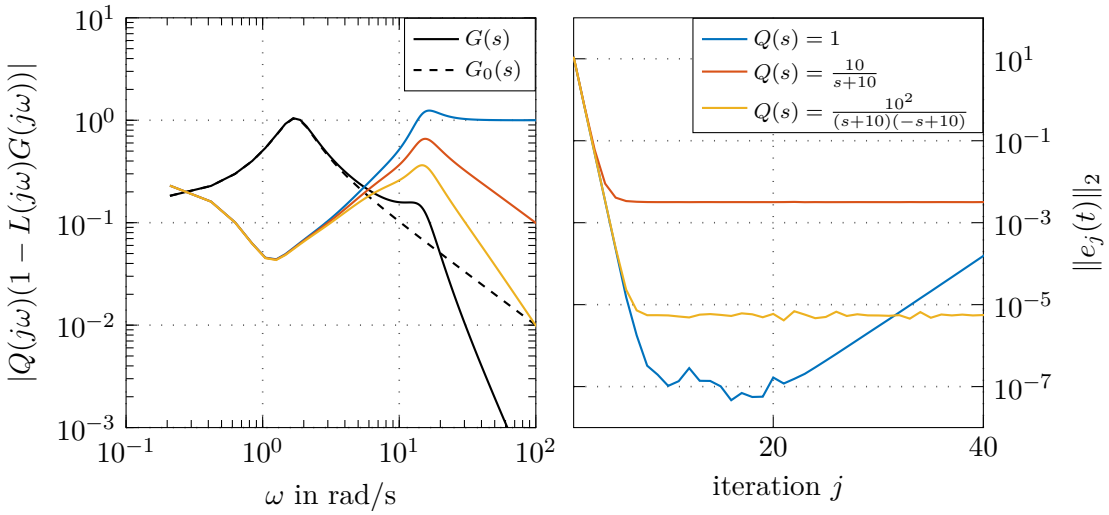


Figure 2.5: Robustness to model-plant mismatch by using a **Q**-filter.

Since there is no systematic design for **Q** in the MIMO case, we will restrict ourselves to the SISO case in the following. In line with Example 2.3, it is generally recommended

to use *zero-phase* filters for $Q(s)$, i.e.,

$$\mathrm{Im}\{Q(j\omega)\} = 0. \tag{2.102}$$

> *Aufgabe* 2.5. Show that a zero-phase filter $\mathbf{Q}(s)$ that is not a constant gain matrix is necessarily noncausal.

Gaussian filter

A particularly simple type of zero-phase filter is the Gaussian filter given by the impulse response

$$Q(t) = \frac{1}{\sigma_q\sqrt{2\pi}}\exp\left(-\frac{t^2}{2\sigma_q^2}\right) \tag{2.103}$$

with the standard deviation σ_q that is related to the 3-dB bandwidth f_c

$$\sigma_q = \frac{\sqrt{\ln(2)}}{2\pi f_c} \tag{2.104}$$

of the corresponding frequency-domain representation

$$Q(j\omega) = \exp\left(-\frac{\sigma_q^2\omega^2}{2}\right). \tag{2.105}$$

Forward-Backward-Filtering

A popular alternative with arbitrary spectral behavior is to use

$$Q(s) = Q_f^\dagger(s)Q_f(s) \tag{2.106}$$

where $Q_f(s)$ is stable and causal. As a result, $Q(s)$ can be easily implemented by integrating the realization of $Q_f(s)$ forward and $Q_f^\dagger(s)$ backward in time (see MATLAB command `filtfilt`). This procedure was used in Example 2.3 to implement the zero-phase Q-filter.

Time delay

One particular type of model-plant mismatch in ILC applications is a temporal delay by time T, i.e.,

$$\Delta G(s) = \exp(-sT). \tag{2.107}$$

Using a pseudo-inversion-based learning law, it follows that

$$\begin{aligned} |1 - L(j\omega)G(j\omega)| &= |1 - L(j\omega)G_0(j\omega)\exp(-j\omega T)| \\ &= \left|1 - \frac{|G_0(j\omega)|^2}{\alpha + |G_0(j\omega)|^2}\exp(-j\omega T)\right|. \end{aligned} \tag{2.108}$$

As one can see, the resulting ILC law is always unstable in the absence of a Q-filter for sufficiently high frequencies ω. Any predictable time-delay should thus be compensated by shifting the time axis respectively.

2.3.5 Implementation aspects

After having designed a suitable learning filter $\mathbf{L}(s)$, there are still some open questions on how to implement the resulting learning law for a practical problem that is typically defined on a finite time horizon $t \in [0, t_f]$.

Spectral factorization

A direct method to achieve this is to use spectral factorization methods to decompose the learning filter into the product of two parts: one that is causal (i.e., its impulse response matrix is in $\mathcal{H}_2$), and one that is anti-causal (i.e., its impulse response matrix is in $\mathcal{H}_2^\perp$). Since the anti-causal part can also be written as the adjoint of a causal system, one obtains the factorization

$$\mathbf{L}(s) = \mathbf{L}_b^\dagger(s)\mathbf{L}_f(s) \tag{2.109}$$

As a result, the learning filter $\mathbf{L}(s)$ can be implemented by finding a state-space representation of $\mathbf{L}_f(s)$ that is solved forward in time and a state-space representation of $\mathbf{L}_b^\dagger(s)$ that is solved backwards in time. Note that this solution procedure requires corresponding initial and terminal conditions for the state-space representations that are not determined from the infinite time-horizon design.

FIR-filter approximation

Finding a factorization (2.109) can be quite tedious except for SISO systems. An alternative solution is to determine the impulse response matrix $\mathbf{L}(t)$ directly. Since $\mathbf{L}(t)$ is in general not of finite support, it has to be truncated in time to a finite-impulse response (FIR) approximation to implement the convolution (2.59). Along this line, $\mathbf{L}(t)$ can either be transformed back analytically using the inverse bilateral Laplace transform and truncated afterwards or one chooses to use numerical methods. By assumption, the system's transfer matrix $\mathbf{G}(s)$ has no poles on the imaginary axis $s = j\omega$, wherefore $\mathbf{L}(s)$ has a ROC around the imaginary axis where $\mathbf{L}(t)$ is given by

$$\mathbf{L}(t) = \mathcal{F}^{-1}\{\mathbf{L}(j\omega)\}. \tag{2.110}$$

A convolution with $\mathbf{L}(t)$ is therefore equivalent to a forward and backward integration of the spectral factorization (2.109). The continuous Fourier transform can be approximately computed using the discrete Fourier transform (DFT) on a sampled temporal and spectral grid. By assuming that $\mathbf{L}(j\omega) = \mathbf{0}$ outside the finite interval $[-\frac{\omega_s}{2}, \frac{\omega_s}{2}]$, we can approximate the inverse Fourier transform for $t \in [-\frac{t_s}{2}, \frac{t_s}{2}]$ on the discrete grids $t_n = (n - N/2)\frac{t_s}{N}$ and

$\omega_m = (m - {}^N/_2)\frac{\omega_s}{N}$ with $0 \le n, m < N$ by

$$\begin{aligned}\mathbf{L}(t_n) &= \frac{1}{2\pi}\int_{-\omega_s/2}^{\omega_s/2} \mathbf{L}(j\omega)\exp[j\omega t_n]\,\mathrm{d}\omega \\ &\approx \frac{\omega_s}{2\pi N}\sum_{m=0}^{N-1}\mathbf{L}(j\omega_m)\exp\left[j\frac{t_s\omega_s}{N^2}\left(n-\frac{N}{2}\right)\left(m-\frac{N}{2}\right)\right] \\ &= \frac{\omega_s}{2\pi N}\exp\left[-j\frac{t_s\omega_s}{2N}\left(n-\frac{N}{2}\right)\right]\sum_{m=0}^{N-1}\mathbf{L}(j\omega_m)\exp\left[-j\frac{t_s\omega_s}{2N}m\right]\exp\left[j\frac{t_s\omega_s}{N^2}mn\right]\end{aligned} \tag{2.111}$$

Calculating the right-hand side of (2.111) can be computationally quite expensive. For the case $\omega_s t_s = 2\pi N$, however, this can be reformulated as[1]

$$\begin{aligned}\mathbf{L}(t_n) &\approx \frac{\omega_s}{2\pi N}\exp\left[-j\pi\left(n-\frac{N}{2}\right)\right]\sum_{m=0}^{N-1}\mathbf{L}(j\omega_m)\exp[-j\pi m]\exp\left[j\frac{2\pi}{N}mn\right] \\ &= \frac{\omega_s}{2\pi N}\exp\left[-j\pi\left(n-\frac{N}{2}\right)\right]\mathrm{DFT}\{\mathbf{L}(j\omega_m)\exp[-j\pi m]\}\end{aligned} \tag{2.112}$$

which is computationally very efficient by using FFT algorithms to compute the DFT for N chosen as a power of 2. Note that this method does not require $\mathbf{L}(s)$ to be a rational transfer matrix, see Example 2.4.

Boundary effects

Finally, to implement such learning laws requires information that is not provided by the infinite time horizon design. This lack was already quite explicit when factorizing the learning law and finding state-space representations for forward and backward integration, where suitable boundary conditions at $t = 0$ and $t = t_f$ are required. Using the convolutional representation (2.59) with a FIR filter of length t_s, i.e., $\mathbf{L}(t) \neq \mathbf{0}$ only if $t \in [-{}^{t_s}/_2, {}^{t_s}/_2]$, yields

$$(\mathbf{L}\mathbf{e}_j)(t) = \int_{-t_s}^{t_f+t_s}\mathbf{L}(t-\tau)\mathbf{e}_j(\tau)\,\mathrm{d}\tau \quad \text{for} \quad t \in [0, t_f]. \tag{2.113}$$

Rather than additional boundary values, the convolutional representation requires values of $\mathbf{e}_j(t)$ for $t \in [-t_s, t_f + t_s]$, which extends beyond the available measurements. Since we assumed that the system is correctly (and identically) initialized, $\mathbf{e}_j(0) = \mathbf{0}$ and thus $\mathbf{e}_j(t) = \mathbf{0}$ for $t \in [-t_s, 0]$ is a obvious choice. For $t = t_f$, the two most widely used options are:

1. $\mathbf{e}(t) = 0$ for $t \in [t_f, t_f + t_s]$ (truncation)
2. $\mathbf{e}(t) = \mathbf{e}(t_f)$ for $t \in [t_f, t_f + t_s]$ (extension)

[1] This links the temporal resolution ${}^{t_s}/_N$ with the spectral resolution ${}^{\omega_s}/_N$, which can be problematic for some applications with large bandwidths where a sufficient temporal resolution requires a very high number of discretization points.

Note that this choice will impact the stability of the learning law in general [2.11]. However, if an infinite time horizon convolution operator $\mathbf{\Psi}$ according to

$$\mathbf{\Psi}\mathbf{e} = \int_{-\infty}^{\infty} \mathbf{\Psi}(t-\tau)\mathbf{e}(\tau)\,\mathrm{d}\tau, \tag{2.114}$$

with $\mathbf{e} \in \mathcal{L}_2(-\infty, \infty)$ is a contraction mapping, i.e., $\|\mathbf{\Psi}\mathbf{e}\|_2 < \|\mathbf{e}\|_2$, then its truncated version

$$\mathbf{\Psi}_T\mathbf{e}' = \int_{0}^{t_f} \mathbf{\Psi}(t-\tau)\mathbf{e}'(\tau)\,\mathrm{d}\tau \tag{2.115}$$

is a contraction mapping on the truncated space $\mathbf{e}' \in \mathcal{L}_2([0, t_f])$, too. This can be seen by using the standard truncation operator

$$(\mathbf{T}\mathbf{e})(t) = \begin{cases} \mathbf{e}(t) & \text{for} \quad t \in [0, t_f], \\ \mathbf{0} & \text{otherwise}, \end{cases} \tag{2.116}$$

since

$$\|\mathbf{\Psi}_T\mathbf{e}'\|_{2,[0,T]} = \|\mathbf{T}\mathbf{\Psi}\mathbf{T}\mathbf{e}\|_2 \leq \|\mathbf{\Psi}\mathbf{T}\mathbf{e}\|_2 \leq \|\mathbf{\Psi}\|_\infty \|\mathbf{T}\mathbf{e}\|_2 = \|\mathbf{\Psi}\|_\infty \|\mathbf{e}'\|_{2,[0,T]} \tag{2.117}$$

using the identity $\mathbf{\Psi}_T = \mathbf{T}\mathbf{\Psi}\mathbf{T}$ and the fact that the induced norm $\|\mathcal{T}\|_\infty = 1$.

Beispiel 2.4. Consider the coupled system of parabolic PDEs

$$\frac{\partial \mathbf{x}}{\partial t}(z,t) = \underbrace{\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}}_{=\mathbf{\Lambda}} \frac{\partial^2}{\partial z^2}\mathbf{x}(z,t) + \underbrace{\begin{bmatrix} 0 & \sigma_{12} \\ \sigma_{21} & 0 \end{bmatrix}}_{=\mathbf{\Sigma}} \mathbf{x}(z,t) \tag{2.118a}$$

defined on the spatial domain $z \in [0, 1]$, with the initial condition $\mathbf{x}(z, 0) = \mathbf{0}$, the boundary conditions

$$\mathbf{x}(0,t) = \mathbf{u}(t) \qquad \frac{\partial \mathbf{x}}{\partial z}(1,t) = \underbrace{\begin{bmatrix} \Gamma_{11} & \Gamma_{12} \\ \Gamma_{21} & \Gamma_{22} \end{bmatrix}}_{\mathbf{\Gamma}} \mathbf{x}(1,t), \tag{2.118b}$$

and the output equation

$$\mathbf{y}(t) = \mathbf{x}(1,t). \tag{2.118c}$$

To obtain a transfer function description of the system (2.118), we apply the bilateral Laplace transform to (2.118a). This gives the spatial ODE

$$\frac{\partial^2}{\partial z^2}\mathbf{x}(z,s) = \underbrace{\mathbf{\Lambda}^{-1}(s\mathbf{I} - \mathbf{\Sigma})}_{=\mathbf{H}(s)}\mathbf{x}(z,s). \tag{2.119}$$

The dependence of $\mathbf{H}(s)$ on the Laplace variable s will be omitted for the following calculations. The matrix $\mathbf{H}$ can be diagonalized by using the state transform $\mathbf{x}(z, s) =$

$\mathbf{P}\boldsymbol{\chi}(z,s)$, which yields

$$\frac{\partial^2}{\partial z^2}\boldsymbol{\chi}(z,s) = \tilde{\mathbf{H}}^2\boldsymbol{\chi}(z,s), \tag{2.120}$$

with the diagonal matrix $\tilde{\mathbf{H}} = \sqrt{\mathbf{P}^{-1}\mathbf{H}\mathbf{P}}$ and the corresponding boundary conditions

$$\boldsymbol{\chi}(0,s) = \mathbf{P}^{-1}\mathbf{u}(s), \qquad \frac{\partial \boldsymbol{\chi}}{\partial z}(1,s) = \tilde{\boldsymbol{\Gamma}}\boldsymbol{\chi}(1,s) \tag{2.121}$$

with $\tilde{\boldsymbol{\Gamma}} = \mathbf{P}^{-1}\boldsymbol{\Gamma}\mathbf{P}$. Applying the ansatz

$$\boldsymbol{\chi}(z,s) = \cosh\left(\tilde{\mathbf{H}}z\right)\begin{bmatrix} c_{11} \\ c_{21} \end{bmatrix} + \sinh\left(\tilde{\mathbf{H}}z\right)\begin{bmatrix} c_{12} \\ c_{22} \end{bmatrix} \tag{2.122}$$

to (2.120), (2.121) yields the solution

$$\mathbf{x}(z,s) = \mathbf{P}\left[\cosh\left(\tilde{\mathbf{H}}z\right) - \sinh\left(\tilde{\mathbf{H}}z\right)\boldsymbol{\Xi}\right]\mathbf{P}^{-1}\mathbf{u}(s), \tag{2.123}$$

with

$$\boldsymbol{\Xi} = \left[\tilde{\mathbf{H}}\cosh\left(\tilde{\mathbf{H}}\right) - \tilde{\boldsymbol{\Gamma}}\sinh\left(\tilde{\mathbf{H}}\right)\right]^{-1}\left[\tilde{\mathbf{H}}\sinh\left(\tilde{\mathbf{H}}\right) - \tilde{\boldsymbol{\Gamma}}\cosh\left(\tilde{\mathbf{H}}\right)\right]. \tag{2.124}$$

Finally, with (2.118c) the transfer matrix is given by

$$\mathbf{G}_u = \mathbf{P}\left[\cosh\left(\tilde{\mathbf{H}}\right) - \sinh\left(\tilde{\mathbf{H}}\right)\boldsymbol{\Xi}\right]\mathbf{P}^{-1}. \tag{2.125}$$

In the absence of stochastic disturbances, a simple pseudo-inversion-based law (2.80) with $\alpha = 1 \cdot 10^{-3}$ is chosen. Using the parameter values $\lambda_1 = \lambda_2 = 1$, $\sigma_{12} = 1/2$, $\sigma_{21} = -1$, $\Gamma_{11} = 0$, $\Gamma_{12} = 1/2$, $\Gamma_{21} = 1$, and $\Gamma_{22} = 1/10$, one obtains a numerical solution of the impulse response matrix $\mathbf{L}(t)$ using (2.112) as shown in Figure 2.6.

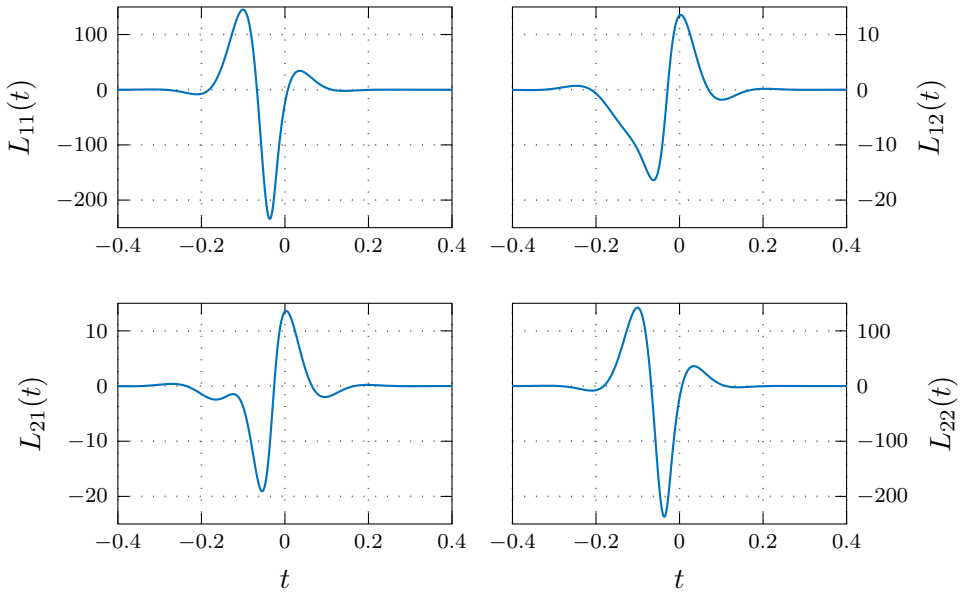


Figure 2.6: Entries of the (non-causal) learning kernel $\mathbf{L}(t)$ calculated as a FIR-approximation using the FFT.

Using the calculated learning kernel to track a desired output trajectory $\mathbf{y}^d(t) = [y_1^d(t), y_2^d(t)]^{\mathrm{T}}$, $t \in [0, 10]$ given by

$$y_1^d(t) = \Theta_5(t - 2.5), \quad y_2^d(t) = 3\frac{\mathrm{d}}{\mathrm{d}t}(\Theta_5(t) + \Theta_5(t - 5)) \tag{2.126}$$

with the smoothed step function

$$\Theta_T(t) = \begin{cases} 0 & t \leq 0 \\ 1 & t \geq T \\ \dfrac{\int_0^t \theta_T(\tau)\,\mathrm{d}\tau}{\int_0^T \theta_T(\tau)\,\mathrm{d}\tau} & t \in (0, T), \end{cases} \tag{2.127}$$

whereby

$$\theta_T(t) = \begin{cases} 0 & t \notin (0, T) \\ \exp\left[-\left((1 - t/T)t/T\right)^{-1.5}\right] & t \in (0, T), \end{cases} \tag{2.128}$$

the system (2.118) converges up to numeric precision of the solver after the first iteration with the resulting state profile $\mathbf{x}_1(z, t)$ shown in (2.7). Using higher values for α reduces the learning rate as expected.

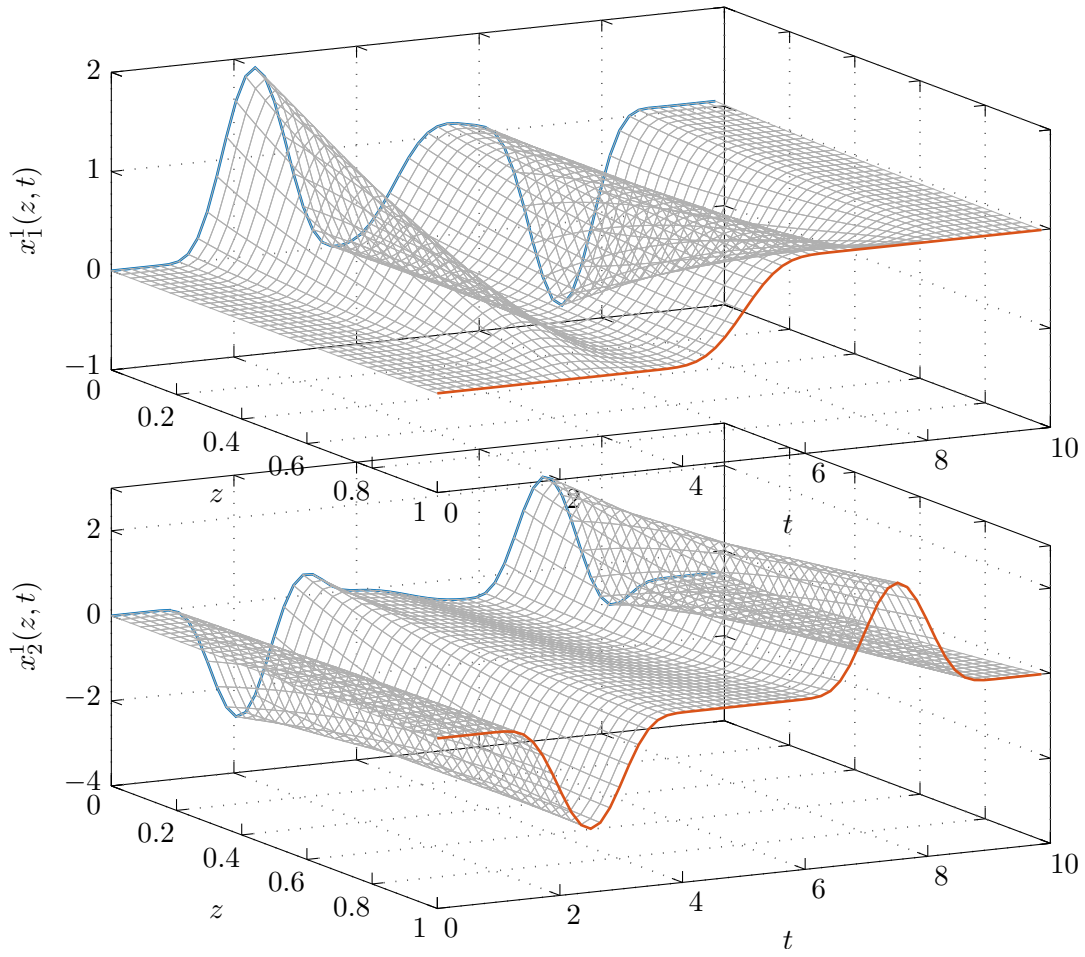


Figure 2.7: State $\mathbf{x}_1(z,t)$ of the PDE system (2.118) with the input $\mathbf{u}_1(t) = \mathbf{x}_1(0,t)$ (blue) that yields $\mathbf{y}_1(t) = \mathbf{x}_1(1,t) \approx \mathbf{y}^d(t)$ (red) after the first iteration.

2.4 Discrete-time systems on finite time horizons

Frequency-domain methods are a very valuable tool to gain insight and intuition, but they are limited to infinite time horizons by design. For the finite time horizon, one has to either analyze the input-output behavior using the (linear) integral operator of the convolution in time-domain or to revert to state-space methods. For discrete-time systems, however, there exist an alternative approach: since the signal spaces of the system input and output are reduced to finite dimensions, every linear mapping between these spaces can be represented as a matrix. This is the basis of the so-called *lifted system representation*, which is commonly used to analyze ILC schemes.

2.4.1 Lifted system respresentation

In this section, we restrict the general state-space description (2.52) to the SISO case without measurement noise, i.e.,

$$\dot{\mathbf{x}}_j(t) = \mathbf{A}\mathbf{x}_j(t) + \mathbf{b}u_j(t) \ , \quad \mathbf{x}_j(0) = \mathbf{x}_0 \tag{2.129a}$$

$$y_j(t) = \mathbf{c}^{\mathrm{T}}\mathbf{x}_j(t) + du_j(t), \tag{2.129b}$$

Note that the lifted-system representation is in principle open to time-varying and MIMO systems. Using a zero-order hold (ZOH) model with a sampling time T_a, we obtain the discrete time system

$$\mathbf{x}_j[k+1] = \mathbf{\Phi}\mathbf{x}_j[k] + \mathbf{\Gamma}u_j[k] \ , \quad \mathbf{x}_j[0] = \mathbf{x}_0 \tag{2.130a}$$

$$y_j[k] = \mathbf{c}^{\mathrm{T}}\mathbf{x}_j[k] + du_j[k] \tag{2.130b}$$

with time index $k = 0, 1, \ldots$ and the sampled state of the j-th iteration $\mathbf{x}_j[k] = \mathbf{x}_j(kT_a)$, the input $u_j[k] = u_j(kT_a)$, the output $y_j[k] = y_j(kT_a)$, and

$$\mathbf{\Phi} = \exp(\mathbf{A}T_a) \quad \text{und} \quad \mathbf{\Gamma} = \int_0^{T_a} \exp(\mathbf{A}\tau)\mathrm{d}\tau\mathbf{b} = \big(\exp(\mathbf{A}T_a) - \mathbf{I}\big)\mathbf{A}^{-1}\mathbf{b} \ . \tag{2.131}$$

The corresponding input-output representation in discrete time reads

$$y_j[0] = \mathbf{c}^{\mathrm{T}}\mathbf{x}_0 + du_j[0] \ , \tag{2.132a}$$

$$y_j[k] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^k\mathbf{x}_0 + \mathbf{c}^{\mathrm{T}}\sum_{m=0}^{k-1}\Big(\mathbf{\Phi}^{k-m-1}\mathbf{\Gamma}u_j[m]\Big) + du_j[k] \ , \quad k = 1, 2, \ldots \ . \tag{2.132b}$$

that can be rewritten using

$$g[k] = \begin{cases} d & \text{for} \quad k = 0 \\ \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{k-1}\mathbf{\Gamma} & \text{for} \quad k = 1, 2, \ldots \end{cases} , \tag{2.133}$$

to obtain the discrete-time input-output representation

$$y_j[k] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^k\mathbf{x}_0 + \sum_{m=0}^{k} g[m]u_j[k-m] \ . \tag{2.134}$$

This equation is the analogous result to (2.53) using the discrete convolution with the impulse response $g[k]$, except that on a finite horizon we cannot assume that the system is relaxed initially and thus there is a remaining contribution of the initial state $\mathbf{x}_0$. By introducing the shift operator $\delta : \delta(z_j[k]) = z_j[k+1]$ and assuming $\mathbf{x}_0 = \mathbf{0}$, one can again define the transfer operator

$$\begin{aligned} G(\delta) = \frac{y_j[k]}{u_j[k]} &= d + \mathbf{c}^{\mathrm{T}}(\delta\mathbf{I} - \mathbf{\Phi})^{-1}\mathbf{\Gamma} = d + \delta^{-1}\mathbf{c}^{\mathrm{T}}\Big(\mathbf{I} - \delta^{-1}\mathbf{\Phi}\Big)^{-1}\mathbf{\Gamma} \\ &= d + \delta^{-1}\mathbf{c}^{\mathrm{T}}\Big(\mathbf{I} + \delta^{-1}\mathbf{\Phi}^1 + \delta^{-2}\mathbf{\Phi}^2 + \ldots\Big)\mathbf{\Gamma} = d + \delta^{-1}\mathbf{c}^{\mathrm{T}}\sum_{k=0}^{\infty}(\delta^{-k}\mathbf{\Phi}^k)\mathbf{\Gamma} \\ &= d + \sum_{k=1}^{\infty}\mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{k-1}\mathbf{\Gamma}\delta^{-k} = \sum_{k=0}^{\infty} g[k]\delta^{-k}. \end{aligned} \tag{2.135}$$

Bemerkung 2.1. The infinite-horizon methods introduced in the previous section for continuous-time systems can be transferred to the discrete-time case directly using (2.135).

Bemerkung 2.2. The relative degree of a discrete-time systems is defined as follows:

Definition 2.10. A system (2.130) with $d = 0$ is of relative degree r if

(A) $\mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{k}\mathbf{\Gamma} = 0\ , \quad k = 0, 1, \dots r-2$

(B) $\mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{r-1}\mathbf{\Gamma} \neq 0\ .$

The relative degree of a discrete-time systems thus merely corresponds to the time index k of $y_j[k]$ at which the input $u_j[0]$ appears for the first time, i.e.,

$$y_j[0] = \mathbf{c}^{\mathrm{T}}\mathbf{x}_0 \tag{2.136a}$$

$$y_j[1] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}\mathbf{x}_0 + \underbrace{\mathbf{c}^{\mathrm{T}}\mathbf{\Gamma}}_{=0} u_j[0] \tag{2.136b}$$

$$y_j[2] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^2\mathbf{x}_0 + \underbrace{\mathbf{c}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Gamma}}_{=0} u_j[0] + \underbrace{\mathbf{c}^{\mathrm{T}}\mathbf{\Gamma}}_{=0} u_j[1] \tag{2.136c}$$

$$\vdots$$

$$y_j[r-1] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{r-1}\mathbf{x}_0 + \underbrace{\mathbf{c}^{\mathrm{T}} \sum_{m=0}^{r-2} \mathbf{\Phi}^{r-m-1}\mathbf{\Gamma}\, u_j[m]}_{=0} \tag{2.136d}$$

$$y_j[r] = \mathbf{c}^{\mathrm{T}}\mathbf{\Phi}^{r}\mathbf{x}_0 + \underbrace{\mathbf{c}^{\mathrm{T}} \sum_{m=0}^{r-1} \mathbf{\Phi}^{r-m-1}\mathbf{\Gamma}\, u_j[m]}_{\neq 0} \tag{2.136e}$$

The case of a discrete-time system (2.130) obtained by (ZOH-) sampling of a continous-time system (2.129) begs the questions how the relative degrees of these two systems compare. Using a series expansion of the exponential matrix $\exp(\mathbf{A}\tau)$ yields

$$\mathbf{c}^{\mathrm{T}}\mathbf{\Gamma} = \mathbf{c}^{\mathrm{T}} \int_0^{T_a} \sum_{m=0}^{\infty} \frac{(\mathbf{A}\tau)^m}{m!}\mathbf{b}\,\mathrm{d}\tau \tag{2.137a}$$

$$= \mathbf{c}^{\mathrm{T}} \int_0^{T_a} \mathbf{b} + \mathbf{A}\mathbf{b}\tau + \ldots + \frac{1}{(n-1)!}\mathbf{A}^{n-1}\mathbf{b}\tau^{n-1} + \mathcal{O}(T_a^n)\,\mathrm{d}\tau \tag{2.137b}$$

$$= \mathbf{c}^{\mathrm{T}}\mathbf{b}T_a + \frac{1}{2!}\mathbf{c}^{\mathrm{T}}\mathbf{A}\mathbf{b}T_a^2 + \ldots + \frac{1}{n!}\mathbf{c}^{\mathrm{T}}\mathbf{A}^{n-1}\mathbf{b}T_a^n + \mathcal{O}(T_a^{n+1}) \tag{2.137c}$$

which is always unequal to zero. A discrete-time system derived from a continous-time system via ZOH is thus always of relative degree $r = 1$ for $d = 0$ (and $r = 0$ for $d \neq 0$).

Before trying to obtain a matrix representation of the system's input-output behavior (2.134), it makes sense to carefully define suitable input and output sequences that shall be related by this mapping. Specifically, the initial input $u_j[0]$ will only start to act on the output $y_j[m]$ with some temporal delay $m = r + w$, where r is the relative degree of the system (2.130) and w is an additional delay due to sampling, conversion and processing of data that is unavoidable in real-world applications. For a finite time horizon $t \in [0, t_f]$ with $t_f = NT_a$ we will thus consider the input and output sequences

$$u_j[k] \ , \quad k = 0, 1, \dots, N-1 \tag{2.138a}$$
$$y_j[k] \ , \quad k = m, m+1, \dots, N+m-1 \tag{2.138b}$$

as well as a desired output sequence

$$y_d[k], \quad k = m, m+1, \dots, N+m-1. \tag{2.139a}$$

Note that this is essentially shifting the time axis of the output sequence relative to the input sequence to compensate for the total system delay (cf. the time delay analysis in Section 2.3.4). Rewriting the corresponding input and output sequences in vector notation, i.e.,

$$\mathbf{u}_j^{\mathrm{T}} = \begin{bmatrix} u_j[0] & u_j[1] & \dots & u_j[N-1] \end{bmatrix} \in \mathbb{R}^N \tag{2.140a}$$
$$\mathbf{y}_j^{\mathrm{T}} = \begin{bmatrix} y_j[m] & y_j[m+1] & \dots & y_j[m+N-1] \end{bmatrix} \in \mathbb{R}^N \tag{2.140b}$$
$$\mathbf{y}_d^{\mathrm{T}} = \begin{bmatrix} y_d[m] & y_d[m+1] & \dots & y_d[m+N-1] \end{bmatrix} \in \mathbb{R}^N \tag{2.140c}$$
$$\mathbf{y}_0^{\mathrm{T}} = \begin{bmatrix} y_0[m] & y_0[m+1] & \dots & y_0[m+N-1] \end{bmatrix} \in \mathbb{R}^N \tag{2.140d}$$
$$\mathbf{e}_j^{\mathrm{T}} = \mathbf{y}_d^{\mathrm{T}} - \mathbf{y}_j^{\mathrm{T}} = \begin{bmatrix} e_j[m] & e_j[m+1] & \dots & e_j[m+N-1] \end{bmatrix} \in \mathbb{R}^N \ , \tag{2.140e}$$

and using (2.134) yields the so-called *lifted system representation*

$$\mathbf{y}_j = \mathbf{y}_0 + \mathbf{G}\mathbf{u}_j \tag{2.141}$$

with the matrix

$$\mathbf{G} = \begin{bmatrix} g[m] & 0 & \cdots & 0 \\ g[m+1] & g[m] & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ g[m+N-1] & g[m+N-2] & \cdots & g[m] \end{bmatrix} \in \mathbb{R}^{N \times N} \ . \tag{2.142}$$

For LTI systems this is a so-called *Toeplitz* matrix where the entries $\mathbf{G}_{ij}$ only depend on the difference $i - j$. An equivalent representation for time-varying systems can be found in [2.12]. Note that since $\mathbf{G}$ is a lower triangular matrix due to causality of the system (2.129) and we assured by definition that $g[m] \neq 0$, $\mathbf{G}$ is of full rank and thus always invertible. Analogous to (2.4), a linear ILC law for the lifted system representation is given by

$$\mathbf{u}_{j+1} = \mathbf{Q}\Big(\mathbf{u}_j + \mathbf{L}\mathbf{e}_j\Big) \tag{2.143}$$

with the $\mathbf{Q}$-filtering matrix

$$\mathbf{Q} = \begin{bmatrix} q[0] & q[-1] & \cdots & q[-(N-1)] \\ q[1] & q[0] & \cdots & q[-(N-2)] \\ \vdots & \vdots & \ddots & \vdots \\ q[N-1] & q[N-2] & \cdots & q[0] \end{bmatrix} \in \mathbb{R}^{N\times N} \tag{2.144}$$

and the learning gain matrix

$$\mathbf{L} = \begin{bmatrix} l[0] & l[-1] & \cdots & l[-(N-1)] \\ l[1] & l[0] & \cdots & l[-(N-2)] \\ \vdots & \vdots & \ddots & \vdots \\ l[N-1] & l[N-2] & \cdots & l[0] \end{bmatrix} \in \mathbb{R}^{N\times N} \ . \tag{2.145}$$

Aufgabe 2.6. Reformulate a PD-type learning law analogous to (2.74) with a Gaussian $\mathbf{Q}$-filter (2.103) in the lifted framework. What options do you have? What about the boundaries of the time horizon?

The system description (2.141) and the learning law (2.143) are very similar to the infinite-horizon case (2.57) and (2.58) except for the constant term $\mathbf{y}_0$. Defining $\tilde{\mathbf{y}}_d = \mathbf{y}_d - \mathbf{y}_0$, one obtains the input iteration

$$\mathbf{u}_{j+1} = \mathbf{\Psi}\mathbf{u}_j + \mathbf{\Lambda}\tilde{\mathbf{y}}_d \tag{2.146}$$

with $\mathbf{\Psi} = \mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})$ and $\mathbf{\Lambda} = \mathbf{Q}\mathbf{L}$ and the corresponding iteration of the output error $\mathbf{e}_j = \mathbf{y}_d - \mathbf{y}_j$ as

$$\mathbf{e}_{j+1} = \mathbf{G}\mathbf{\Psi}\mathbf{G}^{-1}\mathbf{e}_j + (\mathbf{I} - \mathbf{G}\mathbf{Q}\mathbf{G}^{-1})\mathbf{y}_d \ . \tag{2.147}$$

Both iterations are algebraically identical to the infinite-horizon case. We can thus directly transfer stability and convergence results to the lifted system representation, which are restated in the following for completeness.

Satz 2.13 (Asymptotic stability of the ILC law)**.** *The input iteration* (2.146) *of the ILC law* (2.143) *is asymptotically stable if*

$$\rho\Big(\mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})\Big) < 1 \tag{2.148}$$

and $\mathbf{u}_j$ *converges to* $\mathbf{u}_\infty$.

Satz 2.14 (Asymptotic stability of the output iteration)**.** *The output iteration* (2.147) *of the ILC law* (2.143) *is asymptotically stable iff the input iteration is stable, i.e.,*

$$\rho\Big(\mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})\Big) < 1 \tag{2.149}$$

and $\mathbf{e}_j$ *then converges to the asymptotic tracking error*

$$\mathbf{e}_\infty = \Big(\mathbf{I} - \mathbf{G}(\mathbf{I} - \boldsymbol{\Psi})^{-1}\boldsymbol{\Lambda}\Big)\mathbf{y}_d = \Big(\mathbf{I} - \mathbf{G}\boldsymbol{\Phi}\mathbf{G}^{-1}\Big)^{-1}\Big(\mathbf{I} - \mathbf{G}\mathbf{Q}\mathbf{G}^{-1}\Big)\mathbf{y}_d \ . \tag{2.150}$$

Satz 2.15 (Monotonic convergence of the input iteration)**.** *The input iteration* (2.146) *of the ILC law* (2.143) *converges monotonically to* $\mathbf{u}_\infty$*, i.e., it holds that*

$$\|\mathbf{u}_{j+1} - \mathbf{u}_\infty\| \leq \alpha\|\mathbf{u}_j - \mathbf{u}_\infty\| \tag{2.151}$$

for $0 \leq \alpha < 1$ *if*

$$\|\boldsymbol{\Psi}\| = \bar{\sigma}\Big(\mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})\Big) = \alpha < 1 \ . \tag{2.152}$$

Satz 2.16 (Monotonic convergence of the output iteration)**.** *The output iteration* (2.64) *of the ILC law* (2.58) *converges monotonically to* $\mathbf{e}_\infty$*, i.e., it holds that*

$$\|\mathbf{e}_{j+1} - \mathbf{e}_\infty\| \leq \beta\|\mathbf{e}_j - \mathbf{e}_\infty\| \tag{2.153}$$

for $0 \leq \alpha < 1$ *if*

$$\|\mathbf{G}\boldsymbol{\Psi}\mathbf{G}^{-1}\| = \bar{\sigma}\Big(\mathbf{G}\mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G})\mathbf{G}^{-1}\Big) = \beta < 1 \ . \tag{2.154}$$

Bemerkung 2.3. These results are structurally very similar to the infinite time horizon case with a number of significant differences: By applying the Laplace transform on infinite time horizons, one is considering stability for every $\omega \in \mathbb{R}$ independently. While stability can be transferred to the finite time horizon as shown in the previous section, frequency-domain criteria are rather conservative. Conversely, stability criteria using the lifted system representation are sharp by accurately accounting for boundary effects. The dimension of $\mathbf{Q}$ and $\mathbf{L}$ is determined by the length of the sampled time horizon N, which can be problematic for long time horizons.

2.4.2 ILC as an online optimization strategy

Using measurements of a system's behavior to iteratively improve its performance with respect to some cost function can also be seen as an optimization problem that is solved online. Consider the problem

$$\min_{\mathbf{u}} \ \frac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{P}\mathbf{e} + \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{W}\mathbf{u} + \mathbf{u}^{\mathrm{T}}\mathbf{F}\mathbf{e} \tag{2.155a}$$

$$\text{subject to } \mathbf{e} = \mathbf{y}_d - \mathbf{G}\mathbf{u} \ , \tag{2.155b}$$

with the symmetric, positive (semi-) definite weighting matrices $\mathbf{P}$ und $\mathbf{W}$. One can assume that $\mathbf{F}\mathbf{G}$ is a skew-symmetric matrix without loss of generality since any symmetric component could always be absorbed into $\mathbf{W}$. This becomes apparent when plugging

(2.155b) into (2.155a), which yields the equivalent problem

$$\min_{\mathbf{u}} \; J(\mathbf{u}) = \frac{1}{2}\mathbf{u}^{\mathrm{T}}\bar{\mathbf{A}}\mathbf{u} + \mathbf{u}^{\mathrm{T}}\bar{\mathbf{b}} + \bar{\mathbf{c}} \tag{2.156}$$

with

$$\bar{\mathbf{A}} = \mathbf{G}^{\mathrm{T}}\mathbf{P}\mathbf{G} + \mathbf{W} \tag{2.157a}$$

$$\bar{\mathbf{b}} = \Big(\mathbf{F} - \mathbf{G}^{\mathrm{T}}\mathbf{P}\Big)\mathbf{y}_d \tag{2.157b}$$

$$\bar{\mathbf{c}} = \frac{1}{2}\mathbf{y}_d^{\mathrm{T}}\mathbf{P}\mathbf{y}_d. \tag{2.157c}$$

The quadratic expression $\mathbf{u}^{\mathrm{T}}\mathbf{F}\mathbf{G}\mathbf{u}$ vanishes due to $\mathbf{F}\mathbf{G}$ being skew-symmetric. Since we further assumed that $\mathbf{P}$ and $\mathbf{W}$ are symmetric and positive (semi-) definite matrices, the same holds true for $\bar{\mathbf{A}}$. In case $\bar{\mathbf{A}}$ is indeed positive definite, the cost function $J(\mathbf{u})$ is strictly convex and thus has a unique global minimum at $\mathbf{u}_\infty = -\bar{\mathbf{A}}^{-1}\bar{\mathbf{b}}$ which is determined by the necessary and sufficient first-order condition $\nabla J(\mathbf{u}_\infty) = \bar{\mathbf{A}}\mathbf{u}_\infty + \bar{\mathbf{b}} = \mathbf{0}$.

Alternatively, such an optimization problem can be solved iteratively using a gradient-descent method with constant step width α, e.g.,

$$\mathbf{u}_{j+1} = \mathbf{u}_j - \alpha\nabla J(\mathbf{u}_j) = (\mathbf{I} - \alpha\bar{\mathbf{A}})\mathbf{u}_j - \alpha\bar{\mathbf{b}}, \tag{2.158}$$

where $0 < \alpha < 2/\|\bar{\mathbf{A}}\|$ ensures (monotone) convergence, cf. Theorem 2.1.13 in [2.13]. Plugging (2.157b) into (2.158) and suppressing $\mathbf{y}_d$ in favor of $\mathbf{u}_j$ and $\mathbf{e}_j$ using (2.155b) yields an ILC-like update law

$$\mathbf{u}_{j+1} = \mathbf{Q}\Big(\mathbf{u}_j + \mathbf{L}\mathbf{e}_j\Big) \tag{2.159}$$

with

$$\mathbf{Q} = \mathbf{L}_u = \mathbf{I} - \alpha(\mathbf{W} + \mathbf{F}\mathbf{G}) \tag{2.160a}$$

$$\mathbf{Q}\mathbf{L} = \mathbf{L}_e = \alpha\Big(\mathbf{G}^{\mathrm{T}}\mathbf{P} - \mathbf{F}\Big). \tag{2.160b}$$

2.4.3 Norm-optimal ILC strategies

Along the same line of thought, norm-optimal ILC methods [2.12, 2.14] avoid the explicit design of a Q-filter and learning filter by solving the optimization problem

$$\begin{aligned} \min_{\mathbf{u}_{j+1}} \quad & J(\mathbf{u}_{j+1}) = \underbrace{\frac{1}{2}\mathbf{e}_{j+1}^{\mathrm{T}}\mathbf{V}\mathbf{e}_{j+1}}_{J_1(\mathbf{u}_{j+1})} + \underbrace{\frac{1}{2}\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{S}\mathbf{u}_{j+1} + \frac{1}{2}(\mathbf{u}_{j+1} - \mathbf{u}_j)^{\mathrm{T}}\mathbf{R}(\mathbf{u}_{j+1} - \mathbf{u}_j)}_{J_2(\mathbf{u}_{j+1})} \\ \text{u.B.v.} \quad & \mathbf{e}_{j+1} = \mathbf{y}_d - \mathbf{y}_{j+1} = \mathbf{e}_j + \mathbf{G}\mathbf{u}_j - \mathbf{G}\mathbf{u}_{j+1} \end{aligned} \tag{2.161}$$

for *every* iteration. Note that this is in contrast to the previous section, where ILC was rewritten as an *iterative solution* of a *single* optimization problem. We assume that $\mathbf{V}$ and $\mathbf{S}$ are symmetric, positive semidefinite matrices and $\mathbf{R}$, $\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}$, $\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S}$ are symmetric, positive definit matrices.

Plugging the constraint into the cost function of (2.161) yields

$$\begin{aligned} J_1(\mathbf{u}_{j+1}) &= \frac{1}{2}(\mathbf{e}_j + \mathbf{G}\mathbf{u}_j - \mathbf{G}\mathbf{u}_{j+1})^{\mathrm{T}}\mathbf{V}(\mathbf{e}_j + \mathbf{G}\mathbf{u}_j - \mathbf{G}\mathbf{u}_{j+1}) \\ &= \frac{1}{2}\Big(\mathbf{e}_j^{\mathrm{T}}\mathbf{V}\mathbf{e}_j + \mathbf{u}_j^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{e}_j - 2\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{e}_j \\ &\quad + \mathbf{e}_j^{\mathrm{T}}\mathbf{V}\mathbf{G}\mathbf{u}_j + \mathbf{u}_j^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G}\mathbf{u}_j - 2\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G}\mathbf{u}_j + \mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G}\mathbf{u}_{j+1}\Big) \end{aligned} \tag{2.162}$$

and

$$\begin{aligned} J_2(\mathbf{u}_{j+1}) &= \frac{1}{2}\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{S}\mathbf{u}_{j+1} + \frac{1}{2}(\mathbf{u}_{j+1}^{\mathrm{T}} - \mathbf{u}_j^{\mathrm{T}})\mathbf{R}(\mathbf{u}_{j+1} - \mathbf{u}_j) \\ &= \frac{1}{2}\Big(\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{S}\mathbf{u}_{j+1} + \mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{j+1} - 2\mathbf{u}_{j+1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_j + \mathbf{u}_j^{\mathrm{T}}\mathbf{R}\mathbf{u}_j\Big)\ . \end{aligned} \tag{2.163}$$

Using the first-order optimality condition

$$\left(\frac{\partial}{\partial \mathbf{u}_{j+1}} J\right)(\mathbf{u}_{j+1}) = \mathbf{0} \tag{2.164}$$

results in input update

$$\Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S} + \mathbf{R}\Big)\mathbf{u}_{j+1} = \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}\Big)\mathbf{u}_j + \mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{e}_j \tag{2.165}$$

that is equivalent to an ILC law (2.143) with $\mathbf{Q}$-filtering and learning matrices

$$\mathbf{Q} = \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S} + \mathbf{R}\Big)^{-1}\Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}\Big) \tag{2.166a}$$

$$\mathbf{L} = \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}\Big)^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{V}. \tag{2.166b}$$

By considering

$$\begin{aligned} \mathbf{Q}(\mathbf{I} - \mathbf{L}\mathbf{G}) &= \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S} + \mathbf{R}\Big)^{-1}\Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}\Big)\Big(\mathbf{I} - \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{R}\Big)^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G}\Big) \\ &= \Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S} + \mathbf{R}\Big)^{-1}\mathbf{R}\ . \end{aligned} \tag{2.167}$$

one can show that the derived norm-optimal ILC scheme is asymptotically stable, i.e.,

$$\rho\Big(\Big(\mathbf{G}^{\mathrm{T}}\mathbf{V}\mathbf{G} + \mathbf{S} + \mathbf{R}\Big)^{-1}\mathbf{R}\Big) < 1. \tag{2.168}$$

The values of $\mathbf{V}$, $\mathbf{S}$ and $\mathbf{R}$ are tuning parameters that are often simplified by using diagonal matrices, i.e., $\mathbf{V} = v\mathbf{I} > 0$, $\mathbf{S} = s\mathbf{I} > 0$ und $\mathbf{R} = r\mathbf{I} > 0$. It follows that:

- Large values of v increases the weighting of the output error $\mathbf{e}_j$, which increases the convergence rate and reduces $\mathbf{e}_\infty$. The learning gain is becoming more aggressive and the action of the $\mathbf{Q}$-filter is reduced. The limit $v \to \infty$ yields $\mathbf{L} \to \mathbf{G}^{-1}$ and $\mathbf{Q} \to \mathbf{I}$.
- Large values of s penalize the control input $\mathbf{u}_j$, which increases the asymptotic output error $\mathbf{e}_\infty$. Note that s only affects $\mathbf{Q}$ where $s \to 0$ yields $\mathbf{Q} \to \mathbf{I}$.
- Large values of r penalize changes of the control input $\mathbf{u}_{j+1} - \mathbf{u}_j$, which decreases the convergence rate. For $r \to 0$ it follows that $\mathbf{L} \to \mathbf{G}^{-1}$.

2.5 Literatur

[2.1] Z. Bien and J.-X. Xu, *Iterative learning control - analysis, design, integration and applications*. London: Springer, 1998.

[2.2] D. Owens and J. Hätönen, "Iterative learning control - an optimization paradigm," *Annual Reviews in Control*, vol. 29, no. 1, pp. 57–70, 2005.

[2.3] D. A. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *Control Systems, IEEE*, vol. 26, no. 3, pp. 96–114, 2006.

[2.4] H.-S. Ahn, Y.-Q. Chen, and K. Moore, "Iterative learning control: Brief survey and categorization," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 6, pp. 1099–1121, 2007.

[2.5] H.-S. Ahn, K. L. Moore, and Y. Chen, *Iterative learning control - robustness and monotonic convergence for interval system*. London: Springer, 2007.

[2.6] Y. Wang, F. Gao, and F. J. Doyle III, "Survey on iterative learning control, repetitive control, and run-to-run control," *Journal of Process Control*, vol. 19, no. 10, pp. 1589–1600, 2009.

[2.7] J. M. Ortega, "Stability of difference equations and convergence of iterative processes," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 268–282, 1973.

[2.8] D. A. Bristow, "Iterative learning control for precision motion of microscale and nanoscale tracking systems," Ph.D. dissertation, University of Illinois at Urban-Champaign, 2007.

[2.9] J. Ghosh and B. Paden, "Iterative learning control for nonlinear nonminimum phase plants with input disturbances," in *Proceedings of the American Control Conference*, Jun. 1999, pp. 2584–2589. (visited on 05/24/2016).

[2.10] J. Ghosh and B. Paden, "A pseudoinverse-based iterative learning control," *IEEE Transactions on Automatic Control*, vol. 47, no. 5, pp. 831–837, 2002.

[2.11] J. Wallén, S. Gunnarsson, and M. Norrlöf, "Analysis of boundary effects in iterative learning control," *International Journal of Control*, vol. 86, no. 3, pp. 410–415, 2013.

[2.12] J. H. Lee, K. S. Lee, and W. C. Kim, "Model-based iterative learning control with a quadratic criterion for time-varying linear systems," *Automatica*, vol. 36, no. 5, pp. 641–657, 2000.

[2.13] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Dordrecht: Kluwer Academic Publishers, 2004.

[2.14] K. Barton and A. Alleyne, "A norm optimal approach to time-varying ilc with application to a multi-axis robotic testbed," *Control Systems Technology, IEEE Transactions on*, vol. 19, no. 1, pp. 166–180, Jan. 2011.

3 Stochastic optimal control and reinforcement learning

There are different types of machine learning including supervised learning, self-supervised learning and unsupervised learning. *Reinforcement learning refers to a learner or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions.* This is based on evaluative information from the environment and could be called action-based learning. Reinforcement learning implies a cause and effect relationship between actions and reward. It implies goal directed behavior at least insofar as the agent has an understanding of reward versus lack of reward or punishment. *Optimal Control* (OC) and *Reinforcement Learning* (RL) both deal with *sequential decision-making in deterministic and stochastic environments*, but they approach the problem from different perspectives and have distinct characteristics:

Foundation and historical roots:

- OC: Stems from the field of control theory. It traditionally focuses on the mathematical modeling and understanding of systems, often with a well-defined model of the environment in mind.
- RL: Emerged from the realm of artificial intelligence and, to some extent, psychology, inspired by behavioral learning theories. RL often operates in scenarios where the model of the environment is unknown.

Knowledge about the environment:

- OC: Typically assumes a known model of the environment. This model describes state transitions and rewards (or costs) as a function of states and actions.
- RL: Can operate with or without an explicit model of the environment. Model-free RL methods, like Q-learning or policy gradient methods, directly learn a policy or value function without needing to know the dynamics or reward structure.

Solution techniques:

- OC: Methods in OC, such as dynamic programming (Value Iteration, Policy Iteration), rely heavily on the known model to find the optimal policy.
- RL: Employs a wider range of techniques, including but not limited to dynamic programming. Many algorithms, especially model-free ones, rely on interaction with the environment to learn. RL also incorporates deep learning (Deep RL) to handle large-scale problems with high-dimensional input spaces.

Application:

- OC: Traditionally applied to problems with smaller state and action spaces due to the computational intensity of exact methods. However, approximations are possible for larger problems.
- RL: Given its roots in AI, RL has been widely applied to large-scale and high-dimensional problems, especially with the advent of deep learning. Examples include playing Atari games, Go, and various robotics tasks.

Exploration vs. exploitation:

- OC: When formulated classically, the OC problem typically assumes full knowledge of the system, and hence the concepts of exploration and exploitation are not central.
- RL: The trade-off between exploration (trying new actions to discover their effects) and exploitation (choosing known good actions) is a central theme in RL, especially in model-free settings.

Goal:

- OC: The primary goal is to find the optimal policy (or control law) given the system dynamics and reward function.
- RL: While the end goal is also to find an optimal policy, RL places a significant emphasis on learning from interaction. This can involve learning about the environment's dynamics, the reward structure, or directly about the optimal policy or value function.

3.1 Background material on Machine and Deep Learning

Machine Learning (ML) and Deep Learning (DL) are a subset of artificial intelligence (AI) that enables computers to learn from data and make decisions with minimal human intervention. ML and DL algorithms are classified into supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses labeled data to train models, unsupervised learning identifies structures in unlabeled data, and reinforcement learning involves agents optimizing cumulative rewards in an environment. For further reading, we refer to:

- Hastie, T., Tibshirani, R., & Friedman, J, The Elements of Statistical Learning, Springer, 2009.
- Bishop, C. M., & Bishop, H., Deep Learning: Foundation and Concepts, Springer, 2024.
- Prince, S. J. D., Understanding Deep Learning. Cambridge University Press, 2023, https://udlbook.github.io/udlbook/.
- Zhang, A., et. al, Dive into Deep Learning, Cambridge University Press, 2023, https://d2l.ai/.

3.2 Theoretical foundation

Since we are in a stochastic setting, it is important to understand the *basics of probability theory*. We use sans-serif letters such as $\mathsf{x}, \mathbf{x}$, and $\mathbf{X}$ to represent *random variables* (scalars, vectors and matrices) and serif letters such as $x, \mathbf{x}$, and $\mathbf{X}$ to represent the corresponding *deterministic variables* or events. For an introduction to probability theory, please refer to the brief introduction given in the Appendix **??** and for a consistent summary to [3.1]. A comprehensive overview can be found in [3.2].

3.2.1 Stochastic dynamical systems

We consider stochastic, nonlinear dynamical systems with discrete time evolution of the form

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k), \quad k = 0, \dots, N-1 \ , \tag{3.1}$$

and initial condition $\mathbf{x}_0$ drawn from the initial condition distribution $\mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0)$. Here,

- k denotes the discrete time or epoch index,
- $\mathbf{x}_k \in \mathcal{X} \subset \mathbb{R}^n$ is the stochastic state vector and
- $\mathbf{u}_k \in \mathcal{U}(\mathbf{x}_k) \subset \mathcal{U} \subset \mathbb{R}^m$ is the stochastic input vector. The input vector $\mathbf{u}_k$ is restricted to a nonempty subset $\mathcal{U}(\mathbf{x}_k) \subset \mathcal{U}$ that depends on $\mathbf{x}_k$.
- $\mathbf{w}_k \in \mathcal{W}$ is the disturbance vector.
- The horizon length is given by N.
- The mapping $\mathbf{f}_d : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X}$ describes how the system state $\mathbf{x}$ evolves over time.
- The sequence $\mathbf{W}_{[0:N-1]} = (\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{N-1})$ is assumed to be uncorrelated.

Let us use $\mathsf{p}(\mathbf{i}_k)$ to denote the probability distribution, with observation $\mathbf{i}_k$, of the stochastic vector $\mathbf{i}_k \in \mathcal{I}$ at time index k, i.e., $\mathbf{i}_k \sim \mathsf{p}(\mathbf{i}_k)$. A probability distribution is a description of how likely a stochastic variable is to take on each of its possible states. The method of describing this distribution varies based on the nature of the variable, being either discrete or continuous. Using this concept, it is actually possible to write the dynamics of the distribution with the *law of total probability*, cf. (**??**), as

$$\begin{aligned} \mathsf{p}(\mathbf{x}_{k+1}) &= \mathbb{E}_{\mathbf{x}_k, \mathbf{u}_k \sim \mathsf{p}(\mathbf{x}_k, \mathbf{u}_k)} \{ \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \} \\ &= \int_{\mathcal{X}} \int_{\mathcal{U}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \mathsf{p}(\mathbf{x}_k, \mathbf{u}_k) \mathrm{d}\mathbf{u}_k \mathrm{d}\mathbf{x}_k \ , \end{aligned} \tag{3.2}$$

where $\mathsf{p}_{\mathsf{x}} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$ encodes the stochastic dynamics as a conditional distribution of the next state $\mathbf{x}_{k+1}$ as a function of the current state $\mathbf{x}_k$ when applying the control input $\mathbf{u}_k$ [3.3, p. 13]. Thus, (3.1) can be alternatively written as

$$\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, 1, \dots, N-1 \ . \tag{3.3}$$

The fact that at this point two expressions (3.1) and (3.3) are given for the system dynamics is due to the fact that, depending on the problem, one of the two representations is to be preferred in order to obtain mathematically consistent and clear expressions. In the control engineering context, (3.1) is preferred over (3.3), whereas (3.3) is widely used in the reinforcement learning context. We will utilize both notations. Note that by construction, the system (3.1) satisfies the Markov property, see [3.4], which states that $\mathbf{x}_{k+1}$ depends only on quantities of the previous time step k, i.e.

$$\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) = \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_k, \mathbf{u}_k) \ , \quad k = 0, \dots, N-1. \tag{3.4}$$

Bemerkung 3.1. In the reinforcement learning literature, the control input $\mathbf{u}_k$ is referred to as action $\mathbf{a}_k$ and the state $\mathbf{x}_k$ is denoted by $\mathbf{s}_k$.

3.2.2 Rollouts, rewards and return

Let us introduce the *state sequence*

$$\mathbf{X}_{[0:N]} = (\mathbf{x}_0, \dots, \mathbf{x}_N) \tag{3.5}$$

and *input sequence*

$$\mathbf{U}_{[0:N-1]} = (\mathbf{u}_0, \dots, \mathbf{u}_{N-1}) \ , \tag{3.6}$$

which describes the time evolution of the state and input variables. We denote

$$\boldsymbol{\tau}_{[0,N]} = (\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1 \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N) \tag{3.7}$$

as the N-step *rollout*[1]. In optimal control, we are typically interested in minimizing a total cost. In the reinforcement learning literature, the reward is used instead of the cost. Consequently, the accumulative costs (total cost) are minimized and accumulative rewards (return) are maximized. Throughout this work we will adopt the logic and notation commonly found in the reinforcement learning literature. Note that the rewards can be positive and negative.

Definition 3.1. (Discounted return, see [3.5]) The *future discounted return* of a rollout $\boldsymbol{\tau}_{[k:N]}$ is

$$\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) = \sum_{i=k}^{N} \gamma^{i-k} \mathsf{r}_i = \gamma^{N-k} \mathsf{r}_N + \sum_{i=k}^{N-1} \gamma^{i-k} \mathsf{r}_i \tag{3.8}$$

with discount factor $\gamma \in [0, 1]$. It discounts and accumulates the *random immediate reward* $\mathsf{r}_k : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ as well as the *random terminal reward* $\mathsf{r}_N : \mathcal{X} \to \mathbb{R}$.

[1] Rollouts are also frequently called trajectories.

The smaller γ is chosen, the lower future rewards in r_k will be weighted, thereby giving more significance to immediate rewards. Conversely, rewards received for high occupancies in the future are weighted more heavily. Thus, this parameter essentially represents the distance of foresight into the future. Note that if R_k is the value received at time index k, then

$$\begin{aligned}\mathsf{R}_k &= \mathsf{r}_k + \gamma \mathsf{r}_{k+1} + \gamma^2 \mathsf{r}_{k+2} + \ldots + \gamma^{N-k-1} \mathsf{r}_{N-1} + \gamma^{N-k} \mathsf{r}_N \\ &= \mathsf{r}_k + \gamma \Big(\mathsf{r}_{k+1} + \gamma \mathsf{r}_{k+2} + \gamma^2 \mathsf{r}_{k+3} + \ldots + \gamma^{N-k-2} \mathsf{r}_{N-1} + \gamma^{N-k-1} \mathsf{r}_N \Big) \\ &= \mathsf{r}_k + \gamma \mathsf{R}_{k+1} \ . \end{aligned} \tag{3.9}$$

So, the further away a reward is from the initial state $\mathbf{x}_k$, the less actual reward we will receive from it. For the sake of completeness, we introduce the *reward sequence*

$$\mathbf{r}_{[0:N]} = (\mathsf{r}_0, \ldots, \mathsf{r}_N) \ . \tag{3.10}$$

There is a finite and infinite horizon setting in RL. They mainly differ in the time frame over which the agent plans and makes decisions:

- In the *finite horizon setting*, the agent operates over a fixed, known number of time steps, i.e, N is finite.
- In the *infinite horizon setting*, the agent considers an infinite number of future steps, i.e., $N = \infty$.

> *Bemerkung* 3.2. There is no consensus within the community about whether the reward corresponding to the state $\mathbf{x}_k$ is gained at time index k as in [3.5], or time index $k+1$, as in [3.6]. Here, it is assumed that the reward is gained at time index k and is denoted by r_k.

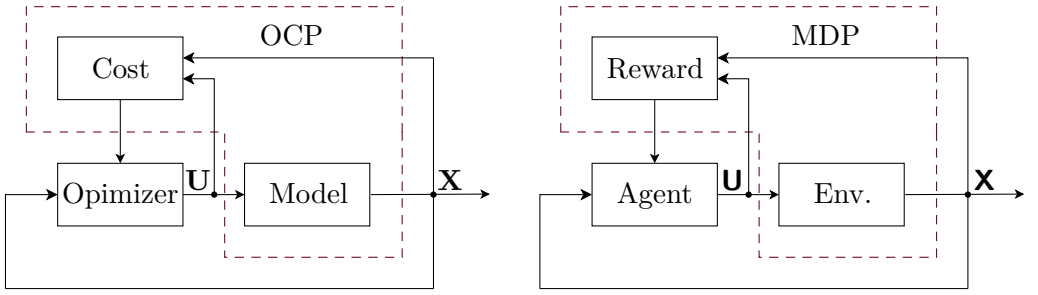
There are two settings for learning and optimization:

- The *episodic/sequential setting*, where the experience is broken up into a series of episodes/sequences. It is our goal to maximize the cumulative reward within a single episode. This setting is typical in scenarios like games or simulations where tasks are naturally episodic and reset after reaching a conclusion.
- The *continuing setting*, where the task doesn't have clear episode boundaries. It is our goal to maximize the cumulative reward over an infinite or very long time horizon, i.e., $N = \infty$. This is more reflective of real-world scenarios like stock market trading or ecosystem management, where the process is ongoing and doesn't reset periodically. In this setting, the notion of discounting future rewards often becomes essential to ensure that the cumulative reward doesn't diverge to infinity.

3.2.3 Different terminologies and interaction models

In control engineering, when addressing a (deterministic) *Optimal Control Problem* (OCP), we refer to a simulation model, a cost function, and a numerical solution method to solve the

dynamical optimization problem. Hence, we adopt the Optimizer-Model-Cost interaction framework, as illustrated in Figure 3.1a. In reinforcement learning, we consider the agent-environment-reward interaction model[2], rooted in the principles of the (stochastic) *Markov Decision Process* (MDP). This interaction model is depicted in Figure 3.1b. In RL,



(a) Optimizer-model-cost interaction model.

(b) Agent-environment-reward interaction model.

Figure 3.1: Interaction models.

the accumulative rewards (return) are maximized and, in OC, the accumulative costs (total cost) are minimized. In control engineering, the environment corresponds to the controlled system, plant or model and the agent to the decision maker or controller. In essence, while there exists a linguistic divergence between control engineering and computational intelligence terminologies, their underlying semantics largely converge. This dichotomy is primarily a byproduct of the distinct historical and academic roots of the two fields. For more information about the different terminologies in Control Systems Engineering and Computational Intelligence, see [3.7, p. 43].

3.2.4 Markov Decision Process

Markov Decision Processes (MDP) formally describe an environment for reinforcement learning. A Markov Process is a memoryless random process, i.e., a sequence of random states with the Markov property (3.4).

Definition 3.2. (Markov Decision Process) A (stationary[a]) Markov Decision Process (MDP) is a fully observable, probabilistic state model. A finite-horizon, discount-reward MDP $\mathcal{M}$ is a tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{R}, \mathsf{p}, \mathsf{p}_{\mathsf{x}_0}, \gamma, N \rangle$ containing:

- $\mathcal{X}$ is the state space
- $\mathcal{U}$ is the input space
- $\mathcal{R}$ is the reward space

[2]Frequently referred to simply as the agent-environment interaction model.

- p is the dynamic function
- $\mathsf{p}_{\mathsf{x}_0}$ is the initial condition function
- $\gamma \in [0, 1]$ is a discount factor and
- N is the horizon length.

[a] The dynamic function is time-independent.

Bemerkung 3.3. Markov models are categorized as either fully or partially observable as well as either autonomous or non-autonomous, cf. Table 3.1. In the partially observable setting, the agent only has access to the output[a] $\mathbf{y}_k$ at each time step k. This model is called *Partially Observable Markov Decision Process* (POMDP) and in addition to the MDP, there is an output-transition probability $\mathsf{o}(\mathbf{y}_k \mid \mathbf{x}_k, \mathbf{u}_k)$. In the autonomous case, there is no input $\mathbf{u}_k$.

[a] Called observation in the reinforcement learning literature.

Table 3.1: Types of Markov Models.

	Fully observable	**Partially observable**
Auto.	Markov Chain	Hidden Markov Model
Non-auto.	Markov Decision Process	Partially Observable Markov Decision Process

The MDP and agent together thereby give rise to an episode

$$\mathbf{x}_0, \mathbf{u}_0, \mathsf{r}_0,\ \mathbf{x}_1, \mathbf{u}_1, \mathsf{r}_1,\ \ldots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathsf{r}_{N-1},\ \mathbf{x}_N, \mathsf{r}_N\ , \tag{3.11}$$

where $\mathbf{x}_k \in \mathcal{X}, \mathbf{u}_k \in \mathcal{U}$ and $\mathsf{r}_k \in \mathcal{R}$.

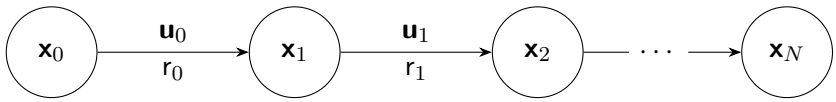


Figure 3.2: Episode.

The states, inputs, and rewards are either *discrete* or *continuous* random variables.

- Discrete variables: These are variables that take values on a finite set. For example, inputs in a Tic-Tac-Toe game are discrete because there are a limited number of squares where a player can place a symbol.

  Beispiel 3.1 (Tic-Tac-Toe game). In the Tic-Tac-Toe game, the board configuration at any given time serves as the outcomes. Each outcome can be represented as a 3×3 matrix or a vector with 9 elements with entries from $\{\mathrm{X}, \mathrm{O}, \text{empty}\}$. That's a total of $3^9 = 19683$ different ways the 3×3 grid can be filled in. The sample space is the set of all possible outcomes and can be

written as $\boldsymbol{\xi} \in \Xi = \{\mathrm{X}, \mathrm{O}, \text{empty}\}^9$. For each element x of the random vector $\mathbf{x}$, a possible mapping from the sample space the real numbers is $\mathsf{x}(\mathrm{X}) = 1$, $\mathsf{x}(\mathrm{O}) = 2$, $\mathsf{x}(\text{empty}) = 3$. Hence, $\mathbf{x}(\boldsymbol{\xi}) \in \mathbb{R}^9$ and $\mathbf{x} \in \mathcal{X} = \{1, 2, 3\}^9 \subseteq \mathbb{R}^9$. The least number of moves required to win a Tic-Tac-Toe game is 5, while the maximum is 9. Assuming player one, using X, goes first and player two, using O, follows, the game proceeds with alternating moves until the board is filled with five X's and four O's. The inputs are discrete and correspond to the placement of $\{\mathrm{X}, \mathrm{O}\}$ in an empty cell. At the start of the game, there are 9 possible inputs, one for each cell on the board. Thus, the first player has $9+7+5+3+1 = 25$ and the second has $8+6+4+2 = 20$ possible moves, not accounting for games ending before the board is full. In a Tic-Tac-Toe game, the reward structure is often based on the game's outcome, which is inherently discrete. The goal of the game is to get three in a row - horizontally, vertically, or diagonally. Play continues until a player achieves this goal or all the spaces are filled with X's and O's. Win-Lose-Draw: A simple way to define the reward is to give a positive value for a win (1), a negative value for a loss (-1), and a smaller value or zero for a draw (0), i.e. $\mathcal{R} = \{1, -1, 0\}$, with $|\mathcal{R}| = 3$. In more sophisticated models, future rewards can be discounted to prioritize short-term gains and a negative reward could be given for each move to encourage the agent to win in fewer steps. This discussion reveals that even elementary games such as Tic-Tac-Toe are founded on complex mathematical concepts.

X	O	X
X	X	O
O	O	X

(a) Win $r = 1$.

X	O	X
X	O	
O	O	X

(b) Loss $r = -1$.

X	O	X
X	O	O
O	X	X

(c) Draw $r = 0$.

Figure 3.3: Reward stucture of the Tic-Toc-Toe game.

Beispiel 3.2 (Frozen Lake). In the Frozen Lake game, the environment is a grid composed of safe tiles (frozen surface) and dangerous tiles (holes). For examlpe, it can be represented as a 4×4 matrix or a vector with 16 elements. In the environment's code, each tile is represented by a letter as follows: (S: starting point, safe), (F: frozen surface, safe), (H: hole, stuck forever) and (G: goal, safe). Each tile can take one of four values $\{\mathrm{S}, \mathrm{F}, \mathrm{H}, \mathrm{G}\}$. By default, the environment is always in the same configuration. The agent A must navigate from the starting point (S) to the goal (G) without falling into holes. The agent's location in the grid is the state and hence there are $16 = |\mathcal{X}| = \{0, 1, \ldots, 15\}$ possible states. Possible inputs are $\{\leftarrow, \downarrow, \rightarrow, \uparrow\}$. We can map the inputs to the real line

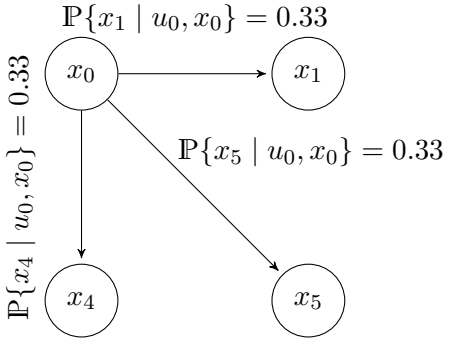
via $\{\mathsf{u}(\leftarrow) = 0, \mathsf{u}(\downarrow) = 1, \mathsf{u}(\rightarrow) = 2, \mathsf{u}(\uparrow) = 3\}$. A sequence of inputs leads the agent to the goal. The agent must learn the optimal path, avoiding holes, and reaching the goal with a minimum number of moves. The game is deterministic in the non-slippery version. In the slippery version, the action the agent takes only has 33% chance of succeeding. In case of failure, one of the three other inputs is randomly taken instead, see Figure 3.4.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

(a) Configuration of the environment.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(b) Current location of the agent as state encoding.



(c) Slippery version.

Figure 3.4: 4×4 Frozen Lake game - configuration, encoding, and stochastic version.

- Continuous variables: These are variables that can take an infinite number of values within a given range. For example, the angle of a rotatory pendulum is a continuous state because it can vary within a range.

  Beispiel 3.3 (Rotatory pendulum). The state of a pendulum is typically characterized by its angle θ and angular velocity ω. These variables are continuous and can vary within a specific range, i.e. $\mathbf{x}^\top = [\theta, \omega] \in \mathbb{R}^2$ and $\mathcal{X} = [0\,\text{rad}, 2\pi\,\text{rad}] \times [-2\,\text{rad/s}, 2\,\text{rad/s}]$ for example. The input for a pendulum could be the torque τ applied to it, which is also a continuous variable, i.e. $\mathsf{u} = \tau \in \mathbb{R}^1$ and $\mathcal{U} = [-1\,\text{Nm}, 1\,\text{Nm}]$. A negative reward can be given based on the angular deviation from the upright position, usually $\theta = 0\,\text{rad}$. The closer the pendulum is to being upright, the smaller the absolute value of the negative reward. To encourage efficient control, the reward can also include a term that penalizes large torque inputs, hence a possible deterministic reward is $r = -(|\theta| + d\tau^2)$, with coefficient $d > 0$.

Value-discrete case

Suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *finite sets* of cardinality $|\mathcal{X}|$, $|\mathcal{U}|$, and $|\mathcal{R}|$. Let us assume that the random variables x' and r have well defined discrete probability distributions dependent only on the preceding state $\mathbf{x}$ and input $\mathbf{u}$. That is, for particular values of these random variables, there is a *dynamic function* $\mathsf{p} : \mathcal{X} \times \mathcal{R} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is

equivalent to the *conditional probability* $\mathbb{P}\{\cdot \mid \cdot\}$. It allows us to predict the future state $\mathbf{x}'$ and current reward r:

$$\mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) = \mathbb{P}\{\mathbf{x}_{k+1} = \mathbf{x}', \mathsf{r}_k = r \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \ . \tag{3.12}$$

The dynamic function gives rise to the *state-transition function* $\mathsf{p}_\mathsf{x} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is given by

$$\mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) = \sum_{r \in \mathcal{R}} \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \tag{3.13}$$

and the *reward function* $\mathsf{p}_\mathsf{r} : \mathcal{R} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is

$$\mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.14}$$

The function p_x defines the dynamics of the MDP

$$\mathbf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \tag{3.15}$$

and the function p_r gives the rewards of the MDP

$$\mathsf{r} \sim \mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.16}$$

The *expected reward* is a two-argument function $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$:

$$\bar{r}(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\mathsf{r} \sim \mathsf{p}_\mathsf{r}(r \mid \mathbf{x}, \mathbf{u})}\{\mathsf{r}\} = \sum_{r \in \mathcal{R}} \sum_{\mathbf{x}' \in \mathcal{X}} r \, \mathsf{p}(\mathbf{x}', r \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.17}$$

Bemerkung 3.4. In reinforcement literature, the random immediate reward is often defined as a three-argument function $\mathsf{R} : \mathcal{X} \times \mathcal{U} \times \mathcal{X}' \to \mathbb{R}$ which also incorporates the next state $\mathbf{x}' \in \mathcal{X}'$, see [3.6]. Note that we can always us the law of total probability (**??**) and state-transition function p_x to convert the formulations because

$$\bar{r}(\mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}' \in \mathcal{X}} R(\mathbf{x}, \mathbf{u}, \mathbf{x}') \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \ . \tag{3.18}$$

For MDPs with *discrete state and input spaces*, we avoid introducing a second index. We denote a specific value-discrete state by $\mathbf{x}_i \in \mathcal{X}$ and a specific value-discrete input by $\mathbf{u}_l \in \mathcal{U}$. In contrast, $\mathbf{x}_k$ and $\mathbf{u}_k$ represent the state and input at a specific time index k. We define the i-to-j-for-l *state-transition probabilities* as

$$p_{ij}^{\mathbf{u}_l} = \mathsf{p}_\mathsf{x}(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{u}_l) = \mathbb{P}[\mathbf{x}_{k+1} = \mathbf{x}_j, \mid \mathbf{x}_k = \mathbf{x}_i, \mathbf{u}_k = \mathbf{u}_l] \ , \tag{3.19}$$

which can be summarized in the *state-transition probability tensor* $\mathbf{P}$, with tensor elements

$$\mathbf{P}^{\mathbf{u}_l}[i, j] = p_{ij}^{\mathbf{u}_l} \ . \tag{3.20}$$

The rows of the state-transition probability sum up to one for each input $\mathbf{u}_l$, i.e.

$$\sum_{j=0}^{|\mathcal{X}|-1} p_{ij}^{\mathbf{u}_l} = 1 \ . \tag{3.21}$$

for all $\mathbf{x}_i \in \mathcal{X}$ and $\mathbf{u}_l \in \mathcal{U}$.

Beispiel 3.4. For $\mathcal{X} = \{x_0, x_1, x_2\}$ and $\mathcal{U} = \{u_0\}$, with state-transition probability matrix

$$\mathbf{P}^{u_0} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} p_{00}^{u_0} & 0 & 0 \\ p_{10}^{u_0} & 0 & 0 \\ p_{20}^{u_0} & 0 & 0 \end{bmatrix}, \tag{3.22}$$

we can draw a state-transition diagram shown in Figure 3.5.

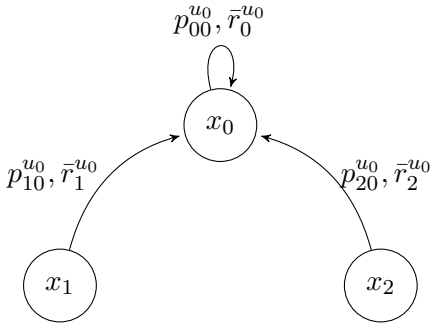


Figure 3.5: A state-transition diagram.

Value-continuous case

Now suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *continuous sets*. Then, the *dynamic function* induces a *state-transition function* $\mathsf{p}_\mathsf{x} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$ of the form

$$\mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) = \int_{\mathcal{R}} \mathsf{p}(\mathbf{x}_{k+1}, r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}r_k \tag{3.23}$$

and a *reward function* $\mathsf{p}_\mathsf{r} : \mathcal{X} \times \mathcal{U} \to [0, 1]$, which is

$$\mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) = \int_{\mathcal{X}} \mathsf{p}(\mathbf{x}_{k+1}, r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}\mathbf{x}_{k+1} \,. \tag{3.24}$$

The function p_x defines the dynamics of the MDP

$$\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \tag{3.25}$$

and the function p_r gives the rewards of the MDP

$$\mathsf{r}_k \sim \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) \,. \tag{3.26}$$

We can can once again determine the *expected rewards* for state-input pairs as a two-argument function $\bar{r}_k : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$,

$$\bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) = \mathbb{E}_{\mathsf{r}_k \sim \mathsf{p}_\mathsf{r}(r_k|\mathbf{x}_k,\mathbf{u}_k)}\{\mathsf{r}_k\} = \int_{\mathcal{R}} r_k \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}_k, \mathbf{u}_k) \mathrm{d}r_k \ . \tag{3.27}$$

3.2.5 Control policy

A control law or a control policy[3] is a function that tells us which is the best input to choose in each state. A policy can be deterministic or stochastic and stationary or nonstationary.

Definition 3.3. (Policy, see [3.3, p. 15]). A (stationary) *stochastic policy* $\pi : \mathcal{X} \times \mathcal{U} \to [0, 1]$ is a distribution over inputs given states

$$\mathbf{u}_k \sim \pi(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.28}$$

A *stationary and deterministic policy* $\pi : \mathcal{X} \to \mathcal{U}$ is a function $\boldsymbol{\mu} : \mathcal{X} \to \mathcal{U}$ that maps the state vector $\mathbf{x}_k$ to control inputs

$$\mathbf{u}_k = \boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.29}$$

A *nonstationary and deterministic policy* $\pi : \mathcal{X} \times \{0, \ldots, N-1\} \to \mathcal{U}$ is a sequence of functions

$$(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{N-1}) \ , \tag{3.30}$$

where $\boldsymbol{\mu}_k$ maps the state vector $\mathbf{x}_k$ to control inputs

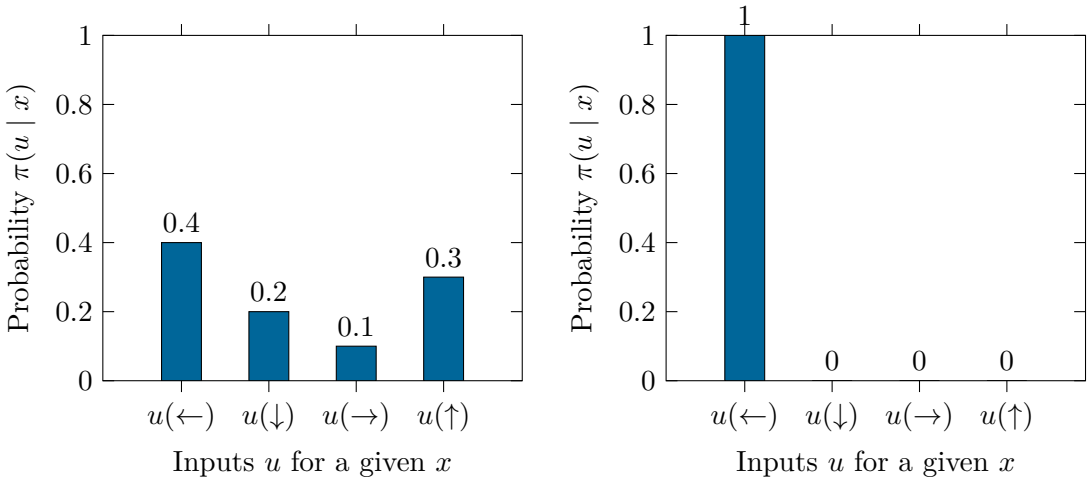
$$\mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) \ . \tag{3.31}$$

The policy is admissible if $\boldsymbol{\mu}_k(\mathbf{x}_k) \in \mathcal{U}(\mathbf{x}_k)$ holds for all $\mathbf{x}_k \in \mathcal{X}_k$ and k. The deterministic policy (3.31) can be written as

$$\pi(\mathbf{u}_k \mid \mathbf{x}_k) = \begin{cases} 1 & \text{if } \mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) \\ 0 & \text{else} \ . \end{cases} \tag{3.32}$$

Beispiel 3.5. In the discrete case, a stochastic policy denoted as $\pi(\mathbf{u} \mid \mathbf{x})$ is a conditional distribution over the inputs $\mathbf{u} \in \mathcal{U}$ given the state $\mathbf{x} \in \mathcal{X}$, $\pi(\mathbf{u} \mid \mathbf{x}) = \mathbb{P}(\mathbf{u} \mid \mathbf{x})$. As an example, if a robot has four inputs $\mathcal{U} = \{\mathsf{u}(\leftarrow), \mathsf{u}(\downarrow), \mathsf{u}(\rightarrow), \mathsf{u}(\uparrow)\}$. The policy at a state $x \in \mathcal{X}$ for such a set of inputs $\mathcal{U}$ is a distribution where the probabilities of the four inputs could be $[0.4, 0.2, 0.1, 0.3]$. Note that we should have $\sum_{u \in \mathcal{U}} \pi(u \mid x) = 1$ for any state x. A deterministic policy is a special case of a stochastic policy in that the distribution $\pi(u \mid x)$ only gives non-zero probability to one particular input, e.g., $[1, 0, 0, 0]$ for our example with four inputs, see Figure 3.6.

[3] It is also often referred to simply as policy.



(a) Stochastic policy.

(b) Deterministic policy.

Figure 3.6: 4×4 Frozen Lake game - policy.

Satz 3.1. *(Probability of observing a rollout, see [3.8, p. 11]) The probability distribution of observing a N-step rollout $\boldsymbol{\tau}$ under a policy π is*

$$\mathsf{p}(\boldsymbol{\tau} \mid \pi) = \mathsf{p}^{\pi}(\boldsymbol{\tau}) = \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.33}$$

Proof. We can proof (3.33) by induction using the product rule (**??**) and the Markov property (3.4). With $\mathsf{p}^{\pi}(\mathbf{x}_0) \doteq \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0)$ and $\mathsf{p}^{\pi}(\mathbf{u}_k \mid \mathbf{x}_k) \doteq \pi(\mathbf{u}_k \mid \mathbf{x}_k)$, we get

$$\begin{aligned}
\mathsf{p}^{\pi}(\mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_0) &= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \underbrace{\mathsf{p}(\mathbf{x}_1, \mathbf{u}_0 \mid \mathbf{x}_0)}_{=\mathsf{p}_{\mathsf{x}}(\mathbf{x}_1 \mid \mathbf{x}_0, \mathbf{u}_0)\pi(\mathbf{u}_0 \mid \mathbf{x}_0)} \\
\mathsf{p}^{\pi}(\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_1, \mathbf{u}_0) &= \mathsf{p}^{\pi}(\mathbf{x}_1, \mathbf{x}_0, \mathbf{u}_0) \underbrace{\mathsf{p}(\mathbf{x}_2, \mathbf{u}_1 \mid \mathbf{x}_1, \cancel{\mathbf{x}_0, \mathbf{u}_0})}_{=\mathsf{p}_{\mathsf{x}}(\mathbf{x}_2 \mid \mathbf{x}_1, \mathbf{u}_1)\pi(\mathbf{u}_1 \mid \mathbf{x}_1)} \\
&= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \\
&\vdots \\
\mathsf{p}^{\pi}(\mathbf{x}_N, \ldots, \mathbf{x}_0, \mathbf{u}_{N-1}, \ldots, \mathbf{u}_0) &= \mathsf{p}_{\mathsf{x}_0}(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k) \ .
\end{aligned} \tag{3.34}$$

□

Whenever any policy π, whether deterministic or probabilistic, is implemented, the resulting process is a Markov process. In value-discrete case, the associated *state-transition probability tensor* is denoted by $\mathbf{P}^{\pi}$. In the deterministic case, $\pi \in \Pi_d$, then at time index

k, the corresponding input $\mathbf{u}_k$ equals $\boldsymbol{\mu}_k(\mathbf{x}_k)$ and we have

$$\mathbf{P}^{\pi}[i,j] = p_{ij}^{\pi} = \mathbb{P}[\mathbf{x}_{k+1} = \mathbf{x}_j \mid \mathbf{x}_k = \mathbf{x}_i, \mathbf{u}_k = \boldsymbol{\mu}_l] \ . \tag{3.35}$$

In the stochastic case, $\boldsymbol{\pi} \in \Pi_p$ and

$$p_{ij}^{\pi} = \mathbb{E}_{\mathbf{u}\sim\pi(\mathbf{u}|\mathbf{x})}\left\{p_{ij}^{\mathbf{u}}\right\} \ . \tag{3.36}$$

3.2.6 Value functions

In reinforcement learning, value functions play a central role in representing and estimating the expected future rewards or returns an agent can obtain from states and inputs. They serve as a foundational concept for many algorithms and provide insight into the quality of different decisions. In the finite horizon setting, we have:

Definition 3.4. (Action-Value Function Q^{π} for a MDP $\mathcal{M}$ and policy π, see [3.6]). The *action-value function*[a] $Q^{\pi} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ for a MDP $\mathcal{M}$ and policy π gives the expected return if you start in state $\mathbf{x}$, take an arbitrary control input $\mathbf{u}$ (which may not have to come from the control law), and then forever after act according to policy π, i.e.

$$Q^{\pi}(\mathbf{x},\mathbf{u}) = \mathbb{E}_{\pi}\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\right\} \tag{3.37a}$$

$$= \mathbb{E}_{\substack{\mathbf{x}_{k+1}\sim\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)\\ \mathbf{u}_k\sim\pi(\mathbf{u}_k|\mathbf{x}_k)}}\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\right\} \ . \tag{3.37b}$$

[a] Also Q-value function.

The action-value function depends on $\mathbf{x}, \mathbf{u}, \pi$ and $\mathsf{p}_{\mathsf{x}}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u})$ but is independent of $\mathbf{X}_{[k+1:N]}$ and $\mathbf{U}_{[k+1:N-1]}$ since they are eliminated by taking the expectation. The action-value function $Q^{\pi}(\mathbf{x},\mathbf{u})$ evaluates how good it is to pick an input $\mathbf{u}$ being in state $\mathbf{x}$.

Definition 3.5. (State-Value Function V^{π}, see [3.6]). The *state-value function* $V^{\pi} : \mathcal{X} \to \mathbb{R}$ gives the expected return if you start in state $\mathbf{x}$ and always act according to policy π, i.e.

$$V^{\pi}(\mathbf{x}) = \mathbb{E}_{\pi}\left\{\mathsf{R}_k\left(\boldsymbol{\tau}_{[k:N]}\right) \mid \mathbf{x}_k = \mathbf{x}\right\} \tag{3.38a}$$

$$= \mathbb{E}_{\mathbf{u}_k\sim\pi(\mathbf{u}_k|\mathbf{x}_k)}\{Q^{\pi}(\mathbf{x}_k,\mathbf{u}_k) \mid \mathbf{x}_k = \mathbf{x}\} \ . \tag{3.38b}$$

Intuitively, for a fixed policy π, the state-value function $V_k^{\pi}(\mathbf{x})$ evaluates how good the situation is in state $\mathbf{x}$. Clearly, in the deterministic case, the action-value function and state-value function are related via

$$\begin{aligned} V^{\pi}(\mathbf{x}_k) &= \mathbb{E}_{\mathbf{u}_k\sim\pi(\mathbf{u}_k|\mathbf{x}_k)}\{Q^{\pi}(\mathbf{x}_k,\mathbf{u}_k) \mid \mathbf{x}_k = \mathbf{x}\} \\ &\overset{(??)}{=} \int_{\mathcal{X}} Q^{\pi}(\mathbf{x}_k,\mathbf{u}_k)\pi(\mathbf{u}_k \mid \mathbf{x}_k)\mathrm{d}\mathbf{x}_k \\ &\overset{(3.32)}{=} Q^{\pi}\left(\mathbf{x}_k, \boldsymbol{\mu}_k(\mathbf{x}_k)\right) \ . \end{aligned} \tag{3.39}$$

Moreover, by definition, the optimal value function produces the maximum return

$$V^*(\mathbf{x}) = \max_{\pi \in \Pi} V^\pi(\mathbf{x}) \quad \text{and} \quad Q^*(\mathbf{x}, \mathbf{u}) = \max_{\pi \in \Pi} Q^\pi(\mathbf{x}, \mathbf{u}) \ . \tag{3.40}$$

Definition 3.6. (Advantage Function A^π for a MDP $\mathcal{M}$, see [3.9]). The *advantage function* $A^\pi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ of a MDP $\mathcal{M}$ for a given policy π, indicates how much better the control input $\mathbf{u}$ is in state $\mathbf{x}$ compared to the average, i.e.

$$A^\pi(\mathbf{x}, \mathbf{u}) = Q^\pi(\mathbf{x}, \mathbf{u}) - V^\pi(\mathbf{x}) \ . \tag{3.41}$$

Advantage functions measure the relative benefit of choosing a specific input over others in a given state.

Bemerkung 3.5. The action-value, state-value, and advantage functions are functions of the time index k in the finite horizon setting.

Beispiel 3.6 (Frozen Lake game continued). In the infinite horizon and value discrete case, the value function can be organized in a table. For instance, for a 4×4 Frozen Lake grid, there are 16 states (each cell is a state) and 4 inputs $\{\leftarrow, \downarrow, \rightarrow, \uparrow\}$. The Q-table is a 16×4 matrix with rows representing states and columns representing inputs. So want to calculate $16 \times 4 = 64$ action-state values. Table 3.2 shows the Q-Table for the example at hand.

State	$u = \mathsf{u}(\leftarrow) = 0$	$u = \mathsf{u}(\downarrow) = 1$	$u = \mathsf{u}(\rightarrow) = 2$	$u = \mathsf{u}(\uparrow) = 3$
$x = 0$	$Q^\pi(0,0)$	$Q^\pi(0,1)$	$Q^\pi(0,2)$	$Q^\pi(0,3)$
$x = 1$	$Q^\pi(1,0)$	$Q^\pi(1,1)$	$Q^\pi(1,2)$	$Q^\pi(1,3)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$x = 14$	$Q^\pi(14,0)$	$Q^\pi(14,1)$	$Q^\pi(14,2)$	$Q^\pi(14,3)$
$x = 15$	$Q^\pi(15,0)$	$Q^\pi(15,1)$	$Q^\pi(15,2)$	$Q^\pi(15,3)$

Table 3.2: Q-Table for the 4×4 Frozen Lake game.

3.2.7 Bellman Expectation Equation

The *Bellman Expectation Equation* serves as the foundational framework for reinforcement learning. It provides a recursive decomposition of the value function, which is essential for evaluating value functions and finding optimal policies.

Value-discrete case

Suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *finite sets* of cardinality $|\mathcal{X}|$, $|\mathcal{U}|$, and $|\mathcal{R}|$. The state-value function can be decomposed into an expected immediate reward plus the discounted

values of the successor state. To show this, we start from the definition of the state-value function (3.38a) and substitute $\mathsf{R}_k = \mathsf{r}_k + \gamma \mathsf{R}_{k+1}$ to get

$$\begin{aligned} V^\pi(\mathbf{x}) &= \mathbb{E}_\pi\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}\} \\ &= \mathbb{E}_\pi\{\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}\} + \gamma \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}\} \ . \end{aligned} \tag{3.42}$$

Note that the distribution $\mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x})$ is a marginal distribution of a distribution that also contains the variables $\mathbf{u}$ and $\mathbf{x}'$, respectively. Thus, we have

$$\mathsf{p}_\mathsf{r}(r \mid \mathbf{x}) = \sum_{\mathbf{x}' \in \mathcal{X}} \sum_{\mathbf{u} \in \mathcal{U}} \mathsf{p}(\mathbf{x}', \mathbf{u}, r_k \mid \mathbf{x}) = \sum_{\mathbf{x}' \in \mathcal{X}} \sum_{\mathbf{u} \in \mathcal{U}} \mathsf{p}(\mathbf{x}', r_k \mid \mathbf{u}, \mathbf{x}) \pi(\mathbf{u} \mid \mathbf{x}) \ , \tag{3.43}$$

where we used $\pi(\mathbf{u} \mid \mathbf{x}) \doteq \mathsf{p}(\mathbf{u} \mid \mathbf{x})$. The first part in (3.42) is the *expected reward* following π because

$$\begin{aligned} \bar{r}^\pi(\mathbf{x}) = \mathbb{E}_\pi\{\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}\} &= \sum_{r_k \in \mathcal{R}} r_k \mathsf{p}_\mathsf{r}(r_k \mid \mathbf{x}) \\ &= \sum_{\mathbf{u} \in \mathcal{U}} \pi(\mathbf{u} \mid \mathbf{x}) \underbrace{\sum_{r_k \in \mathcal{R}} \sum_{\mathbf{x}' \in \mathcal{X}} r_k \ \mathsf{p}(\mathbf{x}', r_k \mid \mathbf{u}, \mathbf{x})}_{\overset{(3.17)}{=} \bar{r}(\mathbf{u}|\mathbf{x})} \ . \end{aligned} \tag{3.44}$$

Let us assume that $R_{k+1} \in \Gamma$ is also a random variable from the finite set Γ. Thus, for the second part in (3.42) follows after some reformulations

$$\begin{aligned} \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}\} &= \sum_{R_{k+1} \in \Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}) \\ &\overset{(??)}{=} \sum_{\mathbf{x}' \in \mathcal{X}} \sum_{R_{k+1} \in \Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}', \mathbf{x}) \mathsf{p}(\mathbf{x}' \mid \mathbf{x}) \\ &= \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}(\mathbf{x}' \mid \mathbf{x}) \underbrace{\sum_{R_{k+1} \in \Gamma} R_{k+1} \mathsf{p}(R_{k+1} \mid \mathbf{x}', \mathbf{x})}_{= \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}' = \mathbf{x}'\} = V^\pi(\mathbf{x}')} \\ &\overset{(??)}{=} \sum_{\mathbf{u}_l \in \mathcal{U}} \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) \pi(\mathbf{u}|\mathbf{x}) V^\pi(\mathbf{x}') \\ &= \mathbb{E}_{\substack{\mathbf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x})}} \{V^\pi(\mathbf{x}')\} \ . \end{aligned} \tag{3.45}$$

Finally, we have found the *discrete Bellman Expectation Equation of the state-value function*

$$V^\pi(\mathbf{x}) = \bar{r}^\pi(\mathbf{x}) + \gamma \mathbb{E}_{\substack{\mathbf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x})}} \{V^\pi(\mathbf{x}')\} \tag{3.46a}$$

$$= \sum_{\mathbf{u} \in \mathcal{U}} \pi(\mathbf{u} \mid \mathbf{x}) \left[\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^\pi(\mathbf{x}') \right] . \tag{3.46b}$$

We can draw similar conclusions for the action-valve function. From (3.37a), we get

$$\begin{aligned} Q^\pi(\mathbf{x},\mathbf{u}) &= \mathbb{E}_\pi\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \\ &= \mathbb{E}_\pi[\mathsf{r}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}] + \gamma \mathbb{E}_\pi\{\mathsf{R}_{k+1} \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\} \\ &= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) \sum_{\mathbf{u}\in\mathcal{U}} \pi(\mathbf{u}'|\mathbf{x}') Q^\pi(\mathbf{x}',\mathbf{u}') \ . \end{aligned} \tag{3.47}$$

Hence, in equivalence to (3.46), we find the *discrete Bellman Expectation Equation of the action-value function*

$$Q^\pi(\mathbf{x},\mathbf{u}) = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\substack{\mathbf{x}'\sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u}) \\ \mathbf{u}'\sim\pi(\mathbf{u}'|\mathbf{x}')}} \{Q^\pi(\mathbf{x}',\mathbf{u}')\} \tag{3.48a}$$

$$= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})} \{V^\pi(\mathbf{x}')\} \tag{3.48b}$$

$$= \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \sum_{\mathbf{x}'\in\mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x},\mathbf{u}) V^\pi(\mathbf{x}') \ . \tag{3.48c}$$

The last two reformulations are due to the relation (3.39).

Value-continuous case

Now suppose $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{R}$ are *continuous sets*. The *continuous Bellman Expectation Equations* can be derived in a similar manner as previously demonstrated. For the *state-value function*, it is given by

$$V^\pi(\mathbf{x}_k) = \bar{r}^\pi(\mathbf{x}_k) + \gamma \mathbb{E}_{\substack{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k) \\ \mathbf{u}_k\sim\pi(\mathbf{u}_k|\mathbf{x}_k)}} \{V^\pi(\mathbf{x}_{k+1})\} \ . \tag{3.49}$$

The recursion for the *action-value function* is

$$Q^\pi(\mathbf{x}_k,\mathbf{u}_k) = \bar{r}(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\substack{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k) \\ \mathbf{u}_{k+1}\sim\pi(\mathbf{u}_{k+1}|\mathbf{x}_{k+1})}} \{Q^\pi(\mathbf{x}_{k+1},\mathbf{u}_{k+1})\} \tag{3.50a}$$

$$= \bar{r}(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)} \{V^\pi(\mathbf{x}_{k+1})\} \ . \tag{3.50b}$$

The last reformulation is once again due to the relation (3.39).

Beispiel 3.7 (Deterministic Bellman Expectation equation). Furthermore, in the *deterministic case*, the Bellman Expectation Equation (3.49) simplifies to the Bellman Equation

$$V^\pi(\mathbf{x}_k) = \bar{r}^\pi(\mathbf{x}_k) + \gamma V^\pi(\mathbf{x}_{k+1}) \ . \tag{3.51}$$

Within this framework, Figure 3.7 illustrates the significance of the Bellman Equation. Here,

- $V^\pi(\mathbf{x}_k)$ can be viewed as a predicted performance,
- $\bar{r}^\pi(\mathbf{x}_k)$ represents the observed immediate reward, and
- $V^\pi(\mathbf{x}_{k+1})$ offers a current prediction of future control inputs.

This concept is called *bootstrapping* in reinforcement learning. It refers to the idea of using the estimates of future value functions to update the current estimate of the value function. In other words, we're "bootstrapping" our value updates based on other estimated values – "Learn a guess from a guess". These concepts are further explored and utilized in the ensuing discussion on temporal difference learning.

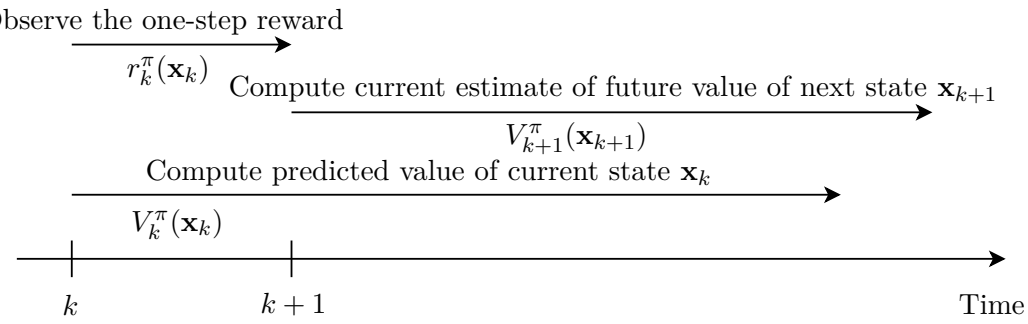


Figure 3.7: The temporal difference perspective of the Bellman Equation illustrates how the equation embodies the processes of action-taking, observation, assessment, and enhancement inherent in reinforcement learning [3.10, p. 469].

3.2.8 Bellman Optimality Equation

The previously obtained results motivate the next theorem.

Value-discrete case

Satz 3.2. *(Value-discrete Bellman Optimality Equation, see [3.11, p. 12]) Let us define the optimal action-state value function* $Q^* : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ *by*

$$Q^*(\mathbf{x}, \mathbf{u}) = \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})} \{ V^*(\mathbf{x}') \} \ . \tag{3.52}$$

Then, Q^* *satisfies the following* Bellman Optimality Equation

$$Q^*(\mathbf{x}, \mathbf{u}) = \bar{r}(\mathbf{x}', \mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}' \sim \mathsf{p}_\mathsf{x}(\mathbf{x}'|\mathbf{x},\mathbf{u})} \left\{ \max_{\mathbf{u}' \in \mathcal{U}} Q^*(\mathbf{x}', \mathbf{u}') \right\} \tag{3.53}$$

with

$$V^*(\mathbf{x}) = \max_{\mathbf{u} \in \mathcal{U}} Q^*(\mathbf{x}, \mathbf{u}) \ . \tag{3.54}$$

Moreover, every policy $\pi \in \Pi_d$ *such that*

$$\boldsymbol{\mu}^*(\mathbf{x}) = \arg\max_{\mathbf{u}\in\mathcal{U}} Q^*(\mathbf{x},\mathbf{u}) \tag{3.55}$$

is optimal.

Proof. Since Q^* is defined by (3.52), it follows that

$$\max_{\mathbf{u}\in\mathcal{U}} Q^*(\mathbf{x},\mathbf{u}) = \max_{\mathbf{u}\in\mathcal{U}} \Big[\bar{r}(\mathbf{x},\mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\{V^*(\mathbf{x}')\}\Big] = V^*(\mathbf{x}) \ . \tag{3.56}$$

This establishes (3.54) and (3.55). Substituting from (3.54) into (3.52) gives (3.53). □

Value-continuous case

In the value-continous case, we the *Bellman Optimality Equation* reads as

$$Q_k^*(\mathbf{x}_k,\mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}\sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)}\left\{\max_{\mathbf{u}_{k+1}\in\mathcal{U}} Q_{k+1}^*(\mathbf{x}_{k+1},\mathbf{u}_{k+1})\right\} \ . \tag{3.57}$$

Obviously, the Bellman Optimality Equation can be used to find the optimal value function V^* and optimal policy π^*.

3.3 Model-based reinforcement learning

In model-based reinforcement learning, we assume to know the dynamic and reward function p_x and p_r, respectively.

3.3.1 Infinite horizon Dynamic Programming

Next we show the *Value and Policy Iteration* used in reinforcement learning. In this section, we will restrict our self to deterministic policies, the infinite horizon setting ($N = \infty$) and the value-discrete case. In this case, the value-discrete Bellman Expectation equation (3.50) with (3.19) can be written as

$$\begin{aligned} Q^\pi(\mathbf{x}, \mathbf{u}) &= \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^\pi(\mathbf{x}') \\ Q^\pi(\mathbf{x}_i, \mathbf{u}_l) &= \bar{r}(\mathbf{x}_i, \mathbf{u}_l) + \gamma \sum_{\mathbf{x}_j \in \mathcal{X}} \mathsf{p}_\mathsf{x}(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{u}_l) V^\pi(\mathbf{x}_j) \\ &= \bar{r}(\mathbf{x}_i, \mathbf{u}_l) + \gamma \sum_{j=1}^{|\mathcal{X}|} p_{ij}^{\mathbf{u}_l} V^\pi(\mathbf{x}_j) \ . \end{aligned} \tag{3.58}$$

Value Iteration

Value iteration is the computation of the state-value function V^π by the Dynamic Programming described in Theorem 3.2. The iteration is performed for the entire state space starting from arbitrarily initialized residual reward [3.6, p. 83]. The iteration rule in Theorem 3.2 represents a fixed point iteration for V^π and converges against V^*, see [3.9]. Therefore, we introduce the *state-value vector*

$$\mathbf{v}^\pi = \begin{bmatrix} V^\pi(\mathbf{x}_0) & \dots & V^\pi\left(\mathbf{x}_{|\mathcal{X}|}\right) \end{bmatrix}^\top \in \mathbb{R}^{|\mathcal{X}|+1} \tag{3.59}$$

and define the *Bellman Iteration Map*[4] $\mathrm{B} : \mathbb{R}^{|\mathcal{X}|+1} \to \mathbb{R}^{|\mathcal{X}|+1}$ via

$$\mathbf{y} \mapsto (\mathrm{B}(\mathbf{y}))[i] = \max_{\mathbf{u}_l \in \mathcal{U}} \left[\bar{r}(\mathbf{x}_i, \mathbf{u}_l) + \gamma \sum_{j=1}^{|\mathcal{X}|} p_{ij}^{\mathbf{u}_l} \mathbf{y}[j] \right]. \tag{3.60}$$

Satz 3.3. *(Theorem 3 in [3.11]) The map* B *is monotone and a contraction with respect to the ℓ_∞-norm. Therefore, the fixed point $\mathbf{v}^{(\infty)}$ of the map* B *satisfies the relation*

$$\mathbf{v}^{(\infty)}[i] = \max_{\mathbf{u}_l \in \mathcal{U}} \left[\bar{r}(\mathbf{x}_i, \mathbf{u}_l) + \gamma \sum_{j=0}^{|\mathcal{X}|-1} p_{ij}^{\mathbf{u}_l} \mathbf{v}^{(\infty)}[j] \right] . \tag{3.61}$$

[4] Called Backup operator in the reinforcement learning literature.

See [3.11] for a proof. Given an initial guess $\mathbf{v}^{(0)} \in \mathbb{R}^{|\mathcal{X}|+1}$, one can iteratively employ the Bellman iteration. These iterations will converge to the unique fixed point, denoted as $\mathbf{v}^{(\infty)}$, of the operator B. The importance of this iterative process is elaborated in the subsequent theorem.

Satz 3.4. *(Theorem 4 in [3.11]) Define $\mathbf{v}^{(\infty)} \in \mathbb{R}^{|\mathcal{X}|+1}$ to be the unique fixed point of* B*, and define $\mathbf{v}^* \in \mathbb{R}^{|\mathcal{X}|+1}$ to equal $V^*(\mathbf{x}), \mathbf{x} \in \mathcal{X}$, where $V^*(\mathbf{x})$ is defined in (3.40). Then $\mathbf{v}^{(\infty)} = \mathbf{v}^*$.*

See [3.11] for a proof. Therefore, the optimal value vector can be computed using the Bellman iteration. However, knowing the optimal value vector does not, by itself, give us an optimal policy. Therefore, after the convergence Bellman iteration, a policy determination according to (3.55) is performed.

The *Infinite Horizon Value Iteration Algorithm* 1 show how to use Value Iteration to find an optimal policy π^*.

Algorithm 1: Infinite Horizon Value Iteration Algorithm

Data: θ is a small number
Result: Find $V^*(\mathbf{x})$ and $\boldsymbol{\mu}^*$,$\forall \mathbf{x} \in \mathcal{X}$
`/* Initialization */`
Initialize $V(\mathbf{x})$ arbitrarily;
$V^{\pi}(\mathbf{x}) \leftarrow 0$,$\forall \mathbf{x} \in \mathcal{X}$;
`/* Loop until convergence */`
$\Delta \leftarrow 0$;
`/* Optimal Value Determination */`
while $\Delta < \theta$ **do**
  for *each* $\mathbf{x} \in \mathcal{X}$ **do**
    $v \leftarrow V^{\pi}(\mathbf{x})$;
    $Q^{\pi}(\mathbf{x}) \leftarrow \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^{\pi}(\mathbf{x}')$;
    $V^{\pi}(\mathbf{x}) \leftarrow \max_{\mathbf{u} \in \mathcal{U}}[Q^{\pi}(\mathbf{x})]$;
    $\Delta \leftarrow \max(\Delta, |v - V^{\pi}(\mathbf{x})|)$;
  end
end
`/* Optimal Policy Determination */`
$Q^*(\mathbf{x}, \mathbf{u}) \leftarrow \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^{\pi}(\mathbf{x}')$;
$\boldsymbol{\mu}^*(\mathbf{x}) \leftarrow \arg\max_{\mathbf{u} \in \mathcal{U}} Q^*(\mathbf{x}, \mathbf{u})$;
return μ^* ,$\forall \mathbf{x} \in \mathcal{X}$

Bemerkung 3.6. Python code of the *Infinite Horizon Value Iteration Algorithm* 1 for the Frozen Lake Example can be found here.

Policy Iteration

When the model is fully known, following Bellman Expectation equation, we can use Dynamic Programming (DP) to iteratively evaluate value functions and improve the policy.

- *Policy Evaluation* (value update) is to compute the state-value functions $V^\pi(\mathbf{x})$ for a given policy π according to the Bellman Expectation equation (3.49).
- Based on the state-value functions, *Policy Improvement* (policy update) generates a better policy $V^{\pi'}(\mathbf{x}) \geq V^\pi(\mathbf{x})$ by acting greedily.

Once again, we show Policy Iteration for the infinite horizon ($N = \infty$) and value-discrete case ($\mathcal{X}$ and $\mathcal{U}$ finite) only.

Policy Evaluation

Define the *state-value vector* $\mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{X}|}$ via

$$\mathbf{v}^\pi = \begin{bmatrix} V^\pi(\mathbf{x}_0) & \dots & V^\pi\left(\mathbf{x}_{|\mathcal{X}|}\right) \end{bmatrix}^\top, \tag{3.62}$$

and the *reward vector* $\mathbf{r}^\pi \in \mathbb{R}^{|\mathcal{X}|+1}$ via

$$\mathbf{r}^\pi = \begin{bmatrix} \bar{r}_0^\pi & \cdots & \bar{r}_{|\mathcal{X}|}^\pi \end{bmatrix}^\top. \tag{3.63}$$

Then, the value-discrete infinite horizon Bellman Expectation equation (3.49) reads in vector notation as

$$\mathbf{v}^\pi = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}^\pi \ . \tag{3.64}$$

Policy Evaluation involves computing the state-value vector $\mathbf{v}^\pi$ for a given policy π solving the Bellman Expectation equation. We also refer to it as the prediction problem. In this context, it is important to note that the induced matrix norm $\|\mathbf{P}^\pi\|_\infty$ is equal to 1, and since $\gamma < 1$, the matrix $\mathbf{I} - \gamma \mathbf{P}^\pi$ is nonsingular. Consequently, for any reward vector $\mathbf{r}^\pi$, there exists a unique $\mathbf{v}^\pi$ that satisfies (3.49). In principle it is possible to deduce from (3.49)

$$\mathbf{v}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r}^\pi \ . \tag{3.65}$$

However, applying this formula can be challenging in practical applications of Markov Decision Problems, mainly due to the large size of the state space $\mathcal{X}$, represented by the integer $|\mathcal{X}|$. Additionally, matrix inversion has a cubic complexity, making it impractical to invert the matrix $\mathbf{I} - \gamma \mathbf{P}^\pi$. Consequently, alternative approaches need to be explored. The Contraction Mapping Theorem provides a feasible solution in such cases.

Satz 3.5. *(Iterative policy evaluation, Theorem 2 in [3.11]) The map* $\mathbf{y} \mapsto \mathrm{T}(\mathbf{y}) = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{y}$ *is monotone and is a contraction with respect to the* ℓ_∞*-norm, with contraction constant* γ*. Therefore, we can choose some vector* $\mathbf{y}^{(0)}$ *arbitrarily, and then define*

$$\mathbf{y}^{(i+1)} = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{y}^{(i)} \ . \tag{3.66}$$

Then $\mathbf{y}^{(i)}$ *converges to the state value vector* $\mathbf{v}^\pi$*.*

See [3.11] for a proof. Similar results can be obtained for the value-continuous case, see [3.4]. Therefore, the transition tensor $\mathbf{P}^\pi$ is replaced by a transition operator.

Policy Improvement

Given a state-value function $V^{\pi}(\mathbf{x})$ of a policy π, it is possible to perform a *policy improvement step*, cf. [3.6, pp. 84–85]. The result is a potentially better strategy π', where $V^{\pi'}(\mathbf{x}) \geq V^{\pi}(\mathbf{x})$ holds. To obtain π', the previous strategy π is no longer followed, but instead the *greedy policy* is chosen, which maximizes the expected reward:

$$\pi' \leftarrow \arg\max_{\mathbf{u}\in\mathcal{U}} Q^{\pi}(\mathbf{x},\mathbf{u}) = \arg\max_{\mathbf{u}\in\mathcal{U}} \left[\bar{r}^{\pi} + \gamma \mathbb{E}_{\mathbf{x}'\sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}'|\mathbf{x},\mathbf{u})}\{V^{\pi}(\mathbf{x}')\}\right] . \tag{3.67}$$

This can be proven by the Policy Improvement Theorem 3.6:

Satz 3.6. *(Policy Improvement Theorem) We consider two policies $\pi(\mathbf{u} \mid \mathbf{x})$ and $\pi'(\mathbf{u} \mid \mathbf{x})$. Let us define*

$$Q^{\pi}(\mathbf{x},\pi') = \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{Q^{\pi}(\mathbf{x},\mathbf{u})\} . \tag{3.68}$$

If $\forall \mathbf{x} \in \mathcal{X}$, we have that $Q^{\pi}(\mathbf{x},\pi') \geq V^{\pi}(\mathbf{x})$, then it holds that

$$V^{\pi}(\mathbf{x}) \leq Q^{\pi}(\mathbf{x},\pi') \leq V^{\pi'}(\mathbf{x}), \forall x . \tag{3.69}$$

This means that π' is at least as good a policy as π.

Proof. By expanding Q^{π}, we can get that $\forall \mathbf{x} \in \mathcal{X}$,

$$\begin{aligned} V^{\pi}(\mathbf{x}) \leq Q^{\pi}(\mathbf{x},\pi') &= \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{Q^{\pi}(\mathbf{x},\mathbf{u})\} \\ &= \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\left\{\mathsf{r}_k + \gamma \underbrace{V^{\pi}(\mathbf{x}')}_{\leq Q^{\pi}(\mathbf{x}',\pi')}\right\} \\ &\leq \mathbb{E}_{\mathbf{u}\sim\pi'(\mathbf{u}|\mathbf{x})}\{\mathsf{r}_k + \gamma Q^{\pi}(\mathbf{x}',\pi')\} \\ &= \mathbb{E}_{\mathbf{u},\mathbf{u}'\sim\pi'}\left\{\mathsf{r}_k + \gamma \mathsf{r}_{k+1} + \gamma^2 V^{\pi}(\mathbf{x}'')\right\} \\ &\leq \dots \\ &\leq \mathbb{E}_{\mathbf{u},\mathbf{u}',\mathbf{u}''\dots\sim\pi'}\left\{\mathsf{r}_k + \gamma \mathsf{r}_{k+1} + \gamma^2 \mathsf{r}_{k+2} + \dots\right\} \\ &= V^{\pi'}(\mathbf{x}) . \end{aligned} \tag{3.70}$$

□

The *Infinite Horizon Policy Iteration Algorithm* 2 shows how to use policy evaluation and policy improvement to find an optimal policy π^*.

Algorithm 2: Infinite Horizon Policy Iteration Algorithm

Data: θ is a small number
`/* Initialization */`
Initialize $V^{\pi}(\mathbf{x})$ arbitrarily;
Randomly initialize policy $\boldsymbol{\mu}$;
`/* Policy Evaluation */`
$\Delta \leftarrow 0$;
while $\Delta < \theta$ **do**
  for *each* $\mathbf{x} \in \mathcal{X}$ **do**
    $v^{\pi} \leftarrow V^{\pi}(\mathbf{x})$;
    $V^{\pi}(\mathbf{x}) \leftarrow \bar{r}^{\pi} + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^{\pi}(\mathbf{x}')$;
    $\Delta \leftarrow \max(\Delta, |v^{\pi} - V^{\pi}(\mathbf{x})|)$;
  end
end
`/* Policy Improvement */`
policy is stable $\leftarrow$ true;
for *each* $\mathbf{x} \in \mathcal{X}$ **do**
  old-policy $\leftarrow \pi$;
  $Q^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}' \mid \mathbf{x}, \mathbf{u}) V^{\pi}(\mathbf{x}')$;
  $\boldsymbol{\mu}(\mathbf{x}) \leftarrow \arg\max_{\mathbf{u} \in \mathcal{U}} Q^{\pi}(\mathbf{x}, \mathbf{u})$;
  if *old-policy* $\neq \pi$ **then**
    policy is stable $\leftarrow$ false;
  end
end
if *policy is stable* **then**
  return $V^{\pi} \approx V^{*}$ and $\pi \approx \pi^{*}$;
else
  go to Policy Evaluation;
end

Finally, we compare Value Iteration and Policy Iteration and draw some conclusions:

Aspect	**Value Iteration**	**Policy Iteration**
Convergence Speed	Faster to optimal value function	Fewer iterations required and converges to the optimal policy faster
Implementation	Simpler, no separate policy needed	More complex with policy evaluation and update
Efficiency per Iteration	Slower due to all states sweep	More efficient with early policy stop
Computational Expense	Varied, generally moderate	Computationally expensive per iteration

Table 3.3: Comparison of Value Iteration and Policy Iteration.

Bemerkung 3.7. Python code of the *Infinite Horizon Policy Iteration Algorithm* 2 for the Frozen Lake Example can be found here.

We will now have a look at a value-continuous example, in particular we investigate the well known *Linear Quadratic Regulator* (LQR) problem from Control Engineering.

Beispiel 3.8 (The Linear Quadratic Regulator problem - infinite time horizon and deterministic case). Consider the problem of the discrete-time linear quadratic regulator (LQR), characterized by deterministic dynamics

$$\mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\mathbf{u}_k \ , \tag{3.71}$$

with k the discrete time index. Let the pair $(\mathbf{\Phi}, \mathbf{\Gamma})$ be reachable. In this example, the state space $\mathcal{X} = \mathbb{R}^n$ and input space $\mathcal{U} = \mathbb{R}^m$ are infinite. The return over an infinite horizon, based on deterministic immediate rewards r_i, is expressed as

$$R_k(\boldsymbol{\tau}_{[k:\infty]}) = \frac{1}{2}\sum_{i=k}^{\infty} r_i(\mathbf{x}_i, \mathbf{u}_i) = \frac{1}{2}\sum_{i=k}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \mathbf{u}_i^\top \mathbf{R}\mathbf{u}_i\Big) \ . \tag{3.72}$$

The return R_k depends on all future states $\mathbf{x}_k, \mathbf{x}_{k+1}, \dots$ and all future control inputs $\mathbf{u}_k, \mathbf{u}_{k+1}, \dots$. We select a stationary control law $\mathbf{u}_k = \boldsymbol{\mu}(\mathbf{x}_k)$ and write the associated action-value function as

$$Q^\pi(\mathbf{x}_k, \mathbf{u}_k) = r(\mathbf{x}_k, \mathbf{u}_k) + V^\pi(\mathbf{x}_{k+1}) \tag{3.73a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.73b}$$

and the associated state-value function as

$$V^\pi(\mathbf{x}_k) = \frac{1}{2}\sum_{i=k}^{\infty} r_i^\pi = \frac{1}{2}\sum_{i=k}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \boldsymbol{\mu}^\top(\mathbf{x}_i)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_i)\Big) \tag{3.74a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k) \ . \tag{3.74b}$$

Note that the state-value function for a fixed control law depends only on the initial state $\mathbf{x}_k$. A difference equation equivalent to the infinite sum (3.74a) is given by

$$\begin{aligned} V^\pi(\mathbf{x}_k) &= \frac{1}{2}\Big(\mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^\top(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k)\Big) + \frac{1}{2}\sum_{i=k+1}^{\infty}\Big(\mathbf{x}_i^\top \mathbf{Q}\mathbf{x}_i + \boldsymbol{\mu}^\top(\mathbf{x}_i)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_i)\Big) \\ &= \frac{1}{2}\Big(\mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^\top(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k)\Big) + V^\pi(\mathbf{x}_{k+1}) \ . \end{aligned} \tag{3.75}$$

Equation (3.75) is exactly the Bellman Equation (3.49) for the LQR problem. Assuming a quadratic state-value function

$$V^\pi(\mathbf{x}_k) = \frac{1}{2}\mathbf{x}_k^\top \mathbf{P}\mathbf{x}_k \ , \tag{3.76}$$

with positive-definite matrix $\mathbf{P}$, yields the Bellman Equation

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{P}\mathbf{x}_k = \mathbf{x}_k^{\mathrm{T}}\mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^{\top}(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k) + \mathbf{x}_{k+1}^{\top}\mathbf{P}\mathbf{x}_{k+1} \ , \tag{3.77}$$

which, using the state equation (3.71), can be written as

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{Q}\mathbf{x}_k + \boldsymbol{\mu}^{\top}(\mathbf{x}_k)\mathbf{R}\boldsymbol{\mu}(\mathbf{x}_k) + (\boldsymbol{\Phi}\mathbf{x}_k + \boldsymbol{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k))^{\top}\mathbf{P}(\boldsymbol{\Phi}\mathbf{x}_k + \boldsymbol{\Gamma}\boldsymbol{\mu}(\mathbf{x}_k)) \ . \tag{3.78}$$

Assuming a stationary state feedback control law $\boldsymbol{\mu}(\mathbf{x}_k) = -\mathbf{K}\mathbf{x}_k$ with control gain $\mathbf{K}$, we get

$$2V^{\pi}(\mathbf{x}_k) = \mathbf{x}_k^{\top}\mathbf{P}\mathbf{x}_k = \mathbf{x}_k^{\top}\mathbf{Q}\mathbf{x}_k + \mathbf{x}_k^{\top}\mathbf{K}^{\top}\mathbf{R}\mathbf{K}\mathbf{x}_k + \mathbf{x}_k^{\top}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})^{\top}\mathbf{P}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})\mathbf{x}_k \ . \tag{3.79}$$

For all state trajectories, we find

$$(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K})^{\top}\mathbf{P}(\boldsymbol{\Phi} - \boldsymbol{\Gamma}\mathbf{K}) - \mathbf{P} + \mathbf{Q} + \mathbf{K}^{\top}\mathbf{R}\mathbf{K} = \mathbf{0} \ . \tag{3.80}$$

This is a Lyapunov equation in Josephs form. That is, the Bellman Equation (3.77) for the discrete-time LQR problem is equivalent to a Lyapunov equation.

3.3.2 Finite horizon Dynamic Programming

Finite horizon Dynamic Programming (DP) is based on Bellman's optimality principle, which states that every optimal solution of (3.40) is composed of optimal partial solutions. The basic idea of DP is to divide the problem into subproblems, which can be solved more easily, and to assemble these partial solutions into the overall solution. We will show the value-continuous Dynamic Programming Algorithm for MDPs only.

Satz 3.7. *(Bellman's Optimality Principle, see [3.3, p. 20]).*
Let $\pi^* = \left(\boldsymbol{\mu}_0^*, \boldsymbol{\mu}_1^*, \ldots, \boldsymbol{\mu}_{N-1}^*\right)$ *be the optimal policy for*

$$V_0^{\pi}(\mathbf{x}_0) = \mathbb{E}_{\pi}\left\{\mathsf{R}_0\left(\boldsymbol{\tau}_{[0:N]}\right) \mid \mathbf{x}_0 = \mathbf{x}_0\right\} \ . \tag{3.81}$$

Considering the subproblem of (3.81) *where, starting from the state* $\mathbf{x}_{k+1}$*, the expected returns*

$$V_{k+1}^{\pi}(\mathbf{x}_{k+1}) = \mathbb{E}_{\pi}\left\{\mathsf{R}_{k+1}\left(\boldsymbol{\tau}_{[k+1:N]}\right) \mid \mathbf{x}_{k+1} = \mathbf{x}_{k+1}\right\} \ . \tag{3.82}$$

are to be maximized, then the truncated policy $\left(\boldsymbol{\mu}_{k+1}^*, \boldsymbol{\mu}_{k+2}^*, \ldots, \boldsymbol{\mu}_{N-1}^*\right)$ *is optimal for the subproblem.*

The underlying idea behind the Principle of Optimality is straightforward. If the truncated policy $\left(\boldsymbol{\mu}_{k+1}^*, \boldsymbol{\mu}_{k+2}^*, \ldots, \boldsymbol{\mu}_{N-1}^*\right)$ were not optimal as stated, we would be able to increase the return further by switching to an optimal policy for the subproblem once we reach

$\mathbf{x}_{k+1}$. Based on Theorem 3.2, the *Dynamic Programming Algorithm* can be stated. The algorithm constructs optimal value functions

$$V_N^*(\mathbf{x}_N), V_{N-1}^*(\mathbf{x}_{N-1}), , \ldots, V_0^*(\mathbf{x}_0) \tag{3.83}$$

starting from $V_N^*(\mathbf{x}_N)$ and proceeding backwards in time.
Proposition 1. (Dynamic Programming Algorithm, see [3.3, p. 23] - backward pass). For every initial state $\mathbf{x}_0$, the maximum expected return $V_0^*(\mathbf{x}_0)$ of the basic problem is equal to $V_0^\pi(\mathbf{x}_0)$, given by the last step of the following algorithm, which proceeds from $\gamma^N \bar{r}_N$ backward in time from $k = N-1$ to $k = 0$, for all $\mathbf{x}_k \in \mathcal{X}$:

$$\begin{aligned} V_N^*(\mathbf{x}_N) &= \gamma^N \bar{r}_N(\mathbf{x}_N) \\ V_k^*(\mathbf{x}_k) &= \max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k, \mathbf{u}_k) \ , \end{aligned} \tag{3.84}$$

with

$$Q_k^*(\mathbf{x}_k, \mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\times(\mathbf{x}_{k+1} | \mathbf{x}_k, \mathbf{u}_k)} \{ V_{k+1}^*(\mathbf{x}_{k+1}) \} \ . \tag{3.85}$$

Furthermore, if $\mathbf{u}_k^* = \boldsymbol{\mu}_k^*(\mathbf{x}_k)$ satisfies the Bellman iteration (3.84) for each $\mathbf{x}_k$ and k, the policy $\pi^* = \left(\boldsymbol{\mu}_0^*, \boldsymbol{\mu}_1^*, \ldots, \boldsymbol{\mu}_{N-1}^* \right)$ is optimal.

Once the value functions $V_0^*(\mathbf{x}_0), \ldots, V_N^*(\mathbf{x}_N)$ have been obtained, we can use the following forward algorithm to construct an optimal control input sequence $\left(\mathbf{u}_0^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_{N-1}^* \right)$ and the corresponding state trajectory $\left(\mathbf{x}_0^*, \mathbf{x}_1^*, \ldots, \mathbf{x}_{N-1}^* \right)$ for a given initial state $\mathbf{x}_0$.
Proposition 2. (Construction of an optimal rollout, see [3.7, p. 12] - forward pass). Starting at the initial condition $\mathbf{x}_0^* \sim \mathsf{p}_{\times_0}(\mathbf{x}_0^*)$, compute forward in time from $k = 0$ to $k = N-1$ the control input via

$$\mathbf{u}_k^* = \boldsymbol{\mu}_k^*(\mathbf{x}_k^*) = \arg \max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k^*, \mathbf{u}_k) \tag{3.86}$$

with

$$Q_k^*(\mathbf{x}_k^*, \mathbf{u}_k) = \bar{r}_k(\mathbf{x}_k^*, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\times\left(\mathbf{x}_{k+1}^* | \mathbf{x}_k^*, \mathbf{u}_k\right)} \{ V_{k+1}^*(\mathbf{x}_{k+1}) \} \tag{3.87}$$

and optimal state trajectory according to

$$\mathbf{x}_{k+1}^* \sim \mathsf{p}_\times(\mathbf{x}_{k+1}^* \mid \mathbf{x}_k^*, \mathbf{u}_k^*) \ . \tag{3.88}$$

See [3.3, p. 25] and [3.7, p. 11] for more information on the Dynamic Programming Algorithm. The *Finite Horizon Value Iteration Algorithm* 3 shows how to use Value Iteration to find an optimal policy π^*.

Algorithm 3: Finite Horizon Value Iteration Algorithm

Data: ϵ is a small number
Result: Find $V_k^*(\mathbf{x}_k)$ and π^* ,$\forall \mathbf{x}_k \in \mathcal{X}$
`/* Initialization */`
Initialize $V_k^\pi(\mathbf{x}_k)$ arbitrarily, expect the $V_N^\pi(\mathbf{x}_N)$;
$V_k^\pi(\mathbf{x}_k) \leftarrow 0, \forall \mathbf{x}_k \in \mathcal{X}$;
`/* Loop until convergence */`
$\Delta \leftarrow 0$;
`/* Optimal Value Determination */`
for $k = N-1, \ldots, 0$ **do**
  for *each* $\mathbf{x}_k \in \mathcal{X}$ **do**
    $v_k \leftarrow V_k^\pi(\mathbf{x}_k)$;
    $Q_k^\pi(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)} \left\{ V_{k+1}^\pi(\mathbf{x}_{k+1}) \right\}$;
    $V_k^\pi(\mathbf{x}_k) \leftarrow \max_{\mathbf{u}_k \in \mathcal{U}} Q_k^\pi(\mathbf{x}_k, \mathbf{u}_k)$;
    $\Delta \leftarrow \max(\Delta, |v_k - V_k^\pi(\mathbf{x}_k)|)$;
  end
end
`/* Optimal Policy Determination */`
$Q_k^*(\mathbf{x}_k, \mathbf{u}_k) \leftarrow \bar{r}_k(\mathbf{x}_k, \mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)} \left\{ V_{k+1}^*(\mathbf{x}_{k+1}) \right\}$;
$\boldsymbol{\mu}_k^*(\mathbf{x}_k) \leftarrow \arg\max_{\mathbf{u}_k \in \mathcal{U}} Q_k^*(\mathbf{x}_k, \mathbf{u}_k)$;
return π^* ,$\forall \mathbf{x}_k \in \mathcal{X}$

Beispiel 3.9 (The Linear Quadratic Regulator problem - finite time horizon and stochastic case). In the discrete-time LQR design, the optimal policy π^* is determined for a linear time-invariant system with discrete-time evolution

$$\mathbf{x}_{k+1} = \boldsymbol{\Phi}_k \mathbf{x}_k + \boldsymbol{\Gamma}_k \mathbf{u}_k + \mathbf{w}_k \ . \tag{3.89}$$

The disturbance $\mathbf{w}_k$ is characterized by the following properties, see [3.12],

$$\begin{aligned} \mathbb{E}\{\mathbf{w}_k\} &= \mathbf{0} \\ \mathbb{E}\left\{\mathbf{w}_k \mathbf{w}_k^\top\right\} &= \mathbf{W} \ . \end{aligned} \tag{3.90}$$

The deterministic rewards are given by

$$r_k(\mathbf{x}_k, \mathbf{u}_k) = \frac{1}{2}(\mathbf{x}_k^\top \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^\top \mathbf{R}_k \mathbf{u}_k) \tag{3.91}$$

$$r_N(\mathbf{x}_N) = \frac{1}{2}\mathbf{x}_N^\top \mathbf{Q}_N \mathbf{x}_N \ , \tag{3.92}$$

where $\mathbf{Q}_k \in \mathbb{R}^{n \times n}$ is positive semi-definite and $\mathbf{R}_k \in \mathbb{R}^{m \times m}$ is positive definite for all $k = 0, \ldots, N$. To determine the optimal policy π^* in the form

$$\mathbf{u}_k = \boldsymbol{\mu}_k(\mathbf{x}_k) = -\mathbf{K}_k \mathbf{x}_k \ , \tag{3.93}$$

we apply the Dynamic Programming Algorithm from Theorem 1.

Starting from the terminal reward $r_N(\mathbf{x}_N)$, a backward recursion is performed based on the Bellman iteration (3.84). We initialize by

$$V_N(\mathbf{x}_N) = \frac{1}{2}\mathbf{x}_N^\top \mathbf{Q}_N \mathbf{x}_N \equiv \frac{1}{2}\mathbf{x}_N^\top \mathbf{P}_N \mathbf{x}_N \ . \tag{3.94}$$

Substituting the system dynamics $\mathbf{x}_N = \mathbf{\Phi}_k \mathbf{x}_{N-1} + \mathbf{\Gamma}_k \mathbf{u}_{N-1} + \mathbf{w}_{N-1}$ results in

$$\begin{aligned} V_{N-1}(\mathbf{x}_{N-1}) &= \min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \left[r_{N-1}(\mathbf{x}_{N-1}\ , \mathbf{u}_{N-1}) + \mathbb{E}_{\mathbf{w}_{N-1}}\{V_N^*(\mathbf{x}_N)\} \right] \\ &= \frac{1}{2} \min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \left[\mathbf{x}_{N-1}^\top \mathbf{Q}_{N-1}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^\top \mathbf{R}_{N-1}\mathbf{u}_{N-1} + \mathbf{x}_N^\top \mathbf{P}_N \mathbf{x}_N \right] . \end{aligned} \tag{3.95}$$

After rearranging, we have

$$\begin{aligned} V_{N-1}^*(\mathbf{x}_{N-1}) = \frac{1}{2} \min_{\mathbf{u}_{N-1}\in\mathbb{R}^m} \Big[& \mathbf{x}_{N-1}^\top (\mathbf{Q}_{N-1} + \mathbf{\Phi}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1}) \mathbf{x}_{N-1} \\ & + \mathbf{u}_{N-1}^\top (\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1}) \mathbf{u}_{N-1} \\ & + 2\mathbf{u}_{N-1}^\top (\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1}) \mathbf{x}_{N-1} \\ & + \mathbb{E}_{\mathbf{w}_{N-1}} \left\{ \mathbf{w}_{N-1}^\top \mathbf{P}_N \mathbf{w}_{N-1} \right\} \Big] \ . \end{aligned} \tag{3.96}$$

Note that this optimization problem is convex in $\mathbf{u}_{N-1}$ as $\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1} > 0$. Therefore, any local minimum is a global minimum, and therefore we can simply apply the first order optimality conditions. Differentiating gives

$$\left(\frac{\partial}{\partial \mathbf{u}_{N-1}} V_{N-1}^* \right)(\mathbf{x}_{N-1}) = (\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1})\mathbf{u}_{N-1} + (\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1})\mathbf{x}_{N-1} \tag{3.97}$$

and setting this to zero yields

$$\mathbf{u}_{N-1}^* = -(\mathbf{R}_{N-1} + \mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Gamma}_{N-1})^{-1} (\mathbf{\Gamma}_{N-1}^\top \mathbf{P}_N \mathbf{\Phi}_{N-1}) \mathbf{x}_{N-1} \ , \tag{3.98}$$

which we write

$$\mathbf{u}_{N-1}^* = -\mathbf{K}_{N-1}\mathbf{x}_{N-1} \ , \tag{3.99}$$

which is a time-varying linear feedback control law. Plugging this feedback policy into (3.96) results in

$$\begin{aligned} V_{N-1}^*(\mathbf{x}_{N-1}) = \mathbf{x}_{N-1}^\top & (\mathbf{Q}_{N-1} + \mathbf{K}_{N-1}^\top \mathbf{R}_{N-1}\mathbf{K}_{N-1} \\ & + (\mathbf{\Phi}_{N-1} + \mathbf{\Gamma}_{N-1}\mathbf{K}_{N-1})^\top \mathbf{P}_N (\mathbf{\Phi}_{N-1} + \mathbf{\Gamma}_{N-1}\mathbf{K}_{N-1})) \mathbf{x}_{N-1} \ . \end{aligned} \tag{3.100}$$

Critically, this implies that the cost is always a positive semi-definite quadratic function of the state. Because the optimal policy is always linear, and the optimal

cost is always quadratic, the DP recursion may be recursively performed backward in time and the minimization may be performed analytically. Following the same procedure, we can write the DP recursion for the discrete-time LQR controller. The optimal feedback gain $\mathbf{K}_k$ according to control law $\mathbf{u}_k^* = -\mathbf{K}_k \mathbf{x}_k$ is obtained from a *backward calculation* from $k = N-1$ to $k = 0$ starting from $\mathbf{P}_N = \mathbf{Q}_N$:

$$\mathbf{K}_k = (\mathbf{R}_k + \mathbf{\Gamma}_k^\top \mathbf{P}_{k+1} \mathbf{\Gamma}_k)^{-1} (\mathbf{\Gamma}_k^\top \mathbf{P}_{k+1} \mathbf{\Phi}_k) \tag{3.101}$$

$$\mathbf{P}_k = \mathbf{Q}_k + \mathbf{K}_k^\top \mathbf{R}_k \mathbf{K}_k + (\mathbf{\Phi}_k + \mathbf{\Gamma}_k \mathbf{K}_k)^\top \mathbf{P}_{k+1} (\mathbf{\Phi}_k + \mathbf{\Gamma}_k \mathbf{K}_k) \ . \tag{3.102}$$

To determine the optimal control sequence $\mathbf{U}_{[0:N-1]}^* = \left(\mathbf{u}_0^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_{N-1}^*\right)$ and optimal state sequence $\mathbf{X}_{[0:N]}^* = (\mathbf{x}_0^*, \mathbf{x}_1^*, \ldots, \mathbf{x}_N^*)$, a *forward calculation* from $k = 0$ to $k = N-1$ is performed starting from $\mathbf{x}_0 = \mathbf{x}_0^*$ using the optimal control law and discrete-time evolution

$$\begin{aligned} \mathbf{u}_0^* &= -\mathbf{K}_0 \mathbf{x}_0^* && \rightarrow \quad \mathbf{x}_1^* = \mathbf{\Phi}_k \mathbf{x}_0^* + \mathbf{\Gamma}_k \mathbf{u}_0^* \\ &\vdots && \quad\ \vdots \\ \mathbf{u}_{N-1}^* &= -\mathbf{K}_{N-1} \mathbf{x}_{N-1}^* && \rightarrow \quad \mathbf{x}_N^* = \mathbf{\Phi}_k \mathbf{x}_{N-1}^* + \mathbf{\Gamma}_k \mathbf{u}_{N-1}^* \ . \end{aligned} \tag{3.103}$$

The relation offers multiple insights. Even when $\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{Q}, \mathbf{R}$ are constant, the policy is still time-varying. This phenomenon arises because early control efforts lead to reduced state costs in all subsequent time steps. As the episode nears its end, the benefits of this early control wane, and the control effort decrease. However, for a linear time-invariant system where $(\mathbf{\Phi}, \mathbf{\Gamma})$ is reachable, the feedback gain $\mathbf{K}_k$ approach a constant value as the episode length grows infinitely.

3.4 Model-free reinforcement learning

The methods described before require knowledge of the system dynamics. However, this is not the case in model-free reinforcement learning, where the dynamics are unknown. As a result, it becomes necessary to develop solution methods that do not rely on this explicit knowledge.

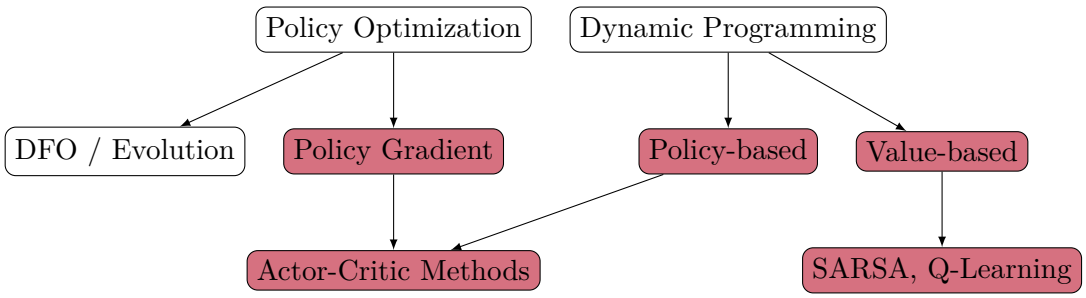


Figure 3.8: Classification of model-free RL algorithms [3.13, p. 4].

In reinforcement learning (RL), there are numerous options for approximating elements like policies, value functions, and dynamic models. Figure 3.8 provides a *classification of model-free RL algorithms*. There are three main approaches: (i.) *Policy Optimization*, (ii.) *Approximate Dynamic Programming*, and (iii.) *Hybrid methods*.

Policy Optimization techniques focus on optimizing the policy, the function mapping the agent's state to its next control input. They consider reinforcement learning as a numerical optimization problem where we optimize the expected reward with respect to the policy's parameters. There are two ways to optimize a policy: *Derivative-free Optimization* (DFO) algorithms and policy gradient methods. DFO algorithms work by perturbing the policy parameters, assessing the performance, and then moving towards better performance. They're simple to implement but scale poorly with increased parameters. On the other hand, *Policy Gradient Methods* estimate the policy improvement direction using various quantities measured by the agent, thus avoiding parameter perturbation. They are more complex to implement but can optimize larger policies than DFO algorithms.

Approximate Dynamic Programming (ADP), is based on learning value functions, predicting the agent's potential reward. ADP algorithms strive to satisfy certain consistency equations that the true value functions obey. Known algorithms for exact solutions in finite state-input RL problems are *policy-based* and *value-based*. They can be combined with function approximation in multiple ways; currently, leading descendents of value iteration focus on *approximating Q-functions*.

Lastly, *Actor-Critic* methods incorporate elements from both policy optimization and dynamic programming. They optimize a policy using value functions for faster optimization and often utilize ideas from approximate dynamic programming for fitting the value functions.

We discuss five key model-free reinforcement learning algorithms. Table 3.4 summarizes the use of these model-free reinforcement learning algorithms. It categorizes these algorithms based on their applicability in discrete and continuous input spaces. On-policy and

off-policy learning refer to how an algorithm learns a policy. On-policy learning improves the policy used to make decisions, while off-policy learning improves a different policy from the one used to generate data.

Algorithm	Policy	Discrete	Continuous	Tabular
Monte Carlo	on	Yes	No	Yes
SARSA	on	Yes	No	Yes
Q-Learning	off	Yes	No	Yes
Deep Q-Network	off	Yes	Yes	No
Policy Gradient	on/off	Yes	Yes	No

Table 3.4: Use of reinforcement learning algorithms in different input spaces.

In the following section, we will specifically focus on what are known as *tabular solution methods*, see [3.6, p. 23], which are applicable in scenarios with an infinite horizon as well as discrete input and state spaces.

3.4.1 Generalized Policy Iteration

In model-free RL, the use of the action-value function $Q^\pi(\mathbf{x}, \mathbf{u})$ is favored over the state-value function $V^\pi(\mathbf{x})$ because $Q^\pi(\mathbf{x}, \mathbf{u})$ eliminates the need to know the state-transition and reward function explicitly, which is consistent with the goal of model-free RL to learn optimal policies without a model of the environment, cf. (3.67). Let us introduce the action-value table/matrix

$$\mathbf{Q}^\pi = \begin{bmatrix} Q^\pi(\mathbf{x}_0, \mathbf{u}_0) & Q^\pi(\mathbf{x}_0, \mathbf{u}_1) & \dots & Q^\pi\left(\mathbf{x}_0, \mathbf{u}_{|\mathcal{U}|-1}\right) \\ Q^\pi(\mathbf{x}_1, \mathbf{u}_0) & Q^\pi(\mathbf{x}_1, \mathbf{u}_1) & \dots & Q^\pi\left(\mathbf{x}_1, \mathbf{u}_{|\mathcal{U}|-1}\right) \\ \vdots & \ddots & \vdots & \\ Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1}, \mathbf{u}_0\right) & Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1}, \mathbf{u}_1\right) & \dots & Q^\pi\left(\mathbf{x}_{|\mathcal{X}|-1}, \mathbf{u}_{|\mathcal{U}|-1}\right) \end{bmatrix}. \tag{3.104}$$

The *Generalized Policy Iteration* (GPI) algorithm refers to an iterative procedure to improve the policy when combining policy evaluation and improvement, cf. Figure 3.9:

$$\pi^{(0)} \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(0)} \xrightarrow{\text{improve}} \pi^{(1)} \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(1)} \dots \xrightarrow{\text{improve}} \pi^* \xrightarrow{\text{evaluate}} (\mathbf{Q}^\pi)^{(*)} \tag{3.105}$$

In GPI, the action-value function $\mathbf{Q}^\pi$ is approximated repeatedly to be closer to the true value $(\mathbf{Q}^\pi)^{(*)}$ of the current policy and in the meantime, the policy π is improved repeatedly to approach optimality, i.e., π^*. This policy iteration process works and always converges to the optimality, independent of the granularity and other details of the two processes. Almost all reinforcement learning methods are well described as GPI [3.6, p. 86].

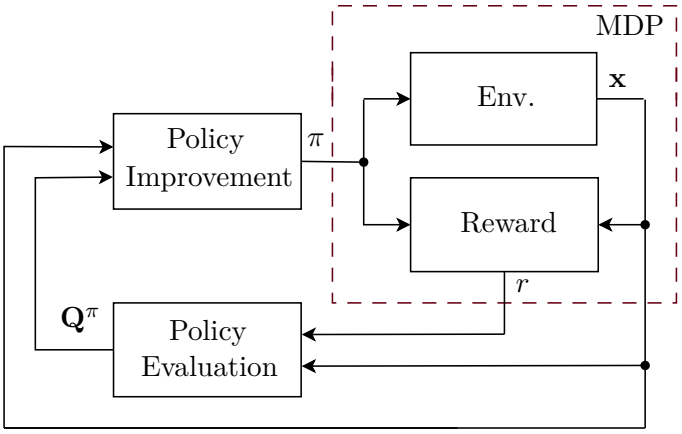


Figure 3.9: Generalized Policy Iteration.

Exploration vs. Exploitation

Efficient decision-making is often challenged by the need to balance exploration and exploitation. Exploration involves seeking new information to improve future decisions, while exploitation leverages current knowledge to maximize immediate rewards.

- Greedy input: When an agent chooses an input with the highest estimated value at the moment. The agent exploits its current knowledge by selecting the greedy input.
- Non-greedy input: When the agent does not choose the action with the highest estimated value and forgoes the immediate reward, hoping to gather more information about other inputs.
- Exploration: This enables the agent to improve its knowledge of each input, which hopefully leads to long-term benefits.
- Exploitation: The agent can choose the greedy input to obtain the highest reward for a short-term advantage. However, a purely greedy input selection can result in suboptimal behavior.

This creates a dilemma between exploration and exploitation, as an agent cannot explore and exploit simultaneously. Striking a balance between *exploration* and *exploitation* is crucial in model-free RL.

Policy Improvement

Policy improvement refers to acting greedily and exploiting current knowledge. In view of (3.67), given an action-value function $Q^\pi(\mathbf{x}, \mathbf{u})$, the greedy policy

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \begin{cases} 1 & \text{if } \mathbf{u} = \arg\max_{\mathbf{w} \in \mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w}) \\ 0 & \text{else} \end{cases}, \tag{3.106}$$

is selected to maximize the expected reward based on current knowledge. Several strategies exist to balance exploration and exploitation:

ϵ-Greedy Method
With ϵ-greedy, at each step in state $\mathbf{x}$, the agent selects a random input with a fixed probability $\epsilon \in [0, 1]$:

- With probability $\frac{\epsilon}{|\mathcal{U}|} + 1 - \epsilon$, choose $\mathbf{u} = \arg\max_{\mathbf{w} \in \mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w})$ and
- with probability $\frac{\epsilon}{|\mathcal{U}|}$, select any input uniformly at random.

Hence, the ϵ-greedy policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \begin{cases} \frac{\epsilon}{|\mathcal{U}|} + 1 - \epsilon & \text{if } \mathbf{u} = \arg\max_{\mathbf{w} \in \mathcal{U}} Q^\pi(\mathbf{x}, \mathbf{w}) \\ \frac{\epsilon}{|\mathcal{U}|} & \text{else} \end{cases}. \tag{3.107}$$

Compared to the greedy method (3.67), the ϵ-greedy method helps in exploring the input space.

Softmax Strategy
A downside of ϵ-greedy exploration is that it randomly picks any possible input for exploration. This means it's just as likely to pick the worst option as it is to pick the almost best one. In contrast, the softmax[5] strategy utilizes input-selection probabilities which are determined by ranking the action-value function estimates using a Boltzmann distribution. The softmax policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \operatorname*{soft\,max}_{\mathbf{u} \in \mathcal{U}} \beta Q^\pi(\mathbf{x}, \mathbf{u}) = \frac{\exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x}, \mathbf{u})\right)}{\sum_{\mathbf{w} \in \mathcal{U}} \exp\left(\frac{1}{\beta} Q^\pi(\mathbf{x}, \mathbf{w})\right)}. \tag{3.108}$$

Here, β is the temperature parameter that controls the stochasticity of the policy. A low β makes the policy more deterministic, choosing the action with the highest Q-value with higher probability. A high β makes the policy more exploratory, distributing the selection probability more uniformly across inputs. In practice, the agent uses the Softmax Strategy to randomly select the next input by sampling from the distribution.

[5] Actually, the softmax function is a softened version of the argmax function. Therefore, it could be more appropriately termed softargmax.

Upper Confidence Bound
The Upper Confidence Bound (UCB) strategy addresses the exploration-exploitation trade-off by selecting inputs based on both their estimated value and the uncertainty of those estimates. It adds a term to the action-value function that increases with the uncertainty of the action value, encouraging exploration of less certain actions. The UCB policy is defined as

$$\pi(\mathbf{u} \mid \mathbf{x}) \leftarrow \arg\max_{\mathbf{w} \in \mathcal{U}} \left(Q^{\pi}(\mathbf{x}, \mathbf{w}) + c \sqrt{\frac{\ln(t)}{N(\mathbf{x}, \mathbf{w})}} \right) , \tag{3.109}$$

where

- c is a parameter that controls the degree of exploration.
- t is the total number of times the state $\mathbf{x}$ has been visited.
- $N(\mathbf{x}, \mathbf{w})$ is the number of times input $\mathbf{w}$ has been chosen in state $\mathbf{x}$.

The exploration term $c\sqrt{\frac{\ln t}{N(\mathbf{x},\mathbf{w})}}$ decreases as $N(\mathbf{x}, \mathbf{w})$ increases, reducing exploration over time as more information is gathered. This method balances the need to explore actions with high uncertainty and the need to exploit actions with high estimated rewards.

Policy Evaluation

If the dynamics of the environment are unknown, several methods can be used for *policy evaluation*. In policy evaluation, we apply the update rule

$$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) + \alpha\delta \tag{3.110}$$

to improve the estimated action-value function $\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$. Here, $\alpha \in (0, 1]$ is the learning rate and δ denotes a residual. From a control engineering perspective, (3.110) is an integrator. It updates $\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$ until the residual δ approaches zero.

Monte Carlo methods
The principle of Monte Carlo (MC) methods is to sample rollouts $\boldsymbol{\tau}$ using the current policy π and approximate the expectation $Q^{\pi}(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\pi}\{\mathsf{R}_k \mid \mathbf{x}_k = \mathbf{x}, \mathbf{u}_k = \mathbf{u}\}$ to compute the empirical mean return[6]

$$\hat{R}_k = \sum_{j=k}^{N} \gamma^{j-k} r_j \tag{3.111}$$

for each state-input pair. Then, with residual

$$\delta = \frac{1}{N(\mathbf{x}, \mathbf{u})} \sum_{k=1}^{N(\mathbf{x},\mathbf{u})} \left[\hat{R}_k - \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \right] , \tag{3.112}$$

[6] Also referred to as Monte Carlo return.

where $N(\mathbf{x}, \mathbf{u})$ is the total number of times the state-input pair is visited, we get from (3.110) the Monte Carlo update rule

$$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) + \frac{1}{N(\mathbf{x}, \mathbf{u})} \sum_{k=1}^{N(\mathbf{x},\mathbf{u})} \left[\hat{R}_k - \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \right] . \tag{3.113}$$

An update law for the visitation count $N(\mathbf{x}, \mathbf{u})$ is mathematically formulated as

$$N(\mathbf{x}, \mathbf{u}) \leftarrow N(\mathbf{x}, \mathbf{u}) + \mathbb{1}_{\mathbf{x}_k=\mathbf{x}, \mathbf{u}_k=\mathbf{u}} .$$

Here, $\mathbb{1}_{\mathbf{x}_k=\mathbf{x}, \mathbf{u}_k=\mathbf{u}}$ is the binary indicator function[7]. The equation (3.113) represents a simple average of the returns. After each episode, for each state-input pair visited, we calculate the return $\hat{R}_k$ from that point until the end of the episode and then update the Q-value estimate for that state-action pair accordingly. This particular MC method is called every-visit MC because we compute the return every time a state is visited in the episode. MC methods need to learn from complete episodes to compute and all the episodes must eventually terminate.

SARSA: On-policy Temporal-Difference Learning
On-policy Temporal-Difference (TD) learning combines the principles of Monte Carlo methods with Dynamic Programming (DP). Under the certainty equivalence assumption, we get from the Bellman Expectation Equation, cf. (3.50),

$$Q^{\pi}(\mathbf{x}, \mathbf{u}) = \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma Q^{\pi}(\mathbf{x}', \mathbf{u}') . \tag{3.114}$$

The key idea in On-policy TD learning is to update the action-value function $\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$ using the TD residual

$$\delta = \bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \hat{Q}^{\pi}(\mathbf{x}', \mathbf{u}') - \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \tag{3.115}$$

and to follow the SARSA update rule, see, e.g., [3.6, p. 120],

$$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) + \alpha \left[\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \hat{Q}^{\pi}(\mathbf{x}', \mathbf{u}') - \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \right] . \tag{3.116}$$

For $\alpha = 0$, the value $\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$ remains unchanged, while for $\alpha = 1$, the old value $\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u})$ is replaced entirely by the so-called TD-target[8] $\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \hat{Q}^{\pi}(\mathbf{x}', \mathbf{u}')$. This becomes evident when we express (3.116) as a first-order low-pass filter in the form

$$\hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) \leftarrow (1 - \alpha) \hat{Q}^{\pi}(\mathbf{x}, \mathbf{u}) + \alpha \left[\bar{r}(\mathbf{x}, \mathbf{u}) + \gamma \hat{Q}^{\pi}(\mathbf{x}', \mathbf{u}') \right] . \tag{3.117}$$

On-policy Temporal-Difference learning is called SARSA because of this data sequence used for calculation – State, Action, Reward, State, Action. The estimate $\hat{Q}^{\pi}$ is updated based on its own estimation, which is a form of bootstrapping, as described in [3.6, p. 89]. TD learning can learn from incomplete rollouts and hence we don't need to track the

[7] If $\mathbf{x}_k = \mathbf{x}$ and $\mathbf{u}_k = \mathbf{u}$, the function $\mathbb{1}_{\mathbf{x}_k=\mathbf{x}, \mathbf{u}_k=\mathbf{u}}$ returns 1, otherwise, it returns 0.
[8] In machine learning, a target value refers to the actual value you aim to predict with your model. It's the outcome or label that the algorithm is being trained to forecast in supervised learning.

rollouts up to termination.

Q-learning: Off-policy Temporal-Difference Learning
Q-learning is an extension of TD-learning that goes beyond predicting the value function Q^π to enable the determination of an optimal policy π^*. The update rule for Q-learning involves updating the estimate $\hat{Q}^\pi$ at each time step, considering both the observed state $\mathbf{x}$ and the corresponding action $\mathbf{u}$. Note that the Bellman Optimality Equations (3.53) reads as

$$Q^*(\mathbf{x},\mathbf{u}) = \bar{r}_k(\mathbf{x}_k,\mathbf{u}_k) + \gamma \mathbb{E}_{\mathbf{x}_{k+1}\sim \mathsf{p}_\mathsf{x}(\mathbf{x}_{k+1}|\mathbf{x}_k,\mathbf{u}_k)} \left\{ \max_{\mathbf{w}\in\mathcal{U}} Q^*(\mathbf{x}',\mathbf{w}) \right\} . \tag{3.118}$$

Since we do not have access to the optimal values Q^*, the idea in Off-policy TD learning is to update the estimate of the action-value function $\hat{Q}^\pi(\mathbf{x},\mathbf{u})$ using the residual

$$\delta = \bar{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^\pi(\mathbf{x}',\mathbf{w}) - \hat{Q}^\pi(\mathbf{x},\mathbf{u}) . \tag{3.119}$$

The Q-learning update law than becomes

$$\hat{Q}^\pi(\mathbf{x},\mathbf{u}) \leftarrow \hat{Q}^\pi(\mathbf{x},\mathbf{u}) + \alpha \left[\bar{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^\pi(\mathbf{x}',\mathbf{w}) - \hat{Q}^\pi(\mathbf{x},\mathbf{u}) \right] . \tag{3.120}$$

The estimated action-value function $\hat{Q}^\pi$ directly approximates optimal action-value function Q^*, independent of the policy being followed. There are a couple of extensions of TD- and Q-learning that are designed to address its limitations and improve its applicability and performance. Key extensions include:

- Double Q-Learning [3.6, p. 125]: Addresses the overestimation of action-state values by decoupling the improvement and evaluation of the input in the action-state value update.
- Eligibility Traces [3.6, p. 287]: The more theoretical view is that Eligibility Traces are a bridge from TD to Monte Carlo methods (forward view). According to the other view, an Eligibility Trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action (backward view). The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. TD(λ) [3.6, p. 293]: Extends the basic TD method by considering a series of TD errors for all time steps until the end of the episode, weighted by a factor λ. The factor $\lambda \in [0,1]$ allows you to balance the trade-off between bias (accuracy) and variance (stability) in the learning updates.
- $Q(\lambda)$ [3.6, p. 312]: This extension applies the concept of Eligibility Traces to Q-Learning. In $Q(\lambda)$, every input-value pair (Q-value) has an associated eligibility trace, which decays over time but gets bumped up when visited. The action-state value updates are then applied to all pairs proportionally to their eligibility, allowing for quicker propagation of information through the state-input space.

Finally, we compare Monte Carlo, SARSA and Q-Learning and draw some conclusions:

Aspect	**Monte Carlo**	**SARSA**	**Q-Learning**
Type of Learning	Episodic	On-policy TD	Off-policy TD
Convergence Speed	Slow (high variance)	Moderate	Fast (high bias)
Policy Dependency	Optimal policy	Sensitive to current policy	Learns optimal policy from any policy
Suitability	Episodic tasks	Close to current strategy tasks	Complex environments with suboptimal actions

Table 3.5: Comparison of Monte Carlo, SARSA, and Q-Learning.

Bemerkung 3.8. Python code of the *Monte Carlo Algorithm*, the *SARSA Algorithm* and *Q-Learning Algorithm* for the Frozen Lake Example can be found here.

3.4.2 Approximate solution methods

For large state and input spaces, methods where the state value function V^π or the action-state value function Q^π are represented by means of a table are no longer manageable. The reason for this is the high memory requirements of the table, since a values must be stored for each state or state-input pair. Moreover, for a continuous state space, an infinite number of values would have to be stored, and the computational cost is considerable has to be applied to each state or state-input pair, respectively. Instead of a table, a function approximator parameterized with the vector $\boldsymbol{\phi}$ can be used to represent the value functions in the form

$$V^\pi(\mathbf{x}) \approx \hat{V}^\pi(\mathbf{x};\boldsymbol{\phi}) = \hat{V}^\pi_{\boldsymbol{\phi}}(\mathbf{x}) \tag{3.121a}$$

$$Q^\pi(\mathbf{x},\mathbf{u}) \approx \hat{Q}^\pi(\mathbf{x},\mathbf{u};\boldsymbol{\phi}) = \hat{Q}^\pi_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{u})\ . \tag{3.121b}$$

As a result, the memory requirement is significantly reduced to only the parameter vector $\boldsymbol{\phi}$. Additionally, by employing a function approximator, the agent can generalize from previously visited states or state-input pairs to unvisited ones. In an ideal scenario where $V^\pi(\mathbf{x})$ and $Q^\pi(\mathbf{x},\mathbf{u})$ are known, one could utilize supervised learning methods to approximate them with $\hat{V}^\pi$ and $\hat{Q}^\pi$. By generating rollouts

$$\boldsymbol{\tau}^{(e)} = \left(\mathbf{x}_0^{(e)}, \mathbf{u}_0^{(e)}, \mathbf{x}_1^{(e)}, \mathbf{u}_1^{(e)}, \ldots, \mathbf{x}_{N-1}^{(e)}, \mathbf{u}_{N-1}^{(e)}, \mathbf{x}_N^{(e)}\right) \tag{3.122}$$

for $e = 0, 1, \ldots, E$, with data

$$\mathcal{D} = \left\{\mathbf{x}_k^{(i)}, V^\pi\left(\mathbf{x}_k^{(e)}\right)\right\}_{e=0}^{E} \tag{3.123a}$$

$$\mathcal{D} = \left\{\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right), Q^\pi\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right)\right\}_{e=0}^{E}\ , \tag{3.123b}$$

the *regression task* can be formulated as follows

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{e\in\mathcal{D}} \left(\hat{V}^\pi_{\boldsymbol{\phi}}\left(\mathbf{x}_k^{(e)}\right) - V^\pi\left(\mathbf{x}_k^{(e)}\right)\right)^2 \tag{3.124}$$

and

$$\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}|} \sum_{e\in\mathcal{D}} \left(\hat{Q}^\pi_{\boldsymbol{\phi}}\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right) - Q^\pi\left(\mathbf{x}_k^{(e)}, \mathbf{u}_k^{(e)}\right)\right)^2\ . \tag{3.125}$$

3.4.3 Deep Q-Network

Q-learning can experience instability and divergence when combined with nonlinear function approximators and bootstrapping. The Deep Q-Network (DQN) addresses these issues by introducing two key innovations to stabilize and enhance the training process:

- *Experience replay*: Instead of using sequential rollouts for updates, DQN stores a collection of steps $\mathbf{e}_k = (\mathbf{x}_k, \mathbf{u}_k, \mathsf{r}_k, \mathbf{x}_{k+1})$ in a replay memory $\mathcal{D}_k = \{\mathbf{e}_0, \ldots, \mathbf{e}_k\}$. This memory contains a diverse range of experiences from multiple past episodes. During training, mini-batches of experiences are randomly sampled from this memory

to update the Q-values. This technique not only improves the efficiency of data utilization but also breaks the correlation between consecutive samples and mitigates the non-stationary distribution issues, leading to more stable and reliable learning.

- *Periodically updated the target network*: In standard Q-learning, the same network estimates both the current and the target Q-values, leading to a moving target problem that can cause harmful correlations and oscillations. DQN addresses this by employing two separate networks: a primary network with parameters $\boldsymbol{\phi}$ for the current Q-value estimation and a target network with parameters $\boldsymbol{\phi}^-$ that remains fixed for a set number of steps C. The target network's sole purpose is to generate the stable target Q-values for the updates. Every C steps, the primary network's weights are copied to the target network, ensuring that the targets are only periodically updated, which significantly enhances the stability of the learning process.

The objective function for DQN is formulated as follows:

$$L(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{x},\mathbf{u},\mathsf{r},\mathbf{x}')\sim U(\mathcal{D})}\left[\left(\mathsf{r}(\mathbf{x},\mathbf{u}) + \gamma \max_{\mathbf{w}\in\mathcal{U}} \hat{Q}^{\pi}_{\boldsymbol{\phi}^-}(\mathbf{x}',\mathbf{w}) - \hat{Q}^{\pi}_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{u})\right)^2\right] \tag{3.126}$$

Here, $U(\mathcal{D})$ represents a uniform distribution over the replay memory $\mathcal{D}$ and $\boldsymbol{\phi}^-$ denotes the parameters of the target network.

3.4.4 Policy Gradients

Policy gradients are a class of algorithms in reinforcement learning that optimize policies directly. They work by calculating the gradient of the expected reward concerning the policy parameters and then adjusting the parameters in the direction that increases the expected reward. This approach is particularly powerful for high-dimensional or continuous input spaces and allows for stochastic policies, offering a more nuanced way of exploring and exploiting the environment. They form the basis for many advanced reinforcement learning methods and are fundamental to understanding and developing new algorithms in the field.

The term *Policy Gradient* (PG) covers methods where a policy is parameterized by the parameter vector $\boldsymbol{\theta}$, i.e.

$$\pi(\mathbf{u}_k \mid \mathbf{x}_k;\boldsymbol{\theta}) = \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \ . \tag{3.127}$$

With this parameterized policy, a multivariate distribution

$$\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0)\prod_{k=0}^{N-1} \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k,\mathbf{u}_k) \tag{3.128}$$

can be introduced, which gives the probability density of observing a N-step rollout $\boldsymbol{\tau}$ under a the policy $\pi_{\boldsymbol{\theta}}$. Thus, with the *discounted return* (3.8), we obtain the parameterized action-value function

$$Q^{\pi_\theta}_k(\mathbf{x}_k,\mathbf{u}_k) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\mathsf{R}_k(\boldsymbol{\tau}) \mid \mathbf{x}_k = \mathbf{x}_k, \mathbf{u}_k = \mathbf{u}_k\} \ , \tag{3.129}$$

and analogously the parameterized state-value function

$$V_k^{\pi_\theta}(\mathbf{x}_k) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\mathsf{R}_k(\boldsymbol{\tau}) \mid \mathbf{x}_k = \mathbf{x}_k\} \ . \tag{3.130}$$

Now, the optimal control problem can be formulated as an optimization over the parameter vector $\boldsymbol{\theta}$, see [3.8, p. 96], in the form

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \ , \tag{3.131}$$

with expected return

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)}\{V_0^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_0)\} = \int_{\mathcal{T}} \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\mathsf{R}_0(\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau} \ . \tag{3.132}$$

To solve (3.131) with (3.132), a gradient ascent method can be employed

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}} \ , \tag{3.133}$$

where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ represents the gradient of the objective function $J(\boldsymbol{\theta})$, and α denotes the step size or learning rate. Policy gradient methods optimize a policy directly by following the gradient of the expected reward with respect to policy parameters. The distinction between Deterministic and Stochastic Policy Gradients is in the way inputs are chosen by the policy:

- Deterministic Policy Gradients (DPG): The policy directly maps states to inputs deterministically. In Deep Deterministic Policy Gradient (DDPG), a noise process is added to the output of the actor network to promote exploration during learning. This controlled randomness in the output enables the deterministic policy to explore the action space effectively.
- Stochastic Policy Gradients (SPG): The policy outputs a probability distribution over inputs, meaning the input is sampled from this distribution, introducing randomness. SPG approaches are often advantageous in settings where exploration is essential, as the stochasticity encourages diverse input selection in the learning process.

3.4.5 Deterministic and stochastic policies

Deterministic policy
For a deterministic policy, we have

$$\mathbf{u} = \boldsymbol{\mu}_\theta(\mathbf{x}) \ . \tag{3.134}$$

The policy $\boldsymbol{\mu}_\theta(\mathbf{x})$ is typically represented by neural networks or other parameterized function approximators. These functions map from the state space to the input space and are designed to capture complex, nonlinear relationships between states and inputs.
Multivariate Gaussian distribution
A multivariate Gaussian distribution (or multivariate normal distribution) is described by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. A diagonal Gaussian distribution is a special case where the covariance matrix only has entries on the diagonal. As a result, we can represent it by a vector. Hence, a popular choice for policy model $\pi_{\boldsymbol{\theta}}(\mathbf{u} \mid \mathbf{x})$ is the *diagonal*

Gaussian policy model. A diagonal Gaussian policy always has a neural network that maps from states to mean inputs $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviations are typically represented by standalone parameters. Given the mean input $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$, and a vector $\mathbf{z}$ of noise from a spherical Gaussian ($\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$), an input sample can be computed with

$$\mathbf{u} = \boldsymbol{\mu}_{\theta}(\mathbf{x}) + \boldsymbol{\sigma}_{\theta} \odot \mathbf{z} \ , \tag{3.135}$$

where $\odot$ denotes the element wise product of two vectors. In a diagonal Gaussian policy model, exploration is achieved by the additive noise term. The mean determines the expected input, while the diagonal elements of the covariance matrix determine the amount of exploration or noise added to each input. When an input is sampled from this distribution, the noise leads to exploration by encouraging slightly different inputs to be taken, even in the same state. This process allows the model to explore various strategies and potentially discover better ones. Note that the log-likelihood of a m-dimensional input, for a diagonal Gaussian with mean $\boldsymbol{\mu} = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma} = \boldsymbol{\sigma}_{\boldsymbol{\theta}}$, is given by

$$\ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u} \mid \mathbf{x})\right) = -\frac{1}{2}\left(\sum_{i=1}^{m}\left(\frac{(\mathbf{u}[i] - \boldsymbol{\mu}[i])^2}{\boldsymbol{\sigma}[i]^2} + 2\ln\left(\boldsymbol{\sigma}[i]\right)\right) + m\ln\left(2\pi\right)\right) . \tag{3.136}$$

3.4.6 Stochastic Policy Gradient

To obtain the Policy Gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ for the stochastic policy (3.127), the following procedure is performed. From (3.132), follows

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \int_{\mathcal{T}} \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}} \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \ . \tag{3.137}$$

Using the identity[9]

$$\nabla_{\boldsymbol{\theta}} \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \frac{\nabla_{\boldsymbol{\theta}} \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right)}{\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})} = \mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \ , \tag{3.138}$$

we obtain

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}} \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \{ \nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) \mathsf{R}_0(\boldsymbol{\tau}) \} \ . \tag{3.139}$$

The term $\ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right)$ can be split into three components using (3.128), i.e.,

$$\ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \ln\left(\mathsf{p}_0(\mathbf{x}_0)\right) + \sum_{k=0}^{N-1} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) + \sum_{k=0}^{N-1} \ln\left(\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)\right) \ . \tag{3.140}$$

Taking the gradient with respect to the parameter vector eliminates the terms that depend on the system dynamics results in

$$\nabla_{\boldsymbol{\theta}} \ln\left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})\right) = \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) = \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) \ . \tag{3.141}$$

[9] Known as the log-derivative-trick.

Substituting (3.141) into (3.139) with (3.129) yields the *Stochastic Policy Gradient* (SPG) for finite time horizons[10], see [3.14],

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \mathsf{R}_0 \right\} . \tag{3.142}$$

Bemerkung 3.9. With a parameterized deterministic policy $\pi_{\boldsymbol{\theta}} : \mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$, the multivariate distribution (3.155) simplifies to

$$\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.143}$$

Taking the logarithm yields

$$\ln \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \right) = \ln \left(\mathsf{p}_0(\mathbf{x}_0) \right) + \sum_{k=0}^{N-1} \ln \left(\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \right) \Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.144}$$

The gradient with respect to $\boldsymbol{\theta}$ is computes as

$$\begin{aligned} \nabla_{\theta} \ln \left(\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \right) &= \nabla_{\boldsymbol{\theta}} \sum_{k=0}^{N-1} \ln(\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k)) \Big|_{\boldsymbol{\theta}(\mathbf{x}_k)} , \\ &= \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \ln \left(\mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \right) \big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} , \end{aligned} \tag{3.145}$$

where the term containing the system dynamics does not cancel out anymore, unlike in (3.141). Thus, the system dynamics must then be known to calculate the policy gradient.

There are two important variants of the Stochastic Policy Gradient:

- *Action-Value Function Policy Gradient*: Incorporates all return information, making it theoretically complete and accurate.
- *Advantage Function Policy Gradient*: Reduces variance in gradient estimates, leading to more stable and efficient learning.

Action-Value Function Policy Gradient

Firstly, the Stochastic Policy Gradient can be formulated in term of the return R_k according to (3.8). Causality requires that future inputs and policies at time k do not depend on past rewards at time i. Thus, for $i < k$, we have

$$0 = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \{ \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \mathbf{r}_i \} . \tag{3.146}$$

[10] $\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}$ is the so-called score function.

With the definition of the return R_k, we get from (3.142)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \left(\sum_{i=0}^{k-1} \gamma^{i-k} r_i + \underbrace{\sum_{i=k}^{N-1} \gamma^{i-k} r_i}_{=\mathsf{R}_k} \right) \right\} \tag{3.147}$$

and with the causality condition (3.146), we find another formulation of the Stochastic Policy Gradient

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \mathsf{R}_k \right\} . \tag{3.148}$$

Secondly, the Stochastic Policy Gradient can be written in term of the state-value function $Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)$. Define $\boldsymbol{\tau}_{[:,k]} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_k, \mathbf{u}_k)$ as the rollout up to time k, and $\boldsymbol{\tau}_{[k,:]}$ as the remainder of the rollout after that. Using the *law of total expectation* (**??**), we can break up (3.142) into:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{k=0}^{N-1} \mathbb{E}_{\boldsymbol{\tau}_{[:,k]} \sim \pi_{\boldsymbol{\theta}}} \left\{ \mathbb{E}_{\boldsymbol{\tau}_{[k,:]} \sim \pi_{\boldsymbol{\theta}}} \left\{ \nabla_{\theta} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \mathsf{R}_k \mid \boldsymbol{\tau}_{[:,k]} \right\} \right\} . \tag{3.149}$$

The gradient of the log-probability is constant with respect to the inner expectation, due to its dependency on $\mathbf{x}_k$ and $\mathbf{u}_k$, and it can be extracted:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{k=0}^{N-1} \mathbb{E}_{\boldsymbol{\tau}_{[:,k]} \sim \pi_{\boldsymbol{\theta}}} \left\{ \nabla_{\theta} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) \mathbb{E}_{\boldsymbol{\tau}_{[k,:]} \sim \pi_{\boldsymbol{\theta}}} \left\{ \mathsf{R}_k \mid \boldsymbol{\tau}_{[:,k]} \right\} \right\} . \tag{3.150}$$

Lastly, the Markov property implies

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) \right\} . \tag{3.151}$$

Since $\mathsf{p}^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{\tau})$ is unknown, the expectation $\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\cdot\}$ is approximated from $e = 1, \dots E$ rollouts $\boldsymbol{\tau}^{(e)}$, collected in the set $\mathcal{D} = \{\boldsymbol{\tau}^{(e)}\}$, by the *empirical mean* as

$$\hat{\mathbf{g}} = \widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{\tau} \in \mathcal{D}} \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{x}_k \mid \mathbf{u}_k) \right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) . \tag{3.152}$$

Advantage Function Policy Gradient

The estimated policy gradient has high variance[11], resulting in poor convergence properties [3.15]. To reduce variance, a *baseline* $b_k(\mathbf{x}_k) \in \mathbb{R}$ can be subtracted from Q_k^{π} in (3.142) without affecting the expectation, see [3.8, p. 98] for a proof, resulting in

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \big(\ln \pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \big) \Big(Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) - b_k(\mathbf{x}_k) \Big) \right\} . \tag{3.153}$$

[11] We say that a method has high variance if you get different results when you run it multiple times. A method with less variance will give you more similar results when you run it multiple times.

Intuitively, the variance is reduced when subtracting a baseline because the operation effectively re-centers the reward signal around a mean value. A widely used baseline is the value function $b_k(\mathbf{x}_k) = V_k^{\pi}(\mathbf{x}_k)$. See [3.16] for different baselines and variants of the Stochastic Policy Gradient. From (3.153), we then get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln \left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k) \right) A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) \right\} , \tag{3.154}$$

with *Advantage function* according to (3.41). The Advantage function quantifies the relative benefit of taking a specific input in a given state compared to the average input for that state.

- By comparing $Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)$ (expected reward for a specific input) with the basline $V_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k)$ (average reward under the policy), the advantage function isolates how much better or worse an input is relative to this baseline.
- Inputs with a positive advantage $A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) > 0$ are better than the policy's average input, and the policy is encouraged to select these inputs more frequently.
- Inputs with a negative advantage $A_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) < 0$ are worse than the average, and the policy is discouraged from taking them.

3.4.7 Deterministic Policy Gradient

The policy gradient (PG) for deterministic policy of the form $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$ is developed in [3.17]. The authors demonstrated that the Deterministic Policy Gradient (DPG) acts as a special case of the Stochastic Policy Gradient (SPG). For a parameterized deterministic policy $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)$, we introduce the multivariate distribution

$$\mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) = \mathsf{p}_0(\mathbf{x}_0) \prod_{k=0}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \mathbf{u}_k) \Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} . \tag{3.155}$$

The cost function to be minimized

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)} \{ V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0) \} = \int_{\mathcal{T}} \mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \mathsf{R}_0(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \tag{3.156}$$

is differentiated with respect to $\boldsymbol{\theta}$, to obtain

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0 \sim \mathsf{p}_0(\mathbf{x}_0)} \{ \nabla_{\boldsymbol{\theta}} V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0) \} . \tag{3.157}$$

To determine the DPG, $\nabla_{\boldsymbol{\theta}} V_0^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_0)$ must be known. The calculation is then carried out for the general case $\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k)$, following the proof in [3.17]:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) &= \nabla_{\boldsymbol{\theta}} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \\ &\overset{(3.50)}{=} \nabla_{\boldsymbol{\theta}} \Big(\bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \Big\{ V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \Big\} \Big) \\ &= \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \nabla_{\boldsymbol{\theta}} \int_{\mathcal{X}} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1} , \end{aligned} \tag{3.158}$$

The last part can be expanded taking the gradient inside the integral

$$\begin{aligned}&\gamma \int_{\mathcal{X}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \\ &+ \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1} \ .\end{aligned} \tag{3.159}$$

Putting things together gives

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} \Bigg(& \bar{r}_k(\mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) + \gamma \underbrace{\int \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1}}_{= \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \left\{ V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\}} \Bigg) \\ & + \gamma \underbrace{\int \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)) \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \mathrm{d}\mathbf{x}_{k+1}}_{= \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \left\{ \nabla_{\theta} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\}} \ .\end{aligned} \tag{3.160}$$

And finally we get

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} V_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k) = \ & \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)\big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} \\ & + \gamma \mathbb{E}_{\mathbf{x}_{k+1} \sim \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{k+1} \mid \mathbf{x}_k, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k))} \left\{ \nabla_{\boldsymbol{\theta}} V_{k+1}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_{k+1}) \right\} \ .\end{aligned} \tag{3.161}$$

Continuing the recursion in (3.161), substituting this into (3.157), and combining the expectation over $\boldsymbol{\tau}$, one obtains an expression for the DPG:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \mathsf{p}^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\boldsymbol{\tau})} \left\{ \sum_{k=0}^{N-1} \gamma^k \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k) \nabla_{\mathbf{u}_k} Q_k^{\boldsymbol{\mu}_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k) \Big|_{\mathbf{u}_k = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_k)} \right\}. \tag{3.162}$$

3.4.8 Actor-Critic Methods

Two main components in policy gradient are the policy model and the value function. It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the policy update, such as by reducing gradient variance in policy gradients, and that is exactly what the Actor-Critic method does. Actor-critic methods consist of two models:

- The *Critic* updates the value function parameters $\boldsymbol{\phi}$ and depending on the algorithm it could be action-value function $Q_{\boldsymbol{\phi}}^{\pi}$ or state-value function $V_{\boldsymbol{\phi}}^{\pi}$.
- The *Actor* updates the policy parameters $\boldsymbol{\theta}$ for $\pi_{\boldsymbol{\theta}}$, in the direction suggested by the critic.

Algorithm 4 shows a vanilla Actor-Critic Stochastic Policy Gradient Algorithm and Algorithm shows a Vanilla Actor-Critic Deterministic Policy Gradient Algorithm.

Algorithm 4: Vanilla Actor-Critic Stochastic Policy Gradient Algorithm

/* Initialization */
Initialize policy parameters $\boldsymbol{\theta}$ arbitrarily;
Initialize value function parameters $\boldsymbol{\phi}$ arbitrarily;
for $e = 0, 1, 2, \ldots, E$ *episodes* **do**
  /* Reset the environment and initial state */
  Initialize state $\mathbf{x}_0$;
  /* Collect rollouts using policy $\pi^{(e)} = \pi\left(\boldsymbol{\theta}^{(e)}\right)$ in environment */
  $\mathcal{D}^{(e)} = \boldsymbol{\tau}^{(1)}, \ldots, \boldsymbol{\tau}^{(E)}$;
  for $k = 0, 1, 2, \ldots, N-1$ *time steps* **do**
    /* Compute Monte Carlo return */
    $\hat{R}_k^{(e)} \leftarrow \sum_{j=k}^{N-1} \gamma^{j-k} r_j$;
    /* Compute Advantage estimates */
    $\hat{A}_k^{(e)} \leftarrow \hat{R}_k^{(e)} - \hat{V}_{\boldsymbol{\phi}^{(e)}}^{\pi}$;
  end
  /* Estimate policy gradient */
  $\hat{\mathbf{g}}^{(e)} \leftarrow \frac{1}{\left|\mathcal{D}^{(e)}\right|} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(e)}} \hat{A}_k^{(e)}$;
  /* Actor learning - compute policy update */
  $\boldsymbol{\theta}^{(e+1)} \leftarrow \boldsymbol{\theta}^{(e)} + \alpha \hat{\mathbf{g}}^{(e)}$;
  /* Critic learning - fit the value function by regression */
  $\boldsymbol{\phi}^{(e+1)} \leftarrow \arg\min_{\boldsymbol{\phi}} \frac{1}{\left|\mathcal{D}^{(e)}\right| N} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \left(\hat{V}_{\boldsymbol{\phi}}^{\pi} - \hat{R}_k^{(e)}\right)^2$;
end

3.4.9 Off-Policy Policy Gradient

The vanilla versions of the actor-critic method are exclusively on-policy: training samples are collected according to the *target policy*, which is the same policy we aim to optimize. A simpler strategy involves two policies: the *target policy* π, which is studied and optimized, and the *behavior policy* β, which drives exploratory actions. Here, learning is based on data "off" the target policy, and the entire mechanism is labeled as off-policy learning. Off-policy methods offer several distinct advantages, including:

- The off-policy approach doesn't require a complete rollout, and it can reuse past episodes (via experience replay), resulting in significantly improved sample efficiency[12].
- The sample collection employs a behavior policy different from the target policy, facilitating more effective exploration.

[12] Sample efficiency in the context of control technology and machine learning refers to the ability of a system to achieve high performance with a limited number of data samples.

Nearly all off-policy methods employ *importance sampling* to estimate expected values from a distribution different than the sample source. In the context of off-policy learning, returns are weighted by the importance sampling ratio, which quantifies the relative likelihood of specific trajectories under the target and behavior policies. Given an initial state $\mathbf{x}_k$, the probability of a future state-input trajectory under any policy π is given by (3.33). Thus, the relative probability of the trajectory under the target and behavior policies, that it the *importance sampling ratio*, is

$$\rho_{[k:N-1]} \doteq \frac{\prod_{j=k}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{j+1} \mid \mathbf{x}_j, \mathbf{u}_j)\pi(\mathbf{u}_j \mid \mathbf{x}_j)}{\prod_{j=k}^{N-1} \mathsf{p}_{\mathsf{x}}(\mathbf{x}_{j+1} \mid \mathbf{x}_j, \mathbf{u}_j)\beta(\mathbf{u}_j \mid \mathbf{x}_j)} = \prod_{j=k}^{N-1} \frac{\pi(\mathbf{u}_j \mid \mathbf{x}_j)}{\beta(\mathbf{u}_j \mid \mathbf{x}_j)} \,. \tag{3.163}$$

Although the trajectory probabilities depend on the MDP's transition probabilities, which are generally unknown, they appear identically in both the numerator and denominator, and thus cancel. The importance sampling ratio ends up depending only on the two policies and the sequence, not on the MDP.

Remember that we aim to estimate the expected returns under the target policy π, but only possess returns R_k^β from the behavior policy β. These returns exhibit an incorrect expectation denoted by $V^\beta(\mathbf{x}) = \mathbb{E}_\beta\Big\{\mathsf{R}_k^\beta \mid \mathbf{x}_k = \mathbf{x}\Big\}$. To correct this, importance sampling is employed. Using the ratio $\rho_{[k:N-1]}$, we can transform these returns to align with the desired expectation

$$V^\pi(\mathbf{x}) = \mathbb{E}_\pi\Big\{\rho_{[k:N-1]}\mathsf{R}_k^\beta \mid \mathbf{x}_k = \mathbf{x}\Big\} \,. \tag{3.164}$$

Now let's see how off-policy policy gradient is computed. Since the training observations are sampled by $\mathbf{u}_k \sim \beta(\mathbf{u}_k \mid \mathbf{x}_k)$, we can rewrite the gradient following certain calculations, see [3.18] for a proof,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\beta\left\{\sum_{k=0}^{N-1} \frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)}{\beta(\mathbf{u}_k \mid \mathbf{x}_k)} \nabla_{\boldsymbol{\theta}} \ln\left(\pi_{\boldsymbol{\theta}}(\mathbf{u}_k \mid \mathbf{x}_k)\right) Q_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{x}_k, \mathbf{u}_k)\right\} \,. \tag{3.165}$$

Here, $\frac{\pi_{\boldsymbol{\theta}}(\mathbf{u}_k|\mathbf{x}_k)}{\beta(\mathbf{u}_k|\mathbf{x}_k)}$ is the *importance weight*. In essence, when implementing policy gradient in the off-policy setting, we can adjust it with a weighted sum, where the weight is the ratio of the target policy to the behavior policy.

3.4.10 Proximal Policy Optimization

Schulman proposed *Proximal Policy Optimization* (PPO) in 2017, and it has since become a widely used method in continuous model-free RL due to its simplicity and strong performance. PPO-clip is an actor-critic method and makes use of a generalized advantage estimate and a clipped surrogate objective function, see [3.19]. It's worth noting that there are other variants of PPO, such as PPO-KL, which incorporate an adaptive Kullback-Leibler penalty term, see [3.16]. Our discussion will focus exclusively on PPO-Clip. This approach is straightforward to implement and has been shown to work well in practice. It doesn't require a second-order optimization process, making it computationally less intensive.

Generalized Advantage Estimation

PPO employs a Generalized Advantage Estimation (GAE) as its advantage function. The purpose of GAE is to significantly reduce the variance of the estimator while keeping the bias introduced as low as possible [3.20, p. 136]. This goal mirrors the fundamental principles of Eligibility Traces, cf. [3.6, p. 125]. Our present discussion is fundamentally grounded in the methodologies outlined in the supplementary material provided by [3.21]. Generally, the Monte Carlo return (3.111) serves as an unbiased estimator of the expected return at a specific state. However, each reward r_k may vary due to the environment dynamics, leading to a high variance in the overall estimation. To mitigate this, an employe an n-step return

$$\begin{aligned}\hat{R}_{[k:k+n]} &= r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \ldots + \gamma^n r_{k+n} \\ &= \sum_{j=k}^{n-1} \gamma^{j-k} r_j + \gamma^n \hat{V}^\pi_{k+n}(\mathbf{x}_{k+n}) \ ,\end{aligned} \tag{3.166}$$

where we estimate the remaining return using a value function estimate $\hat{V}^\pi(\mathbf{x})$. It offers a lower-variance but slightly biased estimate by truncating the sum of returns after n steps. Particular instances such as $\hat{R}_{[k:k+1]} = r_k + \gamma \hat{V}^\pi_{k+1}(\mathbf{x}_{k+1})$, used in Q-learning, and $\hat{R}_{[k,\infty]} = \sum_{j=k}^{N} \gamma^{j-k} r_j = \hat{R}_k$, which reverts to the original Monte Carlo return, illustrate the variance-bias trade-off. An alternative for balancing bias and variance is the λ-return, computed as a weighted average of n-step returns, see [3.6, p. 289], and defined as

$$\hat{R}_k(\lambda) = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{R}_{[k:k+n]} \ . \tag{3.167}$$

Assuming all rewards after step N are zero, such that $\hat{R}_{[k:k+n]} = \hat{R}_{[k:N]}$ for all $n \geq N-k$, the infinite sum can be calculated according to

$$\hat{R}_k(\lambda) = (1-\lambda) \sum_{n=1}^{N-k-1} \lambda^{n-1} \hat{R}_{[k:k+n]} + \lambda^{N-k-1} \hat{R}_{[k:N]} \ . \tag{3.168}$$

Here, $\lambda = 0$ yields $\hat{R}_{[k,k+1]}$, and $\lambda = 1$ provides the full Monte Carlo return $\hat{R}_{[k,\infty]}$. Intermediate values of $\lambda \in (0,1)$ produces interpolants that can be used to balance the bias and variance of the value estimator. Updating the value function with the λ-return leads to the so-called TD(λ) algorithm, and its application for advantage estimation results in the Generalized Advantage Estimator GAE(λ):

$$\hat{A}^\pi_k(\lambda) = \hat{R}_k(\lambda) - \hat{V}^\pi_k(\mathbf{x}_k) \ . \tag{3.169}$$

For instance, setting $\lambda = 0$ yields the expression

$$\hat{A}^\pi_k(0) = \hat{R}_{[k:k+1]} - \hat{V}^\pi_k(\mathbf{x}_k) = r_k + \gamma \hat{V}^\pi_{k+1}(\mathbf{x}_{k+1}) - \hat{V}^\pi_k(\mathbf{x}_k) = \delta_k \ , \tag{3.170}$$

which is equivalent to the TD residual (3.115). Notice that $\hat{R}_k(0) = \hat{R}_{[k:k+1]} = r_k + \gamma \hat{V}^\pi_{k+1}(\mathbf{x}_{k+1})$ is the 1-step estimator for $\hat{Q}^\pi_k(\mathbf{x}_k, \mathbf{u}_k)$. Choosing a value closer to zero

introduces more bias due to the immediate approximation, resulting in lower variance, as discussed in [3.16]. Conversely, employing $\lambda = 1$ leads to high variance and nearly zero bias due to the summation of rewards, i.e.,

$$\hat{A}_k^\pi(1) = \hat{R}_{[k:N]} - \hat{V}_k^\pi(\mathbf{x}_k) = \sum_{j=k}^{N} \gamma^{j-k} r_j \ . \tag{3.171}$$

which is equivalent to the Monte Carlo return (3.111). To summarize, adjusting the value of λ allows control over the tradeoff between bias and variance. A smaller value, such as $\lambda = 0$, reduces variance at the expense of introducing more bias, while a larger value, like $\lambda = 1$, results in higher variance with minimal bias due to reward summation.

Clipping

PPO-clip updates policies via

$$\theta^{(i+1)} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{u}_k \sim \pi_{\boldsymbol{\theta}^{(i)}}} \left\{ L(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}) \right\} \ , \tag{3.172}$$

typically taking multiple steps of (usually mini-batch) Stochastic Gradient Ascent (SGA) to maximize the objective function. The (clipped surrogate) objective function L is given by, see [3.19],

$$L\Big(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}\Big) = \min \left(l_k A^{\pi_{\boldsymbol{\theta}^{(i)}}}(\mathbf{x}_k, \mathbf{u}_k),\ \mathrm{clip}(l_k, \epsilon) A^{\pi_{\boldsymbol{\theta}^{(i)}}}(\mathbf{x}_k, \mathbf{u}_k) \right) \ , \tag{3.173}$$

with likelihood ratio

$$l_k = l_k\Big(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}\Big) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)}{\pi_{\boldsymbol{\theta}^{(i)}}(\mathbf{x}_k, \mathbf{u}_k)} = \frac{\text{new policy}}{\text{old policy}} \tag{3.174}$$

and clipping operator

$$\mathrm{clip}(l_k, \epsilon) = \begin{cases} 1 - \epsilon & \text{if} \quad l_k < 1 - \epsilon \\ 1 + \epsilon & \text{if} \quad l_k > 1 + \epsilon \\ l_k & \text{else} \ . \end{cases} \tag{3.175}$$

The hyperparameter ϵ is a small positive constant, typically set to a value like 0.2, see Figure 3.10 for an illustration. The purpose of this ϵ-clipping is to prevent overly large policy updates, thereby maintaining stability in the learning process. This is achieved by ensuring that the current policy does not deviate excessively from the older one. The term l_k represents the probability ratio between the current and old policy. Let's interpret the implications of l_k:

- If $l_k > 1$, it signifies that for a given state $\mathbf{x}_k$, the corresponding action $\mathbf{u}_k$ is more probable under the current policy than it was under the old policy.
- Conversely, if l_k is between 0 and 1, the action $\mathbf{u}_k$ is less likely under the current policy compared to the old one.

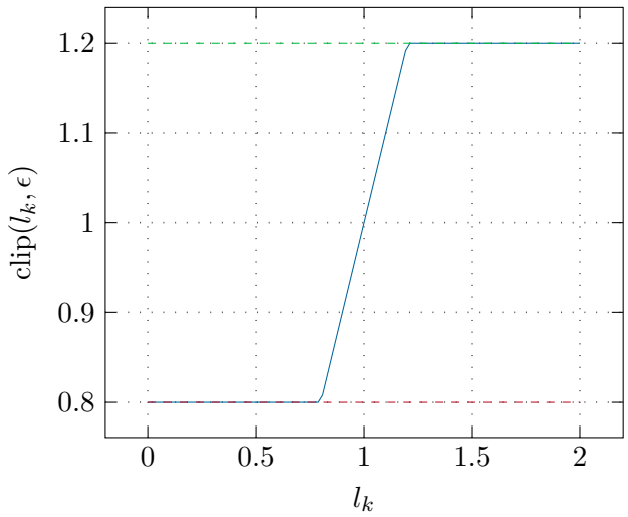


Figure 3.10: Clipping function (3.175) with $\epsilon = 0.2$.

This likelihood ratio serves as a straightforward method to gauge the divergence between the old and current policy. Algorithm 5 shows vanilla Proximal Policy Optimization with Clipping.

Algorithm 5: Proximal Policy Optimization with Clipping

`/* Initialization */`

Initialize policy parameters $\boldsymbol{\theta}$, old policy parameters $\boldsymbol{\theta}_{\text{old}}$;

Initialize value function parameters $\boldsymbol{\phi}$;

for $e = 1, 2, 3, \ldots$ *episodes* **do**

`/* Collect rollouts using the old policy` $\pi^{(e)} = \pi\left(\boldsymbol{\theta}^{(e)}\right)$ `in environment */`

$\mathcal{D}^{(e)} = \{\boldsymbol{\tau}^{(1)}, \ldots, \boldsymbol{\tau}^{(E)}\}$;

for $k = 0, 1, 2, \ldots, N-1$ *time steps* **do**

`/* Compute` λ`-returns */`

$\hat{R}_k^{(e)}(\lambda) \leftarrow (1-\lambda) \sum_{n=1}^{N-k-1} \lambda^{n-1} \hat{R}_{[k:k+n]} + \lambda^{N-k-1} \hat{R}_{[k:N]}$;

`/* Compute generalized advantage estimates */`

$\hat{A}_k^{(e)} \leftarrow \hat{R}_k^{(e)}(\lambda) - \hat{V}_{\boldsymbol{\phi}}^{\pi}(\mathbf{x}_k)$;

end

`/* Actor learning - update policy parameters via SGA in mini-batch */`

$l_k^{(e)} \leftarrow= \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_k, \mathbf{u}_k)}{\pi_{\boldsymbol{\theta}^{(e)}}(\mathbf{x}_k, \mathbf{u}_k)}$;

$\boldsymbol{\theta}^{(e+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} \frac{1}{\left|\mathcal{D}^{(e)}\right|} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \min\left(l_k^{(e)} \hat{A}_k^{(e)},\ \text{clip}\left(\epsilon, l_k^{(e)}\right) \hat{A}_k^{(e)}\right)$;

`/* Critic learning - fit the value function by regression */`

$\boldsymbol{\phi}^{(e+1)} \leftarrow \arg\min_{\boldsymbol{\phi}} \frac{1}{\left|\mathcal{D}^{(e)}\right| N} \sum_{\boldsymbol{\tau} \in \mathcal{D}^{(e)}} \sum_{k=0}^{N-1} \left(\hat{V}_{\boldsymbol{\phi}}^{\pi}(\mathbf{x}_k) - \hat{R}_k^{(e)}(\lambda)\right)^2$;

end

3.5 Literatur

[3.1] M. P. Deisenroth, A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020. [Online]. Available: https://mml-book.github.io/.

[3.2] E. T. Jaynes, *Probability Theory: The Logic of Science*. Cambridge University Press, 2013, vol. 1.

[3.3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2005, vol. 1.

[3.4] M. L. Puterman, *Markov Decision Processes*. John Wiley & Sons, 2005.

[3.5] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2009.

[3.6] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 2018.

[3.7] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[3.8] M. Sugiyama, *Statistical Reinforcment Learning*. CRC Press, 2015.

[3.9] D. Silver, *Reinforcement learning: An introduction*, 2015. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[3.10] F. L. Lewis, D. L. Varbie, and V. Syrmos, *Optimal Control*. John Wiley & Sons, 2012.

[3.11] M. Vidyasagar, "A tutorial introduction to reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2304.00803v1.

[3.12] R. Stengel, *Optimal Control and Estimation*. Dover Publications, 1986.

[3.13] J. Schulman, "Optimizing expectations: From deep reinforcement learning to stochatic computation graphs," Ph.D. dissertation, University of California, Berkeley, 2016.

[3.14] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, pp. 1238–1274, 2013.

[3.15] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[3.16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *ICLR 2015*, 2015. [Online]. Available: arXiv:1506.02438.

[3.17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmille, "Deterministic policy gradient algorithms," in *Proceedings ofthe 31th International Conference on Machine Learning, ICML 32*, 2014.

[3.18] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *Proceedings of the 29 th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

[3.19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: arXiv:1707.06347.

[3.20] L. Graesser and W. Keng, *Foundation of Deep Reinforcement Learning*. Addison-Wesley, 2020.

[3.21] X. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," 2018. [Online]. Available: arXiv:1804.02717.

4 Imitation learning

4.1 Problem Formulation

Similar to the RL problems in previous sections, we assume the system is a Markov Decision Process (MDP) with a state $\mathbf{x}$ and control input $\mathbf{u}$ belongs to the set of admissible states and controls $\mathcal{X}$ and $\mathcal{U}$, respectively. The system dynamics are expressed by the probabilistic transition model:

$$\mathsf{p}(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1}), \tag{4.1}$$

The goal is to define a policy π that defines the closed-loop control law in the form

$$\mathbf{u}_t = \pi(\mathbf{x}_t)\,. \tag{4.2}$$

Here, the key difference in the problem formulation is that it does not require the reward function $\mathsf{r}_t = r(\mathbf{x}_t, \mathbf{u}_t)$. Instead, a set of expert demonstrations, where each demonstration $\boldsymbol{\tau}$ consists of a sequence of state-control pairs,

$$\boldsymbol{\tau} = \{\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \dots\}\,, \tag{4.3}$$

is given. Note that (4.3) are taken from the expert policy π^*. The goal of the imitation learning process is to determine a policy $\hat{\pi}$ that imitates the expert policy π^*.

Definition 4.1. For a system with transition model (4.1), the imitation learning problem is to leverage a set of demonstrations $\mathcal{D} = \{\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_D\}$ from an expert policy π^* to find a policy $\hat{\pi}$ that imitates the expert policy.

Imitation learning methods are typically classified into two main categories: behavior cloning (BC) and inverse reinforcement learning (IRL). Behavior cloning utilizes supervised learning to directly learn a policy from the data. On the other hand, inverse reinforcement learning, also known as inverse optimal control, focuses on modeling a reward function from the data. Once this reward function is obtained, conventional optimal control methods can be applied to build a policy. Inverse RL aims to generalize the policy beyond the demonstrated examples.

When deciding between Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL), it's crucial to determine the most effective way to characterize the desired behavior. The policy generated through an IRL approach remains valid as long as the estimated reward function accurately reflects the intended behavior. Conversely, a policy derived from a BC method is considered valid as long as the mapping from states to actions is correct. The choice between BC and IRL ultimately revolves around the best representation of the desired behavior, which varies based on the specific context of the problem. Thus, a careful

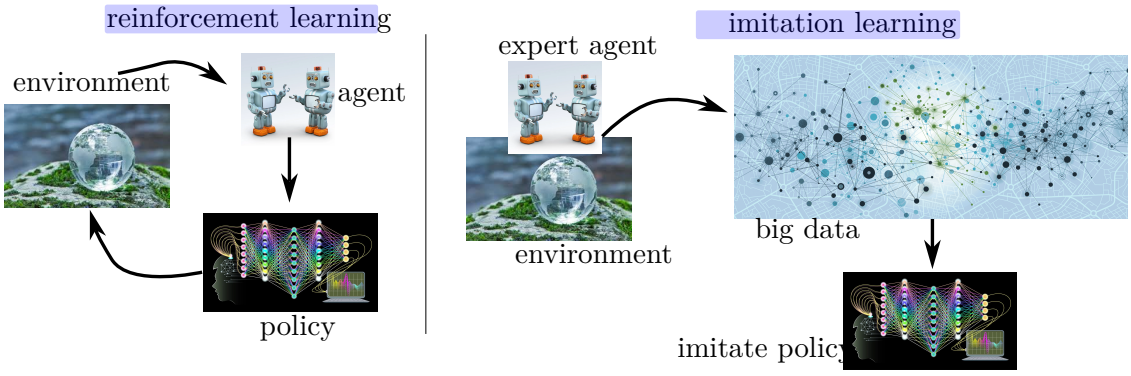


Figure 4.1: Simple illustrations of reinforcement learning and imitation learning.

analysis of how the desired behavior should be executed is essential when implementing imitation learning techniques. Next, we will dive into (also recap) an overview of policy representations in reinforcement/imitation learning.

4.2 Policy Representation

4.2.1 Policy Abstraction.

In imitation learning, we can categorize the policy representations into three main types:

1. Task-level abstraction: The agent learns the policy

$$\pi : \mathbf{x}_t \to [o_1, \ldots, o_T] \,, \tag{4.4}$$

   following option $o \in \mathcal{O}$ where $\mathcal{O}$ is the set of options. Note that options contain a set of actions or trajectories over a period of time T.

2. Trajectory-level abstraction: The agent learns the policy mapping the initial state $\mathbf{x}_0$ to a trajectory $\boldsymbol{\tau}$ as

$$\pi : \mathbf{x}_0 \to \boldsymbol{\tau} \,, \tag{4.5}$$

3. Action-state space abstraction. The agent learns the policy mapping the system state $\mathbf{x}_t$ and the action $\mathbf{u}_t$ in the form

$$\pi : \mathbf{x}_t \to \mathbf{u}_t \,. \tag{4.6}$$

4.2.2 Deterministic Policy

A deterministic policy in trajectory planning abstraction yields a specific trajectory $\boldsymbol{\tau}$ for a given initial state $\mathbf{x}_0$

$$\boldsymbol{\tau} = \pi(\mathbf{x}_0). \tag{4.7}$$

In an action-state space abstraction, a deterministic policy computes a control action $\mathbf{u}$ for a given state $\mathbf{x}$

$$\mathbf{u}_t = \pi(\mathbf{x}_t)\,. \tag{4.8}$$

When a deterministic policy is used, the distribution of the trajectory $\boldsymbol{\tau}$ is expressed as:

$$\mathsf{p}(\boldsymbol{\tau}) = \mathsf{p}(\mathbf{x}_0)\prod_{t=1}^{T}\mathsf{p}(\mathbf{x}_{t+1}|\mathbf{x}_t, \pi(\mathbf{x}_t)). \tag{4.9}$$

4.2.3 Stochastic Policy

A stochastic policy in action-state space samples a control input u from a probability distribution given the state x

$$\mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x}). \tag{4.10}$$

Here, π is the conditional distribution of the control input $\mathbf{u}$ given $\mathbf{x}$. The distribution of the trajectory $\boldsymbol{\tau} = [\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_T, \mathbf{u}_T]$ is expressed as:

$$\mathsf{p}(\boldsymbol{\tau}) = \mathsf{p}(\mathbf{x}_0)\prod_{t=1}^{T}\mathsf{p}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)\pi(\mathbf{u}_t|\mathbf{x}_t). \tag{4.11}$$

Stochastic policies are useful when modeling the randomness of expert behavior.

4.2.4 Trajectory Feature Expectation

To match the behavior between the expert and the agent over a long horizon, we consider trajectory feature matching which describes the behavior of the expert and the learner The expectation of the trajectory features with respect to the agent's policy is in the form

$$\mathbb{E}_{\mathsf{p}(\boldsymbol{\tau})}[\phi(\boldsymbol{\tau})] = \int \mathsf{p}(\boldsymbol{\tau})\phi(\boldsymbol{\tau})d\boldsymbol{\tau} \tag{4.12}$$

where $\mathsf{p}(\boldsymbol{\tau})$ is the trajectory distribution yielded by the agent policy and $\phi(\boldsymbol{\tau})$ is the feature vector of the trajectory $\boldsymbol{\tau}$. When a dataset of trajectories $\mathcal{D} = \{\boldsymbol{\tau}_i\}_{i=1}^{D}$ is available, the expectation of the trajectory feature can be approximated by

$$\mathbb{E}_{\mathsf{p}(\boldsymbol{\tau})}[\phi(\boldsymbol{\tau})] \approx \frac{1}{D}\sum_{i=1}^{D}\phi(\boldsymbol{\tau}_i)\,. \tag{4.13}$$

4.2.5 Feature Matching in Imitation Learning

Imitation learning is formulated as a problem of finding a policy that minimizes the difference between demonstrated and learned behavior. For this purpose, imitation learning methods project demonstrated behavior onto a manifold of a parameterized policy. Therefore, the relationship between the distribution of the demonstrations and the distribution of the parameterized policy is taken into account in the following. Consider a trajectory distribution $\mathsf{p}(\boldsymbol{\tau}|\mathbf{w})$ induced by a policy π with a parameter vector $\mathbf{w}$. Supervised

learning methods often obtain a solution based on the maximum likelihood of the given training data. Note that maximizing the (causal) likelihood is equivalent to minimizing the KL divergence [4.1]

$$\mathsf{D}_{\mathrm{KL}}(\mathsf{q}(\boldsymbol{\tau})||\mathsf{p}(\boldsymbol{\tau}|\mathbf{w})) = \int \mathsf{q}(\boldsymbol{\tau}) \ln \frac{\mathsf{q}(\boldsymbol{\tau})}{\mathsf{p}(\boldsymbol{\tau}|w)} d\boldsymbol{\tau} = \mathbb{E}_{\mathsf{q}}[\ln \mathsf{q}(\boldsymbol{\tau})] - \mathbb{E}_{\mathsf{q}}[\ln \mathsf{p}(\boldsymbol{\tau}|\mathbf{w})] \tag{4.14}$$

where $q(\boldsymbol{\tau})$ is the empirical distribution over trajectories induced by the expert's policy. Note that $\mathbb{E}_{\mathsf{q}}[\cdot]$ is the expectation with respect to $\mathsf{q}(\boldsymbol{\tau})$ [4.2]. The expectations $\mathbb{E}_{\mathsf{q}}$ in (4.14) can be estimated using the demonstrated trajectories drawn from $\mathsf{q}(\boldsymbol{\tau})$. Here, the goal of imitation learning is to learn a parameter vector $\mathbf{w}$ in the form

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathsf{D}_{\mathrm{KL}}(\mathsf{q}(\boldsymbol{\tau})||\mathsf{p}(\boldsymbol{\tau}|\mathbf{w})) \,. \tag{4.15}$$

Since the first term in (4.14) is independent of $\mathbf{w}$, (4.14) can be minimized by maximizing the expected log-likelihood $\mathbb{E}_{\mathsf{q}}[\ln \mathsf{p}(\boldsymbol{\tau}|\mathbf{w})]$.

Now, we take a look at imitation learning in action-state space from an information-theoretic point of view. Taking into account the stochastic distribution of the trajectory in (4.11) and assuming that the representation of the learner and the expert are equivalent and stationary, i.e., $\mathsf{q}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathsf{p}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathsf{p}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, the relation of $\mathsf{p}(\boldsymbol{\tau})$ and $\mathsf{q}(\boldsymbol{\tau})$ is given by

$$\frac{\mathsf{p}(\boldsymbol{\tau})}{\mathsf{q}(\boldsymbol{\tau})} = \frac{\prod_{t=0}^{T} \pi_L(\mathbf{u}_t|\mathbf{x}_t)}{\prod_{t=0}^{T} \pi^*(\mathbf{u}_t|\mathbf{x}_t)} \tag{4.16}$$

where π_L is the learner's policy and π^* is the expert's policy. In this case, (4.14) is written as

$$\begin{aligned}
\mathsf{D}_{\mathrm{KL}}(\mathsf{q}(\boldsymbol{\tau})||\mathsf{p}(\boldsymbol{\tau})) &= \int \mathsf{q}(\boldsymbol{\tau}) \sum_{t=0}^{T} \ln \frac{\pi^*(\mathbf{u}_t|\mathbf{x}_t)}{\pi_L(\mathbf{u}_t|\mathbf{x}_t)} \mathrm{d}\boldsymbol{\tau} && \text{(4.17a)} \\
&= \int \mathsf{q}(\mathbf{x}, \mathbf{u}) \ln \frac{\pi^*(\mathbf{u}|\mathbf{x})}{\pi_L(\mathbf{u}|\mathbf{x})} \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{u} && \text{(4.17b)} \\
&= \mathbb{E}_{\mathsf{q}(\mathbf{x},\mathbf{u})}\Big[\ln \pi^*(\mathbf{u}|\mathbf{x}) - \ln \pi_L(\mathbf{u}|\mathbf{x}) \Big] \,, && \text{(4.17c)}
\end{aligned}$$

where $\mathsf{q}(x, u)$ is the state-action distribution induced by the trajectory distribution $\mathsf{q}(\boldsymbol{\tau})$ of the expert. Since $\mathbb{E}_q[\cdot]$ can be approximated using the trajectories drawn from $\mathsf{q}(\boldsymbol{\tau})$, minimization of the KL divergence in (4.17) can be solved using only the demonstrated trajectories.

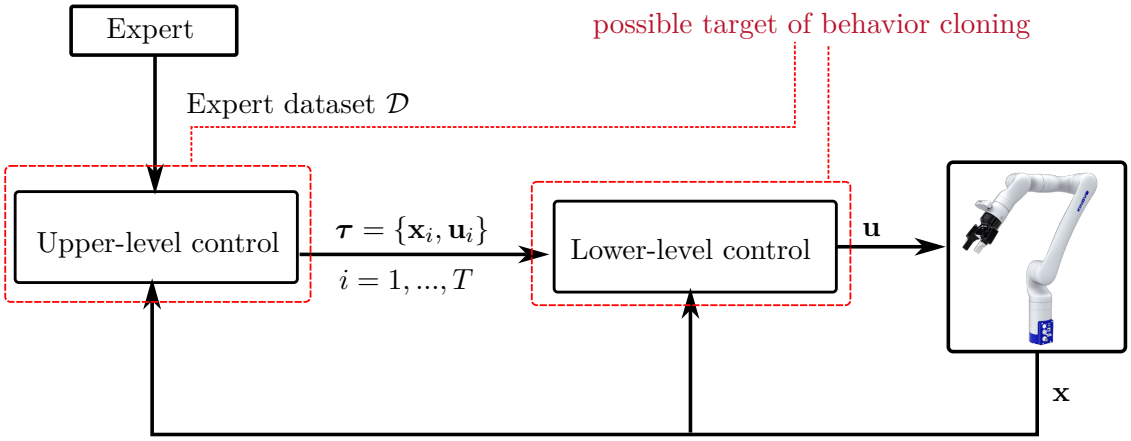


Figure 4.2: Diagram of a robotic system with possible target of imitation learning methods. The upper-level control can be a trajectory planner generating the desired trajectories for the low-level control. Imitation learning can be utilized to learn the upper-level control, i.e., as trajectory-level abstraction, or the low-level control, i.e., as action-state space abstraction

.

4.3 Behavior Cloning

In this section, we take a closer look at the set of behavior cloning (BC) methods. In Figure 4.2, an example of imitation learning in a control diagram of a robotic system is illustrated. Imitation learning can be utilized as a trajectory-level abstraction in the upper-level control or as an action-state space abstraction in the lower-level control.

Behavior cloning approaches use a set of expert demonstrations $\boldsymbol{\tau}_j \in \mathcal{D}$, $j \in \{1, ..., D\}$ to determine a policy π that imitates the expert. This is done using supervised learning techniques, where the difference between the learned policy and the expert demonstrations is minimized with respect to some metric. The objective is to solve the following optimization problem:

$$\pi_L = \arg\min_{\pi} \sum_{\boldsymbol{\tau} \in \mathcal{D}} \sum_{\mathbf{x} \in \boldsymbol{\tau}} C(\pi(\mathbf{x}), \pi^*(\mathbf{x})), \tag{4.18}$$

where C is the cost function, e.g., Euclidean norm or Kullback-Leibler divergence in (4.14), $\pi^*(\mathbf{x})$ is the expert's action at state $\mathbf{x}$, and π_L is the approximated (learner) policy.

Although behavior cloning provides a straightforward approach to imitation learning, it may not yield good performance since the learning process is only based on a set of samples provided by the expert. In many cases, these expert demonstrations will not be uniformly sampled across the entire state space. Thus, the learned policy will perform poorly when operating in states far from those seen in $\boldsymbol{\tau}$. This leads to an accumulation of errors when executing a policy obtained by behavior cloning. Here, a question is raised as follows: "If we can achieve low error by mimicking the expert on the training data, what guarantee can we provide on the total cost of the learned policy when used to perform

the task?" [4.3].

4.3.1 Imitation as Supervised Learning

In the following, we will answer the above question given a model-free behavior cloning problem. The following notations are considered.

- p_π^t is the distribution of states at time t if the agent followed policy π from time step 1 to $t-1$
- $\mathsf{p}_\pi = \frac{1}{T}\sum_{t=1}^{T} \mathsf{p}_\pi^t$ is the average distribution fo the states if the policy π is executed for T steps.

Given a state $\mathbf{x}$, we denote the immediate expected cost of performing action $\mathbf{u}$ at state $\mathbf{x}$ in the form

$$C_\pi(\mathbf{x}) = \mathbb{E}_{a\sim\pi(\mathbf{x})}[C(\mathbf{x},\mathbf{u})]\,, \tag{4.19}$$

which is bounded in $[0, C_{max}]$. Hence, the cost-to-go for executing $T-$steps is expressed as

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\pi^t}[C_\pi(\mathbf{x})] = T\mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\pi}[C_\pi(\mathbf{x})]\,. \tag{4.20}$$

Supposing that we obtain a policy π with the expected loss

$$\epsilon = \mathbb{E}_{\mathbf{x}\sim\mathsf{p}_{\pi^*},\mathbf{u}\sim\pi^*(\mathbf{x})}[l(\mathbf{x},\mathbf{u},\pi)]\ . \tag{4.21}$$

For simplicity, we assume that l is the 0-1 loss in the following

$$l(\mathbf{x},\mathbf{u},\pi) := \begin{cases} 1\,, & \mathbf{u} \neq \boldsymbol{\pi}_E(\mathbf{x}) \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.22}$$

This leads to the following performance guarantee with respect to any task cost $C \in [0,\ C_{max}]$

Theorem 1. Let ϵ follows (4.21), L upper bounds the 0-1 loss, then, see also [4.3]:

$$J(\pi) \leq J(\pi^*) + C_{max}T^2\epsilon \tag{4.23}$$

Intuitively, the quadratic growth in T arises because, once $\hat{\pi}$ makes an error, it may encounter states that π^* never visited, subsequently incurring the maximum cost of 1 at each subsequent step.

Proof. We consider

$$\epsilon_\pi^i = \mathbb{E}_{\mathbf{x}\sim\mathsf{p}_\pi^i,\mathbf{u}\sim\pi(\mathbf{x})}[l(\mathbf{x},\mathbf{u},\pi^*)],\ i = 1,...,T \tag{4.24}$$

is the probability π makes a mistake under the distribution of $\mathsf{p}_{\pi^*}^i$,and $\epsilon_\pi = \frac{1}{T}\epsilon_\pi^i$ is the average error of the policy π.

Let q_t indicate the probability when the learned policy π picked the same action as the expert π^* in the first t steps and vice versa for $\overline{q_t}$. Then, the distribution of states at time step t while executing the policy π can be expressed as

$$\mathsf{p}_\pi^t = q_{t-1}\mathsf{p}_\pi^{t,(c)} + (1-q_{t-1})\mathsf{p}_\pi^{t,(e)} \,, \tag{4.25}$$

where $\mathsf{p}_\pi^{t,(c)}$ denotes the distribution of states executed the policy π having always picked the correct action (i.e., actions match with a decision from the expert π^*) and vice versa for $\mathsf{p}_\pi^{t,(e)}$. Similarly, we can express the distribution of states at time step t while executing the expert policy in the following

$$\mathsf{p}_{\pi^*}^t = q_{t-1}\mathsf{p}_\pi^{t,(c)} + (1-q_{t-1})\mathsf{p}_{\pi^*}^{t(e)} \,. \tag{4.26}$$

Note that $\mathsf{p}_\pi^{t,(c)} = \mathsf{p}_{\pi,E}^{t,(c)}$ since they both presents the distribution of states when π exactly mathes π^*.

We now consider $\epsilon_\pi^{t,(c)}$ and $\epsilon_{\pi^*}^{t,(c)}$ are the probability that π decides a different action $\mathbf{u}$ than π^* in the state distributions $\mathsf{p}_\pi^{t,(c)}$ and $\mathsf{p}_{\pi^*}^{t(e)}$. Then, the probability that π selects a different action than π^* is

$$\epsilon_\pi^t = q_{t-1}\epsilon_\pi^{t,(c)} + \underbrace{(1-q_{t-1})\epsilon_{\pi^*}^{t,(e)}}_{\geq 0} \geq q_{t-1}\epsilon_\pi^{t,(c)} \,. \tag{4.27}$$

Additionally, the probability the learner makes at least 1 mistake in the first t step is

$$1-q_t = (1-q_{t-1}) + \underbrace{q_{t-1}\epsilon_\pi^{t,(c)}}_{\leq \epsilon_\pi^t} \leq (1-q_{t-1}) + \epsilon_\pi^t = \underbrace{1-q_0}_{=0} + \sum_{i=1}^{t}\epsilon_\pi^t \tag{4.28}$$

Similarly, let C_π^t, $C_\pi^{t,(c)}$, and $C_\pi^{t,(e)}$ are the immediate cost of executing policy π with the state distribution p_π^t, $\mathsf{p}_\pi^{t,(c)}$, and $\mathsf{p}_\pi^{t,(e)}$, respectively. Note that the probability of the policy π selects a different action than π^* is $\epsilon_\pi^{t,(c)}$, therefore, the inequality

$$C_\pi^{t,(c)} \leq C_{\pi^*}^{t,(c)} + \epsilon_\pi^{t,(c)} C_{max} \tag{4.29}$$

hold true. Then, we have

$$\begin{aligned} C_\pi^t &= q_{t-1}C_\pi^{t,(c)} + (1-q_{t-1})C_\pi^{t,(e)} \\ &\leq q_{t-1}C_{\pi^*}^{t,(c)} + q_{t-1}\epsilon_\pi^{t,(c)}C_{max} + (1-q_{t-1})C_{max} \\ &= \underbrace{q_{t-1}C_{\pi^*}^{t,(c)}}_{\leq C_{\pi^*}^t} + \underbrace{\left[q_{t-1}\epsilon_\pi^{t,(c)} + (1-q_{t-1})\right]}_{=1-q_t} C_{max} \\ &\leq C_{\pi^*}^t + C_{max}\sum_{i=1}^{t}\epsilon_\pi^t \,. \end{aligned} \tag{4.30}$$

Summing over T steps, we obtain the following inequality

$$\begin{aligned} J(\pi) &\leq J(\pi^*) + C_{max} \sum_{t=1}^{T} \sum_{i=1}^{t} \epsilon_\pi^t \\ &= J(\pi^*) + C_{max} \sum_{t=1}^{T} (T+1-t)\epsilon_\pi^t \\ &\leq J(\pi^*) + C_{max} T \sum_{t=1}^{T} \epsilon_\pi^t =\leq J(\pi^*) + C_{max} T^2 \epsilon_\pi \end{aligned} \tag{4.31}$$

□

The resulting error can be shown to be of order $\mathcal{O}(\epsilon T^2)$, where ϵ is the probability of making a mistake in every time step and T is the horizon length, c.f. [4.3].

4.3.2 DAgger: Dataset Aggregation

To overcome the distributional mismatch between the expert policy mentioned above, the learned policy is designed to continuously collect new expert data. Specifically, once the learned policy leads to states not covered by the expert data, the expert is requested to provide additional information via simulation and/or real-world demonstration. This idea is the core of the Dataset Aggregation (DAgger) algorithm, introduced by [4.4]. This method can be seen as an on-policy approach [4.5] where the expert agent guides the learner to the correct action to take; however, the input distribution of examples is taken from the learner's behavior.

Figure 4.3 introduces an overview of the DAgger algorithm. The most simple Dagger is in the following. In the first iteration, the learner policy $\pi_{L,1}$ is initialized by behavior cloning only expert data, i.e., via regression methods. Then, we use this policy to collect a new set of trajectories (also labeling the quality of this trajectory by using the expert policy). The newly obtained trajectory $\mathcal{D}_1$ and the expert trajectory are joined into the aggregated dataset $\mathcal{D}$. Therefore, the next policy π_{n+1} can be seen as a policy mimicking the expert on the whole dataset $\mathcal{D}$. It is worth noting that to leverage the expert policy, the DAgger algorithm employs a stochastic mixing of expert and learner

$$\pi_i = \beta_i \pi^* + (1-\beta_i)\pi_{L,i} \tag{4.32}$$

to roll out the next dataset. Algorithm 6 presents the details of the DAgger algorithm. In general, by combining expert demonstrations based on the states that the learner encounters, DAGGER addresses the discrepancy between the state distribution generated by the learner's policy and the state distribution of the original demonstration data. This strategy significantly minimizes the amount of training data needed to achieve sufficient performance levels and often leads to improved results over time, as shown in [4.4]. DAGGER can be seen as transforming imitation learning into a supervised learning problem with interaction.

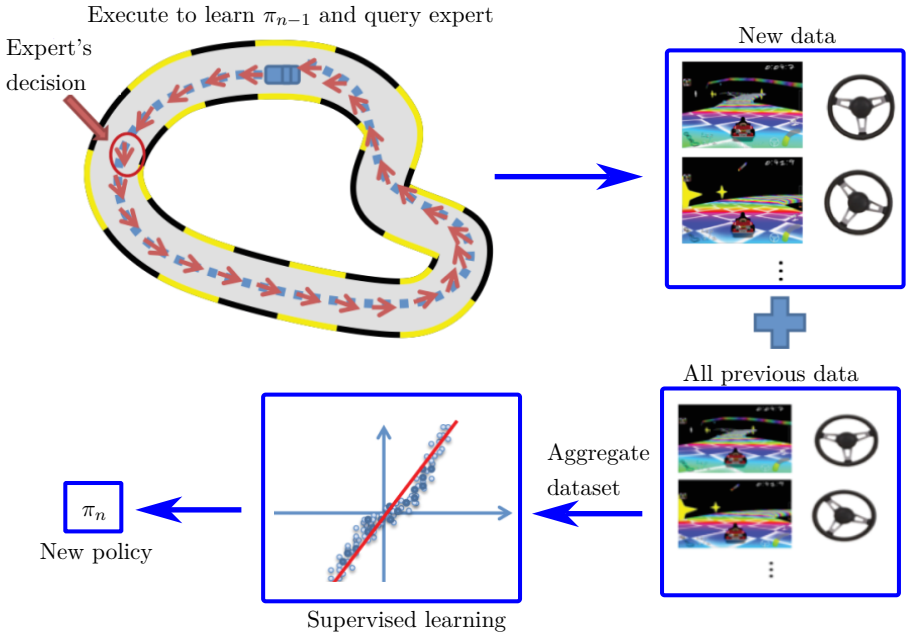


Figure 4.3: An overview of one iteration of the DAGGER algorithm [4.6]. The algorithm begins by initializing the demonstration dataset with a single collection of expert examples. Then, it optimizes the policy and generates new data to expand the dataset.

Algorithm 6: DAgger: Dataset Aggregation [4.4]

```
   /* Initialization                                                    */
1  Initialize dataset of demonstrations 𝒟 ← 𝒟_demo;
2  Initialize π_{L,1};
3  for i = 1 to N do
       /* Roll out the mixed policy to sample trajectory τ             */
4      Roll out policy π_i = β_i π* + (1 − β_i) π_{L,i} to sample trajectory τ = {x_0, x_1, ...};
       /* Query expert to label data from the trajectory               */
5      Generate dataset D_i = {(x_0, π*(x_0)), (x_1, π*(x_1)), ...} from expert;
       /* Aggregate datasets                                           */
6      D ← D ∪ D_i;
       /* Retrain policy using the aggregated dataset                  */
7      Retrain policy π_{L,i+1} using aggregated dataset D;
8  end
9  return π_{L,i} on validation;
```

4.4 Inverse Reinforcement Learning

Although behavior cloning provides a direct way to imitate the expert's actions, it does not take into account the reasons why selecting those actions. To address this, *Inverse*

Reinforcement learning	**Inverse reinforcement learning**
Given:	**Given:**
States $\mathbf{x} \in \mathcal{S}$, actions $\mathbf{u} \in \mathcal{A}$	States $\mathbf{x} \in \mathcal{S}$, actions $\mathbf{u} \in \mathcal{A}$
Transitions $\mathsf{p}(\mathbf{u}'\|\mathbf{x}, \mathbf{u})$	Transitions $\mathsf{p}(\mathbf{u}'\|\mathbf{u}, \mathbf{x})$
Reward function $r(\mathbf{x}, \mathbf{u})$	Samples $\{\boldsymbol{\tau}_i\}$ sampled from $\pi^*(\boldsymbol{\tau})$
Learn: $\pi^*(\mathbf{u}\|\mathbf{x})$	**Learn:** $r_{\mathbf{w}}(\mathbf{x}, \mathbf{u})$ *($\mathbf{w}$ is the vector of reward parameters)*

Table 4.1: Simplified differences between RL and IRL methods

Reinforcement Learning (IRL) aims to recover the reward function from demonstrations of policy. This approach allows the agent to understand the expert's intent expressed in the reward function, which generalizes better than simply mimicking their actions. Table 4.1 recaps the basis difference between RL and IRL.

In IRL, it is assumed that the expert's behavior is optimal with respect to some unknown reward function $r(\mathbf{x}, \mathbf{u})$. The goal is to optimize for this reward function such that the expert's policy π^* maximizes the expected return. A common form of the reward function is a linear combination of features [4.7]

$$r(\mathbf{x}, \mathbf{u}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}, \mathbf{u}), \tag{4.33}$$

where $\mathbf{w}$ is a weight vector, and $\phi(\mathbf{x}, \mathbf{u})$ is a feature vector representing relevant aspects of the state-action pair $(\mathbf{x}, \mathbf{u})$. Note that the reward function can also be parameterized via a neural network parameterized by the weight parameters $r_{\mathbf{w}}(\mathbf{x}, \mathbf{u})$. IRL aims to find the weight vector $\mathbf{w}$ that explains the expert's actions via maximizing the reward function. The expected reward of a policy π is given by

$$\mathbb{E}[R|\pi] = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(\mathbf{x}_t, \mathbf{u}_t) \middle| \pi\right] = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_t, \mathbf{u}_t) \middle| \right] \tag{4.34a}$$

$$= \boldsymbol{\omega}^{\mathrm{T}} \underbrace{\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \boldsymbol{\phi}(\mathbf{x}_t, \mathbf{u}_t) \middle| \pi\right]}_{\boldsymbol{\mu}(\pi)} \tag{4.34b}$$

$$= \boldsymbol{\omega}^T \boldsymbol{\mu}(\pi)\,. \tag{4.34c}$$

By the definition of the expert policy π^*, the following inequality

$$\mathbb{E}[R^*|\pi^*] \geq \mathbb{E}[R|\pi] \iff \mathbf{w}^{*\top} \boldsymbol{\mu}(\pi^*, \mathbf{x}) \geq \mathbf{w}^{\top} \boldsymbol{\mu}(\pi, \mathbf{x}) \quad \forall \pi \tag{4.35}$$

hold true. Note that identifying $\mathbf{w}^*$ associated with the expert policy can be done by finding a $\mathbf{w}$ satisfying the condition (4.35). However, maximizing the reward function in this case is often underconstrained, as multiple reward functions can explain the expert's behavior. For example, the choice of $\mathbf{w} = \mathbf{0}$ trivially satisfies (4.35). This issue is known as *reward ambiguity*, and various approaches have been proposed to resolve this ambiguity,

such as Apprenticeship Learning [4.7] and Maximum Entropy IRL [4.8]. In the next subsections, IRL algorithms are presented to avoid this issue.

Apprenticeship Learning

Apprenticeship Learning builds upon inverse reinforcement learning by focusing on matching the expert's feature expectations rather than recovering the exact reward function. The idea is that the policy π should match the expert's feature expectations $\mu(\pi^*, \mathbf{x})$, even if the exact reward function is not known. Thus, we have the following expression,

$$|\boldsymbol{\mu}(\pi, \mathbf{x}) - \boldsymbol{\mu}(\pi^*)| \leq \epsilon \Rightarrow |\mathbf{w}^{\mathrm{T}}\boldsymbol{\mu}(\pi, \mathbf{x}) - \mathbf{w}^{\mathrm{T}}\boldsymbol{\mu}(\pi^*)| \leq \epsilon, \ \forall \mathbf{w} \text{ with } ||\mathbf{w}|| \leq 1. \tag{4.36}$$

Algorithm 7 outlines the procedure for apprenticeship learning. By iteratively improving the policy, apprenticeship learning guarantees that the learned policy will match the expert's performance, even in cases where the reward function is ambiguous.

Algorithm 7: Apprenticeship Learning

```
   /* Input                                                          */
1  Dataset of the demonstrations 𝒟, termination threshold ϵ
   /* Initialization                                                 */
2  Compute μ* using 𝒟; Randomly initialize policy π_0;
3  Perform rollouts and for simplified notion, we set μ_0 = μ(π_0)
4  for i = 1, ..., N do
       /* Solve the optimization problem                            */
5      Compute
                  ε̂_i = max_w min_{j∈{0,...,i−1}} w^T (μ* − μ_j)        (4.37)
       /* Compute feature expectations                              */
6      Compute feature expectations μ(π_i) = w^T φ(x, u);
7      Compute μ_i = μ(π_i)
8      if t_i ≤ ϵ then
           /* Return best policy if convergence condition is met    */
9          return best policy π̂ from {π_0, ..., π_i};
10     end
11 end
```

4.4.1 Maximum Margin Planning

The Maximum Margin Planning (MMP) [4.9] approach uses an optimization-based solver to compute the reward function weight $\mathbf{w}$ for the problem similar to (4.37), however with some additional terms (margin terms). The idea behind the MMP method is to find a cost function in which the cost of the demonstrated trajectory $\mathcal{C}_{\boldsymbol{\tau}^*}$ is lower than the cost of any trajectory. This inequality constraint can be expressed as

$$\mathcal{C}(\boldsymbol{\tau}^*) \leq \min\{\mathcal{C}(\boldsymbol{\tau}) - L(\boldsymbol{\tau})\}, \tag{4.38}$$

with $L(\boldsymbol{\tau})$ is the loss function.

If we assume that the cost function is linear to the features of the trajectory $\boldsymbol{\tau}$ and moreover, this trajectory feature is linear to the state-action $\boldsymbol{\mu}$ defined in the previous subsection, we have the following expression

$$\mathcal{C}(\boldsymbol{\tau}) = \mathbf{w}^{\mathrm{T}} \underbrace{\boldsymbol{\phi}(\boldsymbol{\tau})}_{\mathbf{F}\boldsymbol{\mu}} = \mathbf{w}^{\mathrm{T}}\mathbf{F}\boldsymbol{\mu} \tag{4.39}$$

with $\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{X}||\mathcal{U}|}$ and $F \in \mathbb{R}^{d\times|\mathcal{X}||\mathcal{U}|}$ is the feature matrix of d-dimension. In the same way, we can assume that the loss function is linear to $\boldsymbol{\mu}$, expressed as $L(\boldsymbol{\tau}) = \mathbf{l}^{\mathrm{T}}\boldsymbol{\mu}$ with $\mathbf{l} \in \mathbb{R}^{|\mathcal{X}||\mathcal{U}|}$. Here, we can define the optimization problem in the following. Given a training set $\mathcal{D} = \{\mathbf{F}, \boldsymbol{\tau}, \mathbf{l}_i\}_{i=1}^{D}$, the optimize $\mathbf{w}$ can be formalized as a quadratic program [4.9].

$$\min_{\mathbf{w}} \mathcal{L}_{MMP} = \min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{D}\Big(\underbrace{\mathcal{C}(\boldsymbol{\tau}_i)}_{\mathbf{w}^{\mathrm{T}}\mathbf{F}_i\boldsymbol{\mu}_i} - \min_{\boldsymbol{\mu}} \{ \underbrace{\mathcal{C}(\boldsymbol{\tau}_i) - L(\boldsymbol{\tau}_i)}_{\mathbf{w}^{\mathrm{T}}\mathbf{F}_i\boldsymbol{\mu} - \mathbf{l}_i^{\mathrm{T}}\boldsymbol{\mu}} \}\Big) + \frac{\lambda}{2}||\mathbf{w}||\,, \tag{4.40}$$

with $\lambda > 0$ is the regularization parameter. The overview of the maximum margin planning is summarized in Algorithm 8.

Algorithm 8: Maximum margin planning [4.9]

```
/* Input                                                              */
```
Dataset of the demonstrations $\mathcal{D}$, $\lambda > 0$, step size $\{\alpha_j\}$, maximum iteration M

while $j < M$ **do**

  for $i = 1, \ldots, N$ **do**

    Compute the loss-augmented cost $\tilde{c}_i = \mathbf{w}^{\mathrm{T}}\mathbf{F}_i - \mathbf{l}_i^{\mathrm{T}}$

    Compute the optimal trajectory:

$$\boldsymbol{\tau}_i^* = \arg\min \tilde{c}_i\boldsymbol{\mu} \tag{4.41}$$

    Extract the optimal state-action $\boldsymbol{\mu}_i^*$ from $\boldsymbol{\tau}_i^*$

  end

  Compute the gradient $\mathbf{g} = \dfrac{\partial \mathcal{L}_{MMP}}{\partial \mathbf{w}}$

  Update the weight $\mathbf{w} \leftarrow \mathbf{w} - \alpha_j\mathbf{g}$

  $j \leftarrow j + 1$

end

```
/* Return                                                             */
```
return $\mathbf{w}$

It is obvious that the policy obtained with the MMP algorithm is based on efficient MDP solvers that generate a deterministic optimal policy in (4.41). However, stochastic policies and approximations in planning are popular for systems with high-dimensional configuration space. Therefore, the next subsection introduces the principle of maximum entropy [4.8] considering the distribution of the resulting trajectories.

4.4.2 Maximum Entropy Inverse Reinforcement Learning

While apprenticeship learning ensures that the feature expectations are matched, there can be ambiguity in the policy distributions that achieve the same feature expectations. The Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) method addresses this ambiguity by finding a distribution over trajectories that matches feature expectations while maintaining maximum entropy, i.e., the least informative distribution.

We can have the following probabilistic models of expert behavior. Let set $\mathcal{O}_t$ is the optimality can be achieved by the polity π_t. This optimality variable can have the value of either 0 or 1 when π_t returns a non-optimal solution or optimal solution, respectively. This probability can be chosen to be equal to the exponential of the reward in the following

$$\mathsf{p}(\mathcal{O}_t|\mathbf{x}_t, \pi_t(\mathbf{x}_t)) = \exp(r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t)) \tag{4.42}$$

Here, we can clearly see that with (4.42), the reward is chosen to be bounded in $(-\infty, 0]$. The backward message $\beta_t(\mathbf{x}_t, \mathbf{u}_t)$ and the forward message are defined as follow

$$\beta_t(\mathbf{x}_t, \mathbf{u}_t) = \mathsf{p}(\mathcal{O}_{t:T}|\mathbf{x}_t, \mathbf{a}_t) \tag{4.43a}$$

$$\alpha_t(\mathbf{x}_t) = \mathsf{p}(\mathbf{x}_t|\mathcal{O}_{1,t-1})\,. \tag{4.43b}$$

The backward message (4.43a) gives us the probability of being optimal now until the end of the trajectory given state and action. If we know this information, we can actually compute the policy $\mathsf{p}(\mathbf{u}_t|\mathbf{s}_t, \mathcal{O}_{1:T})$. The forward message provides the probability of landing at the state $\mathbf{x}_t$ if we follow the optimal policy till the previous time step. In the following, the computation steps for the two mentioned messages are explained in detail.

Using the chain rule, (4.43a) can be expanded as:

$$\beta_t(\mathbf{x}_t, \mathbf{u}_t) = \int \mathsf{p}(\mathcal{O}_{t:T}, \mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)\mathrm{d}\mathbf{x}_{t+1} \tag{4.44a}$$

$$= \int \underbrace{\mathsf{p}(\mathcal{O}_{t+1:T} \mid \mathbf{x}_{t+1})}_{\int \mathsf{p}(\mathcal{O}_{t+1:T}|\mathbf{u}_{t+1}, \mathbf{x}_{t+1})\mathrm{d}\mathbf{u}_{t+1}} \mathsf{p}(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)\mathsf{p}(\mathcal{O}_t \mid \mathbf{x}_t, \mathbf{u}_t)\,\mathrm{d}\mathbf{x}_{t+1} \tag{4.44b}$$

$$= \int \Big[\int \underbrace{\mathsf{p}(\mathcal{O}_{t+1:T} \mid \mathbf{u}_{t+1}, \mathbf{x}_{t+1})}_{\beta_t(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})}\,\mathrm{d}\mathbf{u}_{t+1} \Big] \mathsf{p}(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)\mathsf{p}(\mathcal{O}_t \mid \mathbf{x}_t, \mathbf{u}_t)\,\mathrm{d}\mathbf{x}_{t+1} \tag{4.44c}$$

$$= \mathsf{p}(\mathcal{O}_t \mid \mathbf{x}_t, \mathbf{u}_t)\mathbb{E}_{\mathbf{x}_{t+1} \sim \mathsf{p}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})}[\underbrace{\mathbb{E}_{\mathbf{x}_{t+1} \sim \mathsf{p}(\mathbf{x}_{t+1}|\mathbf{u}_{t+1})}[\beta_{t+1}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})]}_{\beta_{t+1}(\mathbf{x}_{t+1})}] \tag{4.44d}$$

$$= \mathsf{p}(\mathcal{O}_t \mid \mathbf{x}_t, \mathbf{u}_t)\mathbb{E}_{\mathbf{x}_{t+1} \sim \mathsf{p}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})}[\beta_{t+1}(\mathbf{x}_{t+1})]. \tag{4.44e}$$

Thus, we can compute recursive backward messages from $t = T - 1$ to 1. Note that the last value of β_T is simply the last reward of the trajectory. Additionally, we can recover the policy using this backward function in the following.

Now, a question arises, "What is the probability of a trajectory given the agent acting

optimally?"

$$\begin{aligned} \mathsf{p}(\boldsymbol{\tau}|\mathcal{O}_{1:T}) = \frac{\mathsf{p}(\mathcal{O}_{1:T}|\boldsymbol{\tau})\mathsf{p}(\boldsymbol{\tau})}{\mathsf{p}(\mathcal{O}_{1:T})} &\propto \mathsf{p}(\boldsymbol{\tau}) \prod_t \exp(r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t)) \\ &= \mathsf{p}(\boldsymbol{\tau}) \exp(\sum_t r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t)) \, . \end{aligned} \tag{4.45}$$

Our goal is to find the weight vector $\mathbf{w}$ to maximize the average over all trajectories (maximum log-likelihood learning)

$$\mathbf{w}^* = \arg\max_{\mathbf{w}}(\mathcal{L}) = \arg\max_{\mathbf{w}} \left(\frac{1}{D} \sum_{i=1}^{D} \underbrace{\log \mathsf{p}(\boldsymbol{\tau}_i|\mathcal{O}_{1:T})}_{r_{\mathbf{w}}(\boldsymbol{\tau}_i)} - \log(Z) \right) \, . \tag{4.46}$$

$\log Z$ is an entropy term (also called log normalizer) expressed in the form

$$Z = \int \mathsf{p}(\boldsymbol{\tau}) \exp(r_{\mathbf{w}}(\boldsymbol{\tau})) \mathrm{d}\boldsymbol{\tau} \tag{4.47}$$

Here, the log normalizer plays an important role in adjusting that we cannot just assign a high reward to any trajectories. Instead, this intuitively assigns the rewards that make the trajectories that we have encountered look more likely than other trajectories that will be encountered in the future.

The optimal value $\mathbf{w}$ can be found by analyzing the gradient of the loss in (4.46)

$$\begin{aligned} \nabla_{\mathbf{w}}\mathcal{L} &= \frac{1}{D} \sum_{i=1}^{D} \nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i) - \underbrace{\frac{1}{Z} \int \mathsf{p}(\boldsymbol{\tau}) \exp(r_{\mathbf{w}}(\boldsymbol{\tau}))}_{\mathsf{p}(\boldsymbol{\tau}|\mathcal{O}_{1:T}, \mathbf{w})} \nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \pi^*(\boldsymbol{\tau})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i)] - \mathbb{E}_{\boldsymbol{\tau} \sim \mathsf{p}(\boldsymbol{\tau}|\mathcal{O}_{1:T}, \mathbf{w})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau})] \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \pi^*(\boldsymbol{\tau})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i)] - \mathbb{E}_{\boldsymbol{\tau} \sim \mathsf{p}(\boldsymbol{\tau}|\mathcal{O}_{1:T}, \mathbf{w})} \left[\nabla_{\mathbf{w}} \sum_{t=1}^{T} r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t) \right] \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \pi^*(\boldsymbol{\tau})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i)] - \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{x}_t, \mathbf{u}_t) \sim \mathsf{p}(\mathbf{x}_t, \mathbf{u}_t|\mathcal{O}_{1:T}, \mathbf{w})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t)] \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \pi^*(\boldsymbol{\tau})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i)] - \sum_{t=1}^{T} \underbrace{\mathsf{p}(\mathbf{u}_t|\mathbf{x}_t, \mathcal{O}_{1:T}, \mathbf{w})}_{\beta(\mathbf{x}_t, \mathbf{u}_t)/\beta(\mathbf{x}_t)} \underbrace{\mathsf{p}(\mathbf{x}_t|\mathcal{O}_{1:T}, \mathbf{w})}_{\sim\alpha(\mathbf{x}_t)\beta(\mathbf{x}_t)} [\nabla_{\mathbf{w}} r_{\mathbf{w}}(\mathbf{x}_t, \mathbf{u}_t)] \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \pi^*(\boldsymbol{\tau})}[\nabla_{\mathbf{w}} r_{\mathbf{w}}(\boldsymbol{\tau}_i)] - \sum_{t=1}^{T} \beta(\mathbf{x}_t, \mathbf{u}_t)\alpha(\mathbf{x}_t) \nabla_{\mathbf{w}} r_{\mathbf{w}} \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\mathbf{w}} r_{\mathbf{w}}(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}) - \sum_{t=1}^{T} \beta(\mathbf{x}_t, \mathbf{u}_t)\alpha(\mathbf{x}_t) \nabla_{\mathbf{w}} r_{\mathbf{w}} \end{aligned} \tag{4.48}$$

Note that $\beta(\mathbf{x}_t, \mathbf{u}_t)\alpha(\mathbf{x}_t)$ is state-action visitation probability for each pair $(\mathbf{x}_t, \mathbf{u}_t)$.

The classic maximum entropy inverse RL algorithm is as follow, see also [4.8]

Algorithm 9: Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL)

```
   /* Input                                                          */
1  Reward parameter w, learning rate η, and demonstration dataset D
2  while not converged do
       /* Step 1: Compute backward message                          */
3      Compute backward message β(x_t, u_t)
       /* Step 2: Compute forward message                           */
4      Compute forward message α(x_t) // Refer to previous lecture for
           computation
       /* Step 3: Compute state-action visitation frequencies       */
5      β(u_t, x_t)α(x_t)
       /* Step 4: Evaluate the gradient of the likelihood           */
6
                  (1/N) Σ_{i=1}^{N} Σ_{t=1}^{T} ∇_w r_w(x_{i,t}, u_{i,t}) − Σ_{t=1}^{T} β(x_t, u_t)α(x_t)∇_w r_w
       /* Step 5: Update the reward parameter                       */
7      w ← w + η∇_w L
8  end
   /* Return                                                         */
9  return ψ
```

4.5 Conclusion

Imitation learning provides a powerful framework for learning control policies from expert demonstrations without the need for explicit reward functions. Various methods, including behavior cloning, inverse reinforcement learning, and learning from comparisons, offer different ways to approach this problem. By leveraging expert knowledge, imitation learning can produce efficient and effective policies, especially in scenarios where defining a reward function is difficult or sparse rewards make traditional reinforcement learning challenging.

4.6 Literatur

[4.1] S. K. S. Ghasemipour, R. Zemel, and S. Gu, "A divergence minimization perspective on imitation learning methods," in *Conference on robot learning*, PMLR, 2020, pp. 1259–1277.

[4.2] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[4.3] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 661–668.

[4.4] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 627–635.

[4.5] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press, 2018.

[4.6] J. A. Bagnell, "An invitation to imitation," *CMU Technical report*, 2015.

[4.7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[4.8] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *Aaai*, Chicago, IL, USA, vol. 8, 2008, pp. 1433–1438.

[4.9] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.