AUBURN UNIVERSITY
SAMUEL GINN COLLEGE OF ENGINEERING

AERO7970 Stochastic Control & Reinforcement Learning Lecture Notes

Instructor: Dr. Nan Li
Contact: nanli@auburn.edu

Contents

Module I: Review of Probability Theory

1 Basic notations and concepts

Number systems: $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
Sets: X, $\emptyset$, $E \subset X$, $P(X)$

Definition: The **power set** of X is the set of all subsets of X:

$$P(X) = \{E : E \subset X\}.$$

The elements of the power set are sets.

Exercise: If $X = \{1, 2, 3\}$, what is $P(X)$?

Operations: $\cup$, $\cap$, $X \setminus E$, E^c, $\text{card}(X)$

Definition: The **complement** of a subset E of the universal set X is the set of all elements of X that are not in E:

$$E^c = X \setminus E = \{x \in X : x \notin E\}.$$

The notation E^c is used only when the universal set X has been previously specified or is obvious and unique (i.e., there is no need to mention X in the current context).

Exercise: If $X = \{1, ..., n\}$, what is $\text{card}(P(X))$?

$\text{card}(\mathbb{N}) = \text{card}(\mathbb{Z}) = \text{card}(\mathbb{Q})$ is called **countable** (or **countably infinite**).

Definition: Two sets E and F are **disjoint** if $E \cap F = \emptyset$.

De Morgan's laws:

- $(E \cup F)^c = E^c \cap F^c$
- $(E \cap F)^c = E^c \cup F^c$

2 Measurable spaces

Definition: Let X be a nonempty set. An **algebra** on X is a nonempty collection of subsets of X that is closed under finite unions and complements. Formally, $\mathcal{A} \subset P(X)$, $\mathcal{A}$ nonempty, is an **algebra** if $\mathcal{A}$ satisfies the following two properties:

- If $E_1, ..., E_n \in \mathcal{A}$, then $\bigcup_{i=1}^n E_i \in \mathcal{A}$;
- If $E \in \mathcal{A}$, then $E^c \in \mathcal{A}$.

Exercise: Let $X = \{1, 2, 3\}$ and verify if $\mathcal{A}$ is an algebra on X:

- $\mathcal{A} = \{\emptyset, X\}$ (smallest)
- $\mathcal{A} = \{\emptyset, \{1\}, \{2\}, \{3\}, X\}$
- $\mathcal{A} = \{\emptyset, \{1\}, \{2, 3\}, X\}$
- $\mathcal{A} = P(X)$ (largest)

Exercise: Prove that if $\mathcal{A}$ is an algebra on X, then $\emptyset, X \in \mathcal{A}$.

Definition: A **σ-algebra** is an algebra $\mathcal{A}$ that is closed under countable unions, i.e.,

- If $\{E_i\}_{i=1}^{\infty} \in \mathcal{A}$, then $\bigcup_{i=1}^{\infty} E_i \in \mathcal{A}$.

Proposition: A finite algebra ($\text{card}(\mathcal{A}) < \infty$) is always a σ-algebra.

A consequence is that if X is a finite set ($\text{card}(X) < \infty$), then every algebra is a σ-algebra (since $\text{card}(\mathcal{A}) \leq \text{card}(P(X)) = 2^{\text{card}(X)}$).

Proposition: The intersection of (any family of) σ-algebras is a σ-algebra.
Exercise: Prove the above proposition.

Therefore, if we have multiple σ-algebras $\mathcal{A}_1$, ..., $\mathcal{A}_n$, we can construct a new σ-algebra as $\mathcal{A} = \bigcap_{i=1}^{n} \mathcal{A}_i$, which is smaller than (in the sense of being a subset of) any of the original σ-algebras.

Definition: Let $\mathcal{A}$ be an arbitrary collection of subsets of X. The σ-algebra **generated by** $\mathcal{A}$, denoted as $\mathcal{M}(\mathcal{A})$, is the smallest σ-algebra that contains $\mathcal{A}$, i.e.,

$$\mathcal{M}(\mathcal{A}) = \bigcap \{\mathcal{B} \subset P(X) : \mathcal{B} \text{ is a } \sigma\text{-algebra and } \mathcal{A} \subset \mathcal{B}\}$$

Exercise: Let $X = \{1, 2, 3\}$ and $\mathcal{A} = \{\emptyset, \{1\}, \{2\}, \{3\}, X\}$. What is $\mathcal{M}(\mathcal{A})$?

Definition: Let $X = \mathbb{R}$. The σ-algebra generated by the collection of open rays, $(-\infty, x)$ and (x, ∞), is called the **Borel σ-algebra**, typically denoted as $\mathcal{B}_{\mathbb{R}}$.

Exercise: Prove that the σ-algebra generated by the collection of closed rays, $(-\infty, x]$ and $[x, \infty)$, is also the Borel σ-algebra.

Definition: Let X be a set and $\mathcal{M}$ be a σ-algebra on X, then $(X, \mathcal{M})$ is called a **measurable space**, and the sets in $\mathcal{M}$ are called **measurable sets**.

3 Measures

Definition: Let $(X, \mathcal{M})$ be a measurable space. A **measure** on $(X, \mathcal{M})$ is a function $\mu : \mathcal{M} \to [0, \infty]$ that satisfies the following two properties:

- $\mu(\emptyset) = 0$;
- If $\{E_i\}_{i=1}^{\infty}$ are disjoint sets in $\mathcal{M}$, then $\mu\big(\bigcup_{i=1}^{\infty} E_i\big) = \sum_{i=1}^{\infty} \mu(E_i)$ (countable additivity).

The triple $(X, \mathcal{M}, \mu)$ is called a **measure space**.

Proposition: A measure μ on $(X, \mathcal{M})$ has the following three properties:

- If E_1, ..., E_n are disjoint sets in $\mathcal{M}$, then $\mu\big(\bigcup_{i=1}^{n} E_i\big) = \sum_{i=1}^{n} \mu(E_i)$ (finite additivity).
- If $E, F \in \mathcal{M}$ and $E \subset F$, then $\mu(E) \leq \mu(F)$ (monotonicity).
- Let $\{E_i\}_{i=1}^{\infty}$ be a collection of sets in $\mathcal{M}$, then $\mu\big(\bigcup_{i=1}^{\infty} E_i\big) \leq \sum_{i=1}^{\infty} \mu(E_i)$ (subadditivity).

Exercise: Prove the above proposition.

Definition: A **probability measure** on a measurable space $(X, \mathcal{M})$ is a measure μ on $(X, \mathcal{M})$ that satisfies $\mu(X) = 1$.

In this case, $(X, \mathcal{M}, \mu)$ is typically called a **probability space**, where X is called the **sample space**, the elements of X are called **outcomes**, $\mathcal{M}$ is called the **event space**, the sets in $\mathcal{M}$ are called **events**, and $\mu : \mathcal{M} \to [0, 1]$ is called the **probability function**.

Exercise: Prove the following basic probability rules:

- For any event E, $0 \leq \mu(E) \leq 1$;
- For any event E, $\mu(E^c) = 1 - \mu(E)$;
- For any events E and F, $\mu(E \cup F) = \mu(E) + \mu(F) - \mu(E \cap F)$.

4 Measurable functions and random variables

Definition: Let f be a mapping from X to Y. The **inverse mapping** of f, denoted as f^{-1}, is a mapping from $P(Y)$ to $P(X)$ defined by

$$f^{-1}(F) = \{x \in X : f(x) \in F\}$$

for every $F \in P(Y)$. And $f^{-1}(F)$ is called the **pre-image** of F (under f).

A "mapping" is a generalized term for "function" whose codomain may not be real numbers. In other words, a "function" is a "mapping" whose codomain is real numbers. Oftentimes, "function" and "mapping" (or "map") can be used interchangeably.

Definition: Let $(X, \mathcal{M})$ and $(Y, \mathcal{N})$ be two measurable spaces. A function $f : X \to Y$ is said to be $(\mathcal{M}, \mathcal{N})$-**measurable** (or just **measurable**) if for every $F \in \mathcal{N}$, its pre-image $f^{-1}(F)$ is in $\mathcal{M}$ (i.e., pre-images of measurable sets are measureable).

Proposition: Let $(X, \mathcal{M})$ and $(Y, \mathcal{N})$ be two measurable spaces. If $\mathcal{N}$ is the σ-algebra generated by $\mathcal{A} \subset P(Y)$, then a function $f : X \to Y$ is measurable if and only if $f^{-1}(F) \in \mathcal{M}$ for every $F \in \mathcal{A}$.

This proposition says that if we know the σ-algebra $\mathcal{N}$ is generated from $\mathcal{A}$, then to determine the measurability of f we only need to check the pre-image condition $f^{-1}(F) \in \mathcal{M}$ for every set F in $\mathcal{A}$ (i.e., we only need to check this condition for much fewer sets than all sets in $\mathcal{N}$). This result will become very useful soon.

Many functions are measurable. For instance:
Corollary: Every continuous function from $\mathbb{R}$ to $\mathbb{R}$ is $(\mathcal{B}_{\mathbb{R}}, \mathcal{B}_{\mathbb{R}})$-measurable.

Definition: Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A **random variable** is a function from the sample space Ω to $\mathbb{R}$ that is $(\mathcal{F}, \mathcal{B}_{\mathbb{R}})$-measurable. In other words, $f : \Omega \to \mathbb{R}$ is a random variable if

$$f^{-1}(E) \in \mathcal{F}, \quad \forall E \in \mathcal{B}_{\mathbb{R}}$$

Random variables enable us to study probabilities of abstract events using real numbers. From this point on, we will use capital letters (such as X, Y) to denote random variables. If not specified otherwise, we will always assume an underlying probability space $(\Omega, \mathcal{F}, \mathbb{P})$.

Corollary: A function $X : \Omega \to \mathbb{R}$ is a random variable if and only if $\{X \leq x\} = X^{-1}\big((-\infty, x]\big) \in \mathcal{F}$ for every $x \in \mathbb{R}$.
Proof: The result of Homework 1 Problem 3 + the above Proposition.

Definition: If the range of a random variable X is finite or countable, this random variable X is called a **discrete random variable**.

Note the difference between "codomain" and "range." The range of a mapping is all points of its codomain that are images of some points of its domain.

$$X(\Omega) = \{x \in \mathbb{R} : x = X(\omega) \text{ for some } \omega \in \Omega\}$$

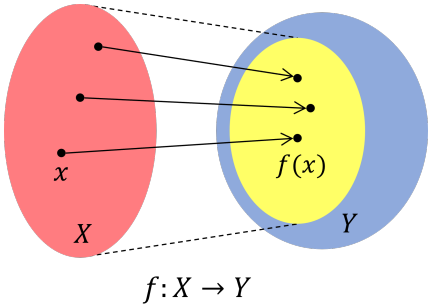

Figure 1: Codomain (blue) versus Range (yellow).

Definition: Let X be a random variable and $E \in \mathcal{B}_{\mathbb{R}}$. The probability that X (takes a value) in E is

$$\mathbb{P}(X \in E) = \mathbb{P}\big(\{\omega \in \Omega : X(\omega) \in E\}\big) = \mathbb{P}\big(X^{-1}(E)\big)$$

Random variables enable us to "push-forward" the probability measure $\mathbb{P}$ on $(\Omega, \mathcal{F})$ to a measure on $(\mathbb{R}, \mathcal{B}_{\mathbb{R}})$. This makes us sometimes "forget" about the underlying sample and event spaces. On the one hand, this enables us to focus on studying numbers. On the other hand, this may lead to some misunderstandings, e.g., "discrete random variables are only useful for studying problems with a finite (or countable) number of outcomes such as throwing a dice."

Indeed, discrete random variables can be defined on a continuous sample space and therefore can be used to study problems with quantities that take continuous values (such as $\mathbb{R}$). This lays a solid theoretical foundation for our use of Markov chains/Markov decision processes with finite state/action values to study real-world problems which typically involve continuous-valued physical quantities.

Proposition: Let X be a discrete random variable that takes a finite/countable list of values x_1, x_2, ..., x_n. Then,

$$\sum_{i=1}^{n} \mathbb{P}(X = x_i) = 1.$$

Proof:

$$\sum_{i=1}^{n} \mathbb{P}(X = x_i) = \sum_{i=1}^{n} \mathbb{P}\big(\{\omega \in \Omega : X(\omega) = x_i\}\big) = \mathbb{P}\left(\bigcup_{i=1}^{n} \{\omega \in \Omega : X(\omega) = x_i\}\right)$$
$$= \mathbb{P}\big(\{\omega \in \Omega : X(\omega) \in \{x_1, x_2, ..., x_n\}\big) = \mathbb{P}(\Omega) = 1 \qquad \blacksquare$$

Definition: Let X be a discrete random variable that takes a finite/countable list of values x_1, x_2, ..., x_n. Then, the **expected value** (also called **expectation**) of X is

$$\mathbb{E}(X) = \sum_{i=1}^{n} x_i \, \mathbb{P}(X = x_i)$$

5 Stochastic processes

Definition: A **stochastic process** is a collection of random variables defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and indexed by some set T.

In many problems, a point $t \in T$ has the meaning of time.

Notations: A stochastic process is often written as $\{X(t) : t \in T\}$, where each $X(t)$ is a random variable (i.e., a function from Ω to $\mathbb{R}$). A stochastic process can also be written as $\{X(t, \omega) : t \in T\}$ to reflect that it is actually a function of two variables, $t \in T$ and $\omega \in \Omega$.

Examples of index sets: $T = \{1, 2\}$, $T = \mathbb{Z}_+$ (discrete-time), $T = \mathbb{R}_+$ (continuous-time).

When $T = \mathbb{Z}_+$, a stochastic process can also be understood as a sequence of random variables $\{X_t\}_{t=0}^{\infty} = \{X_0, X_1, ...\}$.

Definition: A **realization** or **sample path** or **trajectory** of a stochastic process $\{X(t, \omega) : t \in T\}$ is the mapping

$$X(\cdot, \omega_0) : T \to \mathbb{R}$$

for a fixed outcome $\omega_0 \in \Omega$.

Joint probability: Given two random variables X and Y that are defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and two sets $E, F \in \mathcal{B}_{\mathbb{R}}$,

$$\mathbb{P}(X \in E, Y \in F) = \mathbb{P}\big(\{\omega \in \Omega : X(\omega) \in E \text{ and } Y(\omega) \in F\}\big) = \mathbb{P}\big(X^{-1}(E) \cap Y^{-1}(F)\big)$$

Note $\{X \in E\} = \{\omega \in \Omega : X(\omega) \in E\} = X^{-1}(E)$ and $\{Y \in F\} = \{\omega \in \Omega : Y(\omega) \in F\} = Y^{-1}(F)$ are sets/events in $\mathcal{F}$. Thus, $\{X \in E, Y \in F\} = \{\omega \in \Omega : X(\omega) \in E \text{ and } Y(\omega) \in F\} = X^{-1}(E) \cap Y^{-1}(F)$ is a set/event in $\mathcal{F}$ (by closure under finite/countable intersections). Therefore, when speaking about "joint probability," we did not introduce anything new. This "joint probability" concept extends to multiple variables (or a stochastic process) naturally.

Definition: Given two events A and B from a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with $\mathbb{P}(B) > 0$, the **conditional probability of A given B** (or the **probability of A conditioned on B**) is

$$\mathbb{P}(A \,|\, B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

The conditional probability concept extends to random variables/stochastic processes naturally:

$$\mathbb{P}(X \in E \,|\, Y \in F) = \frac{\mathbb{P}(X \in E, Y \in F)}{\mathbb{P}(Y \in F)}$$

Definition: Let X be a discrete random variable and A be an event in $\mathcal{F}$ with nonzero probability. The **conditional expectation** of X given A is

$$\mathbb{E}(X \,|\, A) = \sum_x x\, \mathbb{P}(X = x \,|\, A) = \sum_x x\, \frac{\mathbb{P}(\{X = x\} \cap A)}{\mathbb{P}(A)}$$

where the sum is taken over all possible values of X.

Module II: Markov Chains

6 Definitions and properties

Definition: Let $\{X_t\}_{t=0}^{\infty}$ be a sequence of discrete random variables. The stochastic process $\{X_t\}_{t=0}^{\infty}$ is said to have the **Markov property** if

$$\mathbb{P}(X_{t+1} = x_{t+1} \,|\, X_t = x_t, X_{t-1} = x_{t-1}, ..., X_0 = x_0) = \mathbb{P}(X_{t+1} = x_{t+1} \,|\, X_t = x_t)$$

when both conditional probabilities are well defined. In this case, the process $\{X_t\}_{t=0}^{\infty}$ is called a **Markov chain**.

The behavior of a Markov chain is determined by the **(single-step) transition probabilities**

$$\mathbb{P}(X_{t+1} = x_{t+1} \,|\, X_t = x_t)$$

for $t \in \mathbb{Z}_+$, $x_t \in \text{range}(X_t)$, and $x_{t+1} \in \text{range}(X_{t+1})$.

Definition: A Markov chain $\{X_t\}_{t=0}^{\infty}$ is said to be **time-homogeneous** (or **stationary**) if

$$\mathbb{P}(X_{t+1} = x' \,|\, X_t = x) = \mathbb{P}(X_1 = x' \,|\, X_0 = x)$$

for all t (i.e., transition probabilities are independent of t).

When time-homogeneous, the random variables $\{X_t\}_{t=0}^{\infty}$ can be thought to have the same range, called the **state space** of the chain, and $\mathbb{P}(X_{t+1} = x' \,|\, X_t = x) = \mathbb{P}(X_1 = x' \,|\, X_0 = x)$ is typically referred to as the (single-step) transition probability from **current state** x to **next state** x'.

From now on, we focus on time-homogeneous Markov chains with a finite state space $\mathcal{X} = \{x_1, x_2, ..., x_n\}$. And we use p_{ij} to denote the single-step transition probability from x_j to x_i, i.e.,

$$p_{ij} = \mathbb{P}(X_1 = x_i \,|\, X_0 = x_j)$$

The $n \times n$ matrix $\mathbf{P}$ that has p_{ij} as its entry in row i and column j is called the **(single-step) transition matrix**, typically denoted as

$$\mathbf{P} = [\, p_{ij} \,]$$

Definition: A probability distribution over the state space $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ (or, a state distribution) at time t, denoted as π_t, is a column vector whose ith entry is $\mathbb{P}(X_t = x_i)$, i.e.,

$$\pi_t = \begin{bmatrix} \mathbb{P}(X_t = x_1) \\ \mathbb{P}(X_t = x_2) \\ \vdots \\ \mathbb{P}(X_t = x_n) \end{bmatrix}$$

Examples:

- Given that the state at t is x_1, then $\mathbb{P}(X_t = x_1) = 1$ and $\pi_t = [1, 0, ..., 0]^\top$.
- Given that the probabilities for the state at t to be x_1 and to be x_2 are 0.5 and 0.5, i.e., $\mathbb{P}(X_t = x_1) = \mathbb{P}(X_t = x_2) = 0.5$, then $\pi_t = [0.5, 0.5, 0, ..., 0]^\top$.

Exercise: Consider a Markov chain with a transition matrix $\mathbf{P} = [\,p_{ij}\,]$. Do the following calculations:

- Given that the state at t is x_1 (i.e., $\mathbb{P}(X_t = x_1) = 1$), calculate $\mathbb{P}(X_{t+1} = x_2)$ and π_{t+1}.

$$\mathbb{P}(X_{t+1} = x_2) = \mathbb{P}(X_{t+1} = x_2 \cap X_t \in \mathcal{X}) = \mathbb{P}\left(X_{t+1} = x_2 \cap \left(\bigcup_{j=1}^{n} X_t = x_j\right)\right)$$

$$= \mathbb{P}\left(\bigcup_{j=1}^{n} (X_{t+1} = x_2 \cap X_t = x_j)\right) = \sum_{j=1}^{n} \mathbb{P}(X_{t+1} = x_2 \cap X_t = x_j)$$

$$= \sum_{j=1}^{n} \mathbb{P}(X_{t+1} = x_2 | X_t = x_j)\mathbb{P}(X_t = x_j) = \sum_{j=1}^{n} p_{2j}\,\mathbb{P}(X_t = x_j)$$

$$= p_{21} \cdot 1 + p_{22} \cdot 0 + ... + p_{2n} \cdot 0 = p_{21}$$

$$\pi_{t+1} = \begin{bmatrix} \mathbb{P}(X_{t+1} = x_1) \\ \mathbb{P}(X_{t+1} = x_2) \\ \vdots \\ \mathbb{P}(X_{t+1} = x_n) \end{bmatrix} = \underbrace{\begin{bmatrix} p_{11} \\ p_{21} \\ \vdots \\ p_{n1} \end{bmatrix}}_{\text{1st column of } \mathbf{P}} = \mathbf{P} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{P}\,\pi_t$$

- Given a state distribution at t, π_t, calculate $\mathbb{P}(X_{t+1} = x_2)$ and π_{t+1}.

$$\mathbb{P}(X_{t+1} = x_2) = \sum_{j=1}^{n} p_{2j}\,\mathbb{P}(X_t = x_j)$$

$$= \begin{bmatrix} p_{21} & p_{22} & \cdots & p_{2n} \end{bmatrix} \begin{bmatrix} \mathbb{P}(X_t = x_1) \\ \mathbb{P}(X_t = x_2) \\ \vdots \\ \mathbb{P}(X_t = x_n) \end{bmatrix} = \underbrace{\begin{bmatrix} p_{21} & p_{22} & \cdots & p_{2n} \end{bmatrix}}_{\text{2nd row of } \mathbf{P}} \pi_t$$

$$\pi_{t+1} = \begin{bmatrix} \mathbb{P}(X_{t+1} = x_1) \\ \mathbb{P}(X_{t+1} = x_2) \\ \vdots \\ \mathbb{P}(X_{t+1} = x_n) \end{bmatrix} = \begin{bmatrix} \text{1st row of } \mathbf{P} \\ \text{2nd row of } \mathbf{P} \\ \vdots \\ n\text{th row of } \mathbf{P} \end{bmatrix} \pi_t = \mathbf{P}\,\pi_t$$

Proposition: $\pi_{t+1} = \mathbf{P}\,\pi_t$.

Exercise: Given a Markov chain with a single-step transition matrix $\mathbf{P} = [\,p_{ij}\,]$, calculate the 2-step transition probability $\mathbb{P}(X_{t+2} = x_i \,|\, X_t = x_j)$.

$$\mathbb{P}(X_{t+2} = x_i \,|\, X_t = x_j) = \frac{\mathbb{P}(X_{t+2} = x_i, X_t = x_j)}{\mathbb{P}(X_t = x_j)} = \frac{\mathbb{P}(X_{t+2} = x_i, X_{t+1} \in \mathcal{X}, X_t = x_j)}{\mathbb{P}(X_t = x_j)}$$

$$= \frac{\sum_{k=1}^{n} \mathbb{P}(X_{t+2} = x_i, X_{t+1} = x_k, X_t = x_j)}{\mathbb{P}(X_t = x_j)} = \frac{\sum_{k=1}^{n} \mathbb{P}(X_{t+2} = x_i \,|\, X_{t+1} = x_k, X_t = x_j)\mathbb{P}(X_{t+1} = x_k, X_t = x_j)}{\mathbb{P}(X_t = x_j)}$$

$$= \sum_{k=1}^{n} \mathbb{P}(X_{t+2} = x_i \,|\, X_{t+1} = x_k, X_t = x_j)\frac{\mathbb{P}(X_{t+1} = x_k, X_t = x_j)}{\mathbb{P}(X_t = x_j)}$$

$$= \sum_{k=1}^{n} \mathbb{P}(X_{t+2} = x_i \,|\, X_{t+1} = x_k)\,\mathbb{P}(X_{t+1} = x_k \,|\, X_t = x_j) = \sum_{k=1}^{n} p_{ik}\; p_{kj} = \left(\mathbf{P}^2\right)_{ij}$$

It can be shown in a similar way that $\mathbb{P}(X_{t+N} = x_i \,|\, X_t = x_j) = \left(\mathbf{P}^N\right)_{ij}$, for all $N = 1, 2, 3, ...$

Let $\mathbf{P}^{(N)}$ denote the N**-step transition matrix** defined by

$$\mathbf{P}^{(N)} = \begin{bmatrix} \mathbb{P}(X_{t+N} = x_1 \,|\, X_t = x_1) & \cdots & \mathbb{P}(X_{t+N} = x_1 \,|\, X_t = x_n) \\ & \ddots & \\ \mathbb{P}(X_{t+N} = x_n \,|\, X_t = x_1) & \cdots & \mathbb{P}(X_{t+N} = x_n \,|\, X_t = x_n) \end{bmatrix}$$

Corollary: $\pi_{t+N} = \mathbf{P}^{(N)}\,\pi_t$.

Proposition (Chapman–Kolmogorov Equation):

- $\mathbf{P}^{(N+M)} = \mathbf{P}^{(N)}\,\mathbf{P}^{(M)}$
- $\mathbf{P}^{(N)} = \mathbf{P}^N$

Note that in the above equations $\mathbf{P}^{(N)}$ is a name and $\mathbf{P}^N$ is the nth power of $\mathbf{P}$.

Exercise: Prove the Chapman–Kolmogorov equation.

Definition: Given a Markov chain with a transition matrix $\mathbf{P}$, a state distribution π is a **stationary distribution** if $\pi = \mathbf{P}\pi$.

Note that if $\pi = \mathbf{P}\pi$, then $\pi = \mathbf{P}^N\pi = \mathbf{P}^{(N)}\pi$.

Example:

- $\mathbf{P} = I = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$ (stationary distribution not unique)

Theorem: Every time-homogeneous Markov chain with a finite state space has a stationary distribution.

Proof: Let $\mathbf{P}$ be a transition matrix. We have

$$\mathbf{1}^\top\mathbf{P} = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ & \ddots & \\ p_{n1} & \cdots & p_{nn} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n p_{i1} & \cdots & \sum_{i=1}^n p_{in} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} = \mathbf{1}^\top$$

$$\mathbf{1}^\top(\mathbf{P} - I) = \mathbf{0}^\top$$

The vector of ones, $\mathbf{1}$, is a left-eigenvector of the transition matrix $\mathbf{P}$ corresponding to the eigenvalue 1 (left-eigenvalues = right-eigenvalues).

Let v_j denote the jth column of $\mathbf{P} - I$. The second line above implies $\mathbf{1}^\top v_j = 0$ (i.e., $\mathbf{1} \perp v_j$).

Because $\mathbf{1}$ and v_j, $j = 1, ..., n$, are $n + 1$ vectors in $\mathbb{R}^n$, they must be linearly dependent. That is, there are real numbers a and b_j, $j = 1, ..., n$, not all 0, such that

$$a\mathbf{1} + \sum_{j=1}^{n} b_j v_j = \mathbf{0}$$

Then, left-multiply both sides of the above equation by $\mathbf{1}^\top$ and obtain

$$a\,\mathbf{1}^\top\mathbf{1} + \sum_{j=1}^{n} b_j\,\mathbf{1}^\top v_j = \mathbf{1}^\top\mathbf{0}$$
$$a\cdot n + \sum_{j=1}^{n} b_j\cdot 0 = 0 \implies a = 0$$

Therefore, we obtain

$$\sum_{j=1}^{n} b_j v_j = \mathbf{0}$$

where not all b_j are 0. Recall that v_j denotes the jth column of $\mathbf{P} - I$. Therefore, the above equation can be rewritten as

$$\begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = (\mathbf{P} - I) \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{=\mathbf{b}} = (\mathbf{P} - I)\,\mathbf{b} = \mathbf{0} \iff \mathbf{b} = \mathbf{P}\mathbf{b}$$

The vector $\mathbf{b}$ is a right-eigenvector of $\mathbf{P}$ corresponding to the eigenvalue 1. We now show that the entries of $\mathbf{b}$, b_j, $j = 1, ..., n$, all have the same sign.

Consider the ith row of $\mathbf{b} = \mathbf{P}\mathbf{b}$,

$$b_i = \begin{bmatrix} p_{i1} & \cdots & p_{in} \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \sum_{j=1}^{n} p_{ij} b_j$$

Take the absolute value and we have

$$|b_i| = \Big| \sum_{j=1}^{n} p_{ij} b_j \Big| \le \sum_{j=1}^{n} p_{ij} |b_j|$$

with equality if and only if all b_j have the same sign. Summing over $i = 1, 2, ..., n$,

$$\sum_{i=1}^{n} |b_i| \le \sum_{i=1}^{n}\sum_{j=1}^{n} \big(p_{ij}|b_j|\big) = \sum_{j=1}^{n} \underbrace{\left(\sum_{i=1}^{n} p_{ij}\right)}_{=1} |b_j| = \sum_{j=1}^{n} |b_j|$$

with equality if and only if all b_j have the same sign. Since we do have an equality $\sum_{i=1}^{n} |b_i| = \sum_{j=1}^{n} |b_j|$, it must hold that all b_j do have the same sign.

Now if $b_i \ge 0$, we let $\pi = \frac{\mathbf{b}}{\sum_{j=1}^{n} b_j}$; otherwise we let $\pi = -\frac{\mathbf{b}}{\sum_{j=1}^{n} b_j}$. We have (1) all entries of π are non-negative, (2) $\sum_{i=1}^{n} \pi_i = 1$, i.e., π is a state distribution, and (3)

$$\pi = \frac{1}{\sum_{j=1}^{n} b_j}\mathbf{b} = \frac{1}{\sum_{j=1}^{n} b_j}\mathbf{P}\mathbf{b} = \mathbf{P}\frac{\mathbf{b}}{\sum_{j=1}^{n} b_j} = \mathbf{P}\pi$$

Therefore, π is a stationary distribution.

Since for any time-homogeneous Markov chain with a finite state space (with an arbitrary transition matrix $\mathbf{P}$) we can find a stationary distribution π following the above procedure, we have shown the result. ■

Definition: Given a Markov chain with a transition matrix $\mathbf{P}$, a state x_i is said to be **accessible** from state x_j, denoted as $x_i \leftarrow x_j$, if there exists some integer $N > 0$ such that $\mathbb{P}(X_N = x_i \,|\, X_0 = x_j) > 0$ (i.e., $(\mathbf{P}^{(N)})_{ij} > 0$). Two states x_i and x_j are said to **communicate**, denoted as $x_i \leftrightarrow x_j$, if they are accessible from each other.

Accessibility means it is possible to get to x_i from x_j.

Definition: A Markov chain with a transition matrix $\mathbf{P}$ is said to be **irreducible** if all states communicate, that is, for each pair of states (x_i, x_j), there exists $N_{ij} > 0$ such that $(\mathbf{P}^{(N_{ij})})_{ij} > 0$. It is said to be **reducible** if it is not irreducible.

Irreducibility means it is possible to transition from any state to any other state.

Review of linear algebra:

Given an $m \times n$ real matrix $\mathbf{P}$, the **rank** of $\mathbf{P}$ is the maximum number of its linearly independent columns (or rows), the **nullity** of $\mathbf{P}$ is the maximum number of linearly independent real vectors in its null space $\{v \in \mathbb{R}^n : \mathbf{P}v = \mathbf{0}\}$, and we have the **rank-nullity theorem**:

$$\text{rank}(\mathbf{P}) + \text{nullity}(\mathbf{P}) = n$$

If $\mathbf{P}$ is $n \times n$, then we have $\text{nullity}(\mathbf{P}) = \text{nullity}(\mathbf{P}^\top)$.

Theorem: Given a time-homogeneous Markov chain with a finite state space, if it is irreducible, then its stationary distribution is unique.

Proof: Let $v \in \mathbb{R}^n$ be such that $v^\top \mathbf{P} = v^\top$ (such a vector exists, e.g., $v = \mathbf{1}$). Find the maximum entry of v, v_j, that is, $v_j \geq v_k$ for all $k = 1, ..., n$. Let x_i be an arbitrary state in the finite state space $\mathcal{X}$. Because the chain is irreducible, there exists $N > 0$ such that $(\mathbf{P}^{(N)})_{ij} > 0$. Consider the jth column of $v^\top = v^\top \mathbf{P} = v^\top \mathbf{P}^N = v^\top \mathbf{P}^{(N)}$,

$$\begin{aligned}
v_j &= \sum_{k=1}^{n} v_k (\mathbf{P}^{(N)})_{kj} = v_i (\mathbf{P}^{(N)})_{ij} + \sum_{k \neq i} v_k (\mathbf{P}^{(N)})_{kj} \\
&\leq v_i (\mathbf{P}^{(N)})_{ij} + \sum_{k \neq i} v_j (\mathbf{P}^{(N)})_{kj} \qquad \text{(since } v_k \leq v_j \text{ and } (\mathbf{P}^{(N)})_{kj} \geq 0\text{)} \\
&\leq v_j (\mathbf{P}^{(N)})_{ij} + \sum_{k \neq i} v_j (\mathbf{P}^{(N)})_{kj} \qquad \text{(with equality iff } v_i = v_j \text{ since } (\mathbf{P}^{(N)})_{ij} > 0\text{)} \\
&= v_j \sum_{k=1}^{n} (\mathbf{P}^{(N)})_{kj} = v_j \cdot 1 = v_j
\end{aligned}$$

Since we do have an equality, it must hold that $v_i = v_j$, that is, the ith entry of v is equal to its maximum entry v_j. Because x_i is arbitrary (i.e., $i = 1, ..., n$), the vector v must satisfy that all of its entries are equal. Therefore, we have shown that $v^\top \mathbf{P} = v^\top$ implies $v = c \cdot \mathbf{1}$ with some constant c.

Since $(\mathbf{P} - I)^\top v = \mathbf{0}$ iff $v^\top \mathbf{P} = v^\top$, the above shows that the nullity of $(\mathbf{P} - I)^\top$ is 1. Then, according to the rank-nullity theorem, the nullity of $(\mathbf{P} - I)$ is also 1, indicating that all real vectors $\mathbf{b}$ that satisfy $(\mathbf{P} - I)\,\mathbf{b} = \mathbf{0}$ (i.e., $\mathbf{b} = \mathbf{P}\mathbf{b}$) are linearly dependent.

Let π be a stationary distribution of $\mathbf{P}$, i.e., satisfying $\pi = \mathbf{P}\pi$. Any stationary distribution of $\mathbf{P}$, π', must satisfy $\pi' = c \cdot \pi$ with some constant c (by the linear dependence shown above). Because $1 = \sum_{i=1}^n \pi'_i = c \sum_{i=1}^n \pi_i = c$, we must have $\pi' = \pi$, i.e., the stationary distribution π is unique. ■

When having a unique stationary distribution π, we want to study the long-run behavior of the Markov chain, in particular, whether $\pi_N = \mathbf{P}^{(N)}\pi_0 \to \pi$. ("stability")

Example:

- $\mathbf{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ (alternating, not converging)

$$\pi_{t+1} = \mathbf{P}\,\pi_t = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \pi_{t,1} \\ \pi_{t,2} \end{bmatrix} = \begin{bmatrix} \pi_{t,2} \\ \pi_{t,1} \end{bmatrix}$$

Definition: Let $\mathcal{T}(x_i) = \{N > 0 : (\mathbf{P}^{(N)})_{ii} > 0\}$, the set of all $N > 0$ such that the Markov chain can start in x_i at $t = 0$ and end in x_i at $t = N$. The **period** of x_i is the greatest common divisor (gcd) of $\mathcal{T}(x_i)$.

Lemma: If a Markov chain is irreducible, then the period of all states is equal,

$$\gcd\, \mathcal{T}(x_i) = \gcd\, \mathcal{T}(x_j), \qquad \forall\, x_i, x_j \in \mathcal{X}$$

and is called the period of the chain.

Proof: Given x_i and x_j, because the chain is irreducible, there exist M and N such that $(\mathbf{P}^{(M)})_{ij} > 0$ and $(\mathbf{P}^{(N)})_{ji} > 0$. Then, $(\mathbf{P}^{(M+N)})_{ii} \geq (\mathbf{P}^{(M)})_{ij} \cdot (\mathbf{P}^{(N)})_{ji} > 0$ and $(\mathbf{P}^{(M+N)})_{jj} \geq (\mathbf{P}^{(N)})_{ji} \cdot (\mathbf{P}^{(M)})_{ij} > 0$. Thus, $M + N \in \mathcal{T}(x_i)$ and $M + N \in \mathcal{T}(x_j)$.

If $K \in \mathcal{T}(x_i)$, i.e., $(\mathbf{P}^{(K)})_{ii} > 0$, then $(\mathbf{P}^{(M+N+K)})_{jj} \geq (\mathbf{P}^{(N)})_{ji} \cdot (\mathbf{P}^{(K)})_{ii} \cdot (\mathbf{P}^{(M)})_{ij} > 0$, i.e., $M + N + K \in \mathcal{T}(x_j)$. Because $M + N \in \mathcal{T}(x_j)$ and $M + N + K \in \mathcal{T}(x_j)$, gcd $\mathcal{T}(x_j)$ is a common divisor of $M + N$ and $M + N + K$, that is, $M + N$ and $M + N + K$ are both multiples of gcd $\mathcal{T}(x_j)$. In this case, $K = (M + N + K) - (M + N)$ is also a multiple of gcd $\mathcal{T}(x_j)$.

We have shown that for any $K \in \mathcal{T}(x_i)$, gcd $\mathcal{T}(x_j)$ is a divisor of K, i.e., gcd $\mathcal{T}(x_j)$ is a common divisor of $\mathcal{T}(x_i)$. Then, by the definition of gcd, gcd $\mathcal{T}(x_j)$ is a divisor of gcd $\mathcal{T}(x_i)$.

Using a similar argument, we can also show that gcd $\mathcal{T}(x_i)$ is a divisor of gcd $\mathcal{T}(x_j)$. Therefore, gcd $\mathcal{T}(x_i)$ and gcd $\mathcal{T}(x_j)$ are divisors of each other, which implies gcd $\mathcal{T}(x_i) =$ gcd $\mathcal{T}(x_j)$. ■

Definition: An irreducible Markov chain is said to be **aperiodic** if its period is equal to 1 (i.e., gcd $\mathcal{T}(x) = 1$ for all $x \in \mathcal{X}$); it is said to be **periodic** with period k if gcd $\mathcal{T}(x) = k > 1$.

Theorem: An irreducible Markov chain is aperiodic if and only if there exists $N > 0$ such that $(\mathbf{P}^{(N)})_{ij} > 0$ for all $i, j = 1, ..., n$ (the N is independent of i, j), called **regular**.

Proof: Suppose there exists $N > 0$ such that $(\mathbf{P}^{(N)})_{ij} > 0$ for all $i, j = 1, ..., n$. Then, we have (1) $(\mathbf{P}^{(N)})_{ii} > 0$, and (2) $(\mathbf{P}^{(N+1)})_{ii} = \sum_{j=1}^{n} (\mathbf{P}^{(N)})_{ij} \mathbf{P}_{ji} > 0$. Therefore, $N, N + 1 \in \mathcal{T}(x_i)$, which implies gcd $\mathcal{T}(x_i) = 1$. Thus, the chain is aperiodic.

Now suppose the chain is aperiodic. For each state x_i, because gcd $\mathcal{T}(x_i) = 1$, there exists $N_i > 0$ such that $N_i, N_i + 1 \in \mathcal{T}(x_i)$. For any $N \geq N_i(N_i - 1)$, it can be written as $N = qN_i + r$ with non-negative integers q, r satisfying $q \geq N_i - 1 \geq r$. Hence, $N = qN_i + r(N_i + 1 - N_i) = (q - r)N_i + r(N_i + 1)$. Then, we have

$$(\mathbf{P}^{(N)})_{ii} = (\mathbf{P}^{(q-r)N_i + r(N_i+1)})_{ii} \geq (\mathbf{P}^{(q-r)N_i})_{ii} \cdot (\mathbf{P}^{r(N_i+1)})_{ii} \geq ((\mathbf{P}^{N_i})_{ii})^{q-r} \cdot ((\mathbf{P}^{N_i+1})_{ii})^{r} > 0$$

Therefore, we have shown that for each state x_i, there exists $N_i' = N_i(N_i - 1)$ such that $(\mathbf{P}^{(N)})_{ii} > 0$ for all $N \geq N_i'$.

Because the chain is irreducible, for each pair of states (x_i, x_j), there exists $N_{ij} > 0$ such that $(\mathbf{P}^{(N_{ij})})_{ij} > 0$. Now let $N' \geq \max_i N_i' + \max_{i,j} N_{ij}$. This N' is independent of i, j. For each (x_i, x_j), we have

$$(\mathbf{P}^{(N')})_{ij} = (\mathbf{P}^{(N_{ij}' + N_{ij})})_{ij} \geq (\mathbf{P}^{(N_{ij}')})_{ii} \cdot (\mathbf{P}^{(N_{ij})})_{ij} > 0$$

where $N_{ij}' = N' - N_{ij}$ satisfies

$$N_{ij}' = N' - N_{ij} \geq N_i' + N_{ij} - N_{ij} = N_i'$$

(therefore $(\mathbf{P}^{(N_{ij}')})_{ii} > 0$). This completes the proof. ■

Remark: If $(\mathbf{P}^{(N)})_{ij} > 0$ for $i, j = 1, ..., n$, then $(\mathbf{P}^{(N+M)})_{ij} > 0$ for $i, j = 1, ..., n$ for all $M \geq 0$. This can be seen from

$$(\mathbf{P}^{(N+1)})_{ij} = \sum_{k=1}^{n} (\mathbf{P}^{(N)})_{ik} \mathbf{P}_{kj}$$

where $(\mathbf{P}^{(N)})_{ik} > 0$ for all k and $\mathbf{P}_{kj} > 0$ for at least one k.

Theorem (Fundamental Theorem of Markov Chains): Given a time-homogeneous Markov chain with a finite state space, if it is irreducible and aperiodic (i.e., **ergodic**), then, starting with any initial distribution π_0, the state distribution π_t converges to its (unique) stationary distribution π as $t \to \infty$, i.e., $\lim_{t\to\infty} \pi_t = \pi$. In particular, let $\mathbf{P}$ denote its transition matrix. There exist $0 < \theta < 1$ and $C > 0$ such that

$$\|(\mathbf{P}^{(N)})_{\cdot j} - \pi\|_{TV} \leq C\,\theta^N, \qquad j = 1, ..., n$$

where $(\mathbf{P}^{(N)})_{\cdot j}$ denotes the jth column of $\mathbf{P}^{(N)}$, and $\|\pi' - \pi\|_{TV} = \frac{1}{2}\sum_{i=1}^{n} |\pi'_i - \pi_i|$ is called the total variation distance between distributions π' and π.

Proof: Read https://arxiv.org/pdf/2204.00784.pdf

The latter also characterizes a speed of convergence that is independent of the initial distribution.

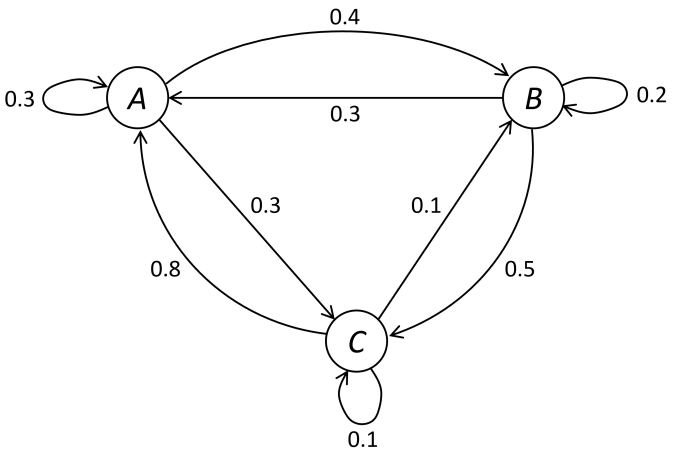

Figure 2: Graph representation of Markov chain.

Definition: A **directed graph** is a pair $G = (V, E)$, where V is a set of **vertices** (or, **nodes**), and E is a set of ordered pairs of vertices, called **edges** (or **arcs**). A **weighted graph** (or, a **network**) is a graph in which a number is assigned to each edge (called the **weight** of the edge).

7 Applications

7.1 PageRank algorithm

PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. It is named after both the term "web page" and co-founder Larry Page. PageRank is a way of measuring the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

See slides "Google's PageRank Algorithm."

7.2 Training Markov chain models

Definition: A **partition** of a set X is a set of non-empty subsets of X, $\{X_i\}_{i\in I}$, such that every element x of X is in exactly one of these subsets, i.e., $\bigcup_{i\in I} X_i = X$ and $X_i \cap X_j = \emptyset, \forall i \neq j$.

Problem: Given trajectory data $\{(x_t^k, x_{t+1}^k)\}_{k=1}^K$ of a process, estimate a Markov chain model that can approximately represent the process.

Examples:

- Real-world data: Traffic data can be used to estimate a human driver model; weather data can be used to estimate a model for weather forecast, etc.
- Simulated data: A process or a dynamic system may have a model, but either highly-complex or not given in an explicit function form (e.g., a Simulink block diagram), making it difficult to conduct any theoretical analysis with the model. Meanwhile, the model is able to generate trajectory data, which can be used to estimate a Markov chain model for theoretical analysis.

Solution procedure:

Step 1: Partition the state space $X \subset \mathbb{R}^n$ into cells $\{X_i\}_{i=1}^n$.

Here, by partition, we mean $\bigcup_{i=1}^n X_i = X$ and $X_i \cap X_j = \emptyset$ for $i \neq j$.

Although the definition does not require a partition to be finite (i.e., a partition can be an infinite collection of subsets), we choose to partition X into a finite number of cells for computational purposes.

Step 2: Assign a unique Markov state x_i to each cell X_i. For later purposes, we let x_i be the mean value (or middle point) of the cell (i.e., $x_i = \frac{\int_{X_i} x \, \mathrm{d}x}{\int_{X_i} \mathrm{d}x}$).

Previously, x_i is just a name of the Markov state. Now it has certain physical meaning.

Step 3: For each j, count the number of transitions from X_j to X_i, N_{ij}, for all i. A transition from X_j to X_i is a data point (x_t^k, x_{t+1}^k) such that $x_t^k \in X_j$ and $x_{t+1}^k \in X_i$.

Step 4: Estimate the transition probability $p_{ij} = \mathbb{P}(x^+ = x_i \,|\, x = x_j)$ as

$$p_{ij} = \frac{N_{ij}}{\sum_{k=1}^n N_{kj}}$$

Therefore, $\sum_{i=1}^n p_{ij} = \frac{\sum_{i=1}^n N_{ij}}{\sum_{k=1}^n N_{kj}} = 1$.

Step 5: Construct the transition matrix $\mathbf{P}$ as $\mathbf{P} = [\, p_{ij} \,]$.

Computer algorithms:

Algorithm 1 Batch Estimation of Markov Chain Model (version 1)

1: Initialize $N_{ij}(0) = 0$ for all $i, j = 1, ..., n$.
2: **for** $k = 1 : K$ **do**
3: $N_{ij}(k) = N_{ij}(k-1) + \mathbb{1}_{ij}(x_t^k, x_{t+1}^k)$ for all $i, j = 1, ..., n$, where $\mathbb{1}_{ij}(x, x^+)$ is called an indicator function with $\mathbb{1}_{ij}(x, x^+) = 1$ if $x \in X_j$ and $x^+ \in X_i$ and $\mathbb{1}_{ij}(x, x^+) = 0$ otherwise.
4: **end for**
5: $p_{ij} = \frac{N_{ij}(K)}{\sum_{l=1}^n N_{lj}(K)}$ for all $i, j = 1, ..., n$.

Algorithm 1 has the following drawback: As k increases, the variables $N_{ij}(k)$ go to infinity. This problem can be solved by keeping track of the **frequency** of transition, $F_{ij}(k)$, instead of the **number** of of transition, $N_{ij}(k)$, where

$$\begin{aligned}
F_{ij}(k) &= \frac{N_{ij}(k)}{k} \\
&= \frac{N_{ij}(k-1) + \mathbb{1}_{ij}(x_t^k, x_{t+1}^k)}{k} \\
&= \frac{(k-1)F_{ij}(k-1) + \mathbb{1}_{ij}(x_t^k, x_{t+1}^k)}{k} \\
&= \frac{k-1}{k} F_{ij}(k-1) + \frac{1}{k}\,\mathbb{1}_{ij}(x_t^k, x_{t+1}^k) \qquad \text{(a convex combination of previous } F_{ij} \text{ and current } \mathbb{1}_{ij}) \\
&= F_{ij}(k-1) + \frac{1}{k}\Big(\mathbb{1}_{ij}(x_t^k, x_{t+1}^k) - F_{ij}(k-1)\Big)
\end{aligned}$$

We have

$$p_{ij} = \frac{N_{ij}(K)}{\sum_{l=1}^n N_{lj}(K)} = \frac{N_{ij}(K)/K}{\sum_{l=1}^n N_{lj}(K)/K} = \frac{F_{ij}(K)}{\sum_{l=1}^n F_{lj}(K)}$$

Algorithm 2 Batch Estimation of Markov Chain Model (version 2)

1: Initialize $F_{ij}(0) = 0$ for all $i, j = 1, ..., n$.
2: **for** $k = 1 : K$ **do**
3: $F_{ij}(k) = F_{ij}(k-1) + \frac{1}{k}\big(\mathbb{1}_{ij}(x_t^k, x_{t+1}^k) - F_{ij}(k-1)\big)$ for all $i, j = 1, ..., n$, where $\mathbb{1}_{ij}(x, x^+) = 1$ if $x \in X_j$ and $x^+ \in X_i$ and $\mathbb{1}_{ij}(x, x^+) = 0$ otherwise.
4: **end for**
5: $p_{ij} = \frac{F_{ij}(K)}{\sum_{l=1}^n F_{lj}(K)}$ for all $i, j = 1, ..., n$.

From the above algorithm, we can derive a recursive version for online updating the Markov chain model as new data points come:

Algorithm 3 Recursive Estimation of Markov Chain Model

1: Initialize $F_{ij}(0) = \varepsilon$ for all $i, j = 1, ..., n$, where $\varepsilon > 0$ is a small constant.
2: **for** $t = 1, 2, ...$ **do**
3: $F_{ij}(t) = F_{ij}(t-1) + \alpha\big(\mathbb{1}_{ij}(x_{t-1}, x_t) - F_{ij}(t-1)\big)$ for all $i, j = 1, ..., n$, where $\alpha \in (0, 1)$ is a constant update rate.
4: $p_{ij}(t) = \frac{F_{ij}(t)}{\sum_{l=1}^n F_{lj}(t)}$ for all $i, j = 1, ..., n$.
5: **end for**

We can observe the following two differences:

- We replace the "time-varying" update rate $\frac{1}{k}$ with a constant rate $\alpha \in (0, 1)$. Because Algorithm 2 is equivalent to Algorithm 1, using the rate $\frac{1}{k}$ (which decreases as k increases), every data point has the same weight. Then, using a constant rate α (which does not decrease as t increases), we are putting more weights on data with larger t (i.e., more recent data). In other words, we are forgetting previous (outdated) data and updating the model to fit recent data. As a result, the Markov chain model is able to "track" a slowly-changing process (e.g., system parameter changing due to fuel consumption or aging).

- We initialize $F_{ij}(0) = \varepsilon > 0$ instead of $F_{ij}(0) = 0$.

  For batch estimation, because we use the update rate $\frac{1}{k}$, at $k = 1$, we have

  $$F_{ij}(1) = F_{ij}(0) + \frac{1}{1}\big(\mathbb{1}_{ij}(x_t^1, x_{t+1}^1) - F_{ij}(0)\big) = \mathbb{1}_{ij}(x_t^1, x_{t+1}^1),$$

  i.e., due to the update rate 1, the value of $F_{ij}(0)$ is immediately "forgotten" (so it does not matter how we initialize $F_{ij}(0)$).

  For recursive estimation, if we initialize $F_{ij}(0) = 0$, then for all j such that $x_0 \notin X_j$, we have

  $$F_{ij}(1) = F_{ij}(0) + \alpha\big(\mathbb{1}_{ij}(x_0, x_1) - F_{ij}(0)\big) = 0,\ \forall i = 1, ..., n$$

  which will cause the denominator in Step 4 to be 0, returning an error. By initializing $F_{ij}(0)$ as a small positive constant, we avoid this issue. Meanwhile, because it is small, its influence will be damped quickly due to the "forgetting" of using a constant update rate.

The obtained Markov chain model can be used to predict system behavior probabilistically – predict a probability distribution π_t of future state x_t. In some cases a deterministic prediction of state is useful. In such cases, we can use the expected value as the deterministic prediction, i.e.,

$$\begin{aligned}
\hat{x}_t = \mathbb{E}(x_t) &= \int_X x \cdot \mathrm{pdf}(x)\,\mathrm{d}x = \sum_{i=1}^{n} \int_{X_i} x \cdot \mathrm{pdf}(x)\,\mathrm{d}x \\
&= \sum_{i=1}^{n} \int_{X_i} x \cdot \mathrm{pdf}_i\,\mathrm{d}x = \sum_{i=1}^{n} \mathrm{pdf}_i \int_{X_i} x\,\mathrm{d}x = \sum_{i=1}^{n} \mathrm{pdf}_i \left(x_i \int_{X_i} \mathrm{d}x \right) \\
&= \sum_{i=1}^{n} x_i \left(\mathrm{pdf}_i \int_{X_i} \mathrm{d}x \right) = \sum_{i=1}^{n} x_i \cdot \pi_{t,i}
\end{aligned}$$

To get the expression $\hat{x}_t = \sum_{i=1}^{n} x_i \cdot \pi_{t,i}$, we have assumed that on each cell X_i, the state is uniformly distributed and $\mathrm{pdf}(x) = \mathrm{pdf}_i = \frac{\pi_{t,i}}{\int_{X_i} \mathrm{d}x}$ (the probability density is equal to the probability of the set, $\pi_{t,i}$, divided by the area of the set).

The assumption of uniform distribution on cells may not hold, and this causes prediction errors. In general, if the cell is small, then the error caused by this will also be small. Therefore, a finer partition (i.e., using smaller but a larger number of cells to partition the state space X) typically leads to a more accurate Markov chain model, but also requires more data and computation to train a larger transition matrix.

See Matlab code "Training MC model."

Module III: Markov Decision Processes

8 Definitions

A decision-maker, agent, or controller is faced with the problem of influencing the behavior of a probabilistic system as it evolves through time. It does this by making decisions or choosing actions. Its goal is to choose a sequence of actions which causes the system to perform optimally with respect to some predetermined performance criterion.

Definition: We consider the following collection of objects:

$$\{T, S, A, p(s'|s,a), r_t(s,a)\}$$

where T denotes the set of **decision epochs** (either finite, $T = \{0, 1, ..., N\}$, or infinite, $T = \{0, 1, 2, ...\}$), $S = \{s_1, s_2, ..., s_n\}$ denotes a finite set of states (called the **state space**), $A = \{a_1, a_2, ..., a_m\}$ denotes a finite set of actions (called the **action space**), $p(s'|s,a)$ denotes the probability of reaching a state $s' \in S$ at the next decision epoch (**next state**) if the state at the current decision epoch (**current state**) is $s \in S$ and the action chosen at the current decision epoch (**current action**) is $a \in A$, called a **transition probability**, and $r_t(s,a)$ denotes a **reward** received at the decision epoch $t \in T$ if in state $s \in S$ and action $a \in A$. We call such a collection of objects a **Markov Decision Process (MDP)**.

Remarks:

- The decision epochs are often interpreted as times. Therefore, at a decision epoch $t \in T$ is often said to be at a time (or step) $t \in T$. However, in many problems decision epochs may not correspond to times. Example: a decision tree.
- When T is finite, $T = \{0, 1, ..., N\}$, the problem is said to be a **finite-horizon** problem; when T is infinite, $T = \mathbb{Z}_+$, the problem is said to be an **infinite-horizon** problem.
- The set of allowable actions in state s may be state-dependent, and thus may be denoted as A_s, where A_{s_i} may not be equal to A_{s_j}. Example: a decision tree. Incorporating state-dependent action sets A_s has little effect on theory. Therefore, for simplicity, we assume $A_s = A$ for all $s \in S$. In many practical problems, this can be easily achieved through adding some dummy actions at certain s.
- The transitions are Markovian (i.e., have the Markov property): Given the current state and current action, (s,a), the probability distribution of next state, s', does not depend on past history of states and actions.

$$p(s'|s,a) = \mathbb{P}(s_{t+1} = s' \,|\, s_t = s, a_t = a) = \mathbb{P}(s_{t+1} = s' \,|\, s_t = s, a_t = a, s_{t-1}, a_{t-1}, ..., s_0, a_0)$$

- Transition probabilities sum up to 1:

$$\sum_{s' \in S} p(s'|s,a) = \sum_{s'=s_1}^{s_n} p(s'|s,a) = 1$$

- The reward function $r(s,a)$ often does not change with decision epoch t (i.e., time-independent). However, in finite-horizon problems, it is typical that no decision is made at the last decision epoch N and we use a separate reward function that only depends on state, $r_N(s)$, to represent how good the final state is (called a **terminal reward**). The subscript t of $r_t(s,a)$ facilitate treating terminal reward.

- In some problems, the received reward depends not only on the current state and action (s, a) but also on the next state s', i.e., we have a reward function in the form of $r_t(s, a, s')$. In such a case, let

$$r_t(s, a) = \sum_{s' \in S} r_t(s, a, s')\, p(s'|s, a)$$

  i.e., $r_t(s, a)$ is the expected reward conditioned on the current state and action pair.

Definition: A **decision rule** describes a procedure for choosing actions at a specified decision epoch. A **deterministic Markovian decision rule** is a function from states to actions, $d_t : S \to A$, indicating that, at decision epoch t, if the state is s, then choosing the action $a = d_t(s)$. A **policy**, π, specifies the decision rule to be used at each decision epoch, i.e., $\pi = (d_0, d_1, ..., d_N)$. A policy is **stationary** (or time-independent) is $d_t = d$ for all $t \in T$.

9 A one-period Markov decision problem

Problem: Let $T = \{0, 1\}$. Consider a terminal reward function $r_1(s')$, i.e., when the agent reaches state s' at the end time $t = 1$, it receives a terminal reward $r_1(s')$, and this holds for all $s' \in S$. The agent starts in state s at $t = 0$ and tries to choose an action $a \in A$ to maximize the sum of the immediate reward, $r_0(s, a)$, and the expected terminal reward.

Remarks:

- Because the state evolves stochastically, at $t = 0$, the agent does not know which state s' it will reach at $t = 1$. Therefore, the best thing it can do is to maximize the expected reward.
- Its objective can be expressed as

$$\max_{a \in A}\ r_0(s, a) + \mathbb{E}\big\{r_1(s') \,|\, s, a\big\} = \max_{a \in A}\ \mathbb{E}\big\{r_0(s, a) + r_1(s') \,|\, s, a\big\}$$

- Recall the conditional expectation of a random variable X given an event A is

$$\mathbb{E}(X \,|\, A) = \sum_x x\, \mathbb{P}(X = x \,|\, A)$$

  where the sum is taken over all possible values of X. Thus, $\mathbb{E}\big\{r_1(s') \,|\, s, a\big\}$ can be written as

$$\begin{aligned}\mathbb{E}\big\{r_1(s') \,|\, s, a\big\} &= \sum_{s' \in S} r_1(s')\, \mathbb{P}(\text{next state} = s' \,|\, \text{current state} = s, \text{current action} = a) \\ &= \sum_{s' \in S} r_1(s')\, p(s'|s, a)\end{aligned}$$

- Therefore, the objective can be written as

$$\max_{a \in A}\ r_0(s, a) + \sum_{s' \in S} p(s'|s, a)\, r_1(s')$$

- This problem is easy to solve: For every $a \in A$, we evaluate $r_0(s, a) + \sum_{s' \in S} p(s'|s, a)\, r_1(s')$. Then, we compare the values and find the a that has the highest value, i.e.,

$$a^* \in \operatorname*{argmax}_{a \in A}\ r_0(s, a) + \sum_{s' \in S} p(s'|s, a)\, r_1(s')$$

- Since the above procedure applies to any initial state $s \in S$, we can generate an **optimal decision rule**, $d_0^* : S \to A$, such that

$$d_0^*(s) \in \underset{a \in A}{\text{argmax}} \; r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s')$$

Solution:

$$d_0^*(s) \in \underset{a \in A}{\text{argmax}} \; r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s')$$

Notation: Let X be a set and $g(x)$ be a real-valued function on X,

$$\underset{x \in X}{\text{argmax}}\, g(x) = \big\{ x' \in X : g(x') \geq g(x) \text{ for all } x \in X \big\}$$

The obtained optimal decision rule $d_0^* : S \to A$ is a deterministic rule – given a state $s \in S$, it chooses an action $a = d_0^*(s)$ deterministically.

Question: Can we obtain a larger reward by using a randomized decision rule – given a state $s \in S$, choosing actions $a \in A$ according to some probability distribution $q : A \to [0,1]$?

First note that randomized decision rules "include" deterministic rules, where the latter can be viewed as a randomized decision rule that assigns probability 1 to $a = d_0^*(s)$ and 0 to all other actions at each state $s \in S$. Therefore, the highest reward that can be obtained using a randomized decision rule is no less than the highest reward by using deterministic rules. (The maximum on a larger set is no less than the maximum on a smaller set.)

Recall that for each action $a \in A$, the reward is

$$r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s').$$

If we consider a probability distribution q for choosing actions, then the expected reward is

$$\mathbb{E}\Big\{ r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s') \,\Big|\, a \sim q \Big\} = \sum_{a \in A} q(a) \Big(r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s') \Big)$$

which satisfies

$$\begin{aligned} \sum_{a \in A} q(a) \Big(r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s') \Big) &\leq \sum_{a \in A} q(a) \Big(\max_{\alpha \in A} r_0(s,\alpha) + \sum_{s' \in S} p(s'|s,\alpha)\, r_1(s') \Big) \\ &= \max_{\alpha \in A} r_0(s,\alpha) + \sum_{s' \in S} p(s'|s,\alpha)\, r_1(s') \end{aligned}$$

This means we cannot obtain a larger expected reward by considering randomized decision rules.

Value functions: Now define

$$Q_0(s,a) = \mathbb{E}\big\{ r_0(s,a) + r_1(s') \,|\, s, a \big\} = r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s')$$

and

$$V_0(s) = \max_{a \in A} \; \mathbb{E}\big\{ r_0(s,a) + r_1(s') \,|\, s, a \big\} = \max_{a \in A} \; r_0(s,a) + \sum_{s' \in S} p(s'|s,a)\, r_1(s')$$

According to their definitions, $Q_0(s,a)$ represents the expected cumulative reward $(r_0 + r_1)$ that we will receive if we choose action a in state s at the decision epoch $t = 0$, $V_0(s)$ represents the highest expected cumulative reward that we can obtain if we are in state s at $t = 0$, and we have

$$(1) \qquad V_0(s) = \max_{a \in A} Q_0(s,a)$$

$$(2) \qquad d_0^*(s) \in \operatorname*{argmax}_{a \in A} Q_0(s,a)$$

Question: If we want to maximize our expected cumulative reward and we have the freedom to choose the state s to start with, which state should we choose?

We should choose a state s^* that has the highest V-value, i.e.,

$$s^* \in \operatorname*{argmax}_{s \in S} V_0(s)$$

Compare $V_0(s)$ and $V_1(s) = r_1(s)$:

- $V_0(s)$ represents the highest expected cumulative reward that we can obtain if we are in state s at the decision epoch $t = 0$, where cumulative reward means the sum of rewards r_t over all future epochs, $t \geq 0$.
- $V_1(s)$ represents the reward that we receive if we are in state s at $t = 1$, which is also the "highest expected cumulative reward" over $t \geq 1$ that we can obtain if we are in state s at $t = 1$.

We can see that the role of $V_0(s)$ at $t = 0$ is similar to the role of $V_1(s)$ at $t = 1$. This similarity allows us to develop an algorithm, called **(Stochastic) Dynamic Programming**, to solve any multi-period Markov decision problem, $T = \{0, 1, ..., N\}$, by decomposing it into a series of one-period Markov decision problems. By solving these one-period Markov decision problems using the procedure introduced above, we will obtain the solution to the original multi-period Markov decision problem.

10 Finite-horizon MDP and Stochastic Dynamic Programming

For an MDP $\{T, S, A, p(s'|s,a), r_t(s,a)\}$ with a finite number of decision epochs $T = \{0, 1, ..., N\}$, a policy is a sequence of decision rules, $\pi = \{d_0, d_1, ..., d_{N-1}\}$, that describe the procedure for choosing actions at each decision epoch $t \in T$.

We now use Π^{RH} to denote all randomized history-dependent policies and use Π^{DM} to denote all deterministic Markovian policies. Then, $\Pi^{\text{DM}} \subset \Pi^{\text{RH}}$.

The goal is to find a policy $\pi \in \Pi^{\text{RH}}$ that maximizes the expected total/cumulative reward over the decision making horizon,

$$\begin{aligned} \mathcal{R}^\pi(s) &= \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{N-1} r_t(s_t, a_t) + r_N(s_N) \right\} \\ &= \mathbb{E} \left\{ \sum_{t=0}^{N-1} r_t(s_t, a_t) + r_N(s_N) \,\middle|\, s_0 = s, a_t \sim d_t(h_t) \right\} \end{aligned}$$

where $h_t = \{s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t\}$ denotes the collection of all historical information up to t. If $\pi \in \Pi^{\text{DM}}$, then $\mathcal{R}^\pi(s)$ can be written as

$$\mathcal{R}^\pi(s) = \mathbb{E} \left\{ \sum_{t=0}^{N-1} r_t(s_t, d_t(s_t)) + r_N(s_N) \,\middle|\, s_0 = s \right\}$$

Such a policy, π^*, is called an **optimal policy**, i.e.,

$$\mathcal{R}^{\pi^*}(s) \geq \mathcal{R}^{\pi}(s), \quad s \in S$$

for all $\pi \in \Pi^{\mathrm{RH}}$.

Theorem: For an MDP $\{T, S, A, p(s'|s,a), r_t(s,a)\}$ with finite T, S, and A, there exists a deterministic Markovian policy, $\pi \in \Pi^{\mathrm{DM}}$, that is optimal.

Proof: Section 4.4 of *Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.*

The fundamental reason is that the immediate rewards and transition probabilities depend on the past only through the current state of the system. Due to the above result, we can restrict our search space to Π^{DM}, which is MUCH smaller than Π^{RH}.

For a given $\pi \in \Pi^{\mathrm{DM}}$, we can evaluate its reward using the following algorithm:

The Policy Evaluation Algorithm

1. Set $t = N$ and $V_N^{\pi}(s) = r_N(s)$ for all $s \in S$;
2. If $t = 0$, go to Step 5; otherwise go to Step 3;
3. Let $t \leftarrow t - 1$ and compute $V_t^{\pi}(s)$ for each $s \in S$ according to
$$V_t^{\pi}(s) = r_t(s, d_t(s)) + \sum_{s' \in S} p(s'|s, d_t(s)) V_{t+1}^{\pi}(s')$$
4. Return to Step 2;
5. Output $\mathcal{R}^{\pi}(s) = V_0^{\pi}(s)$.

Using the above algorithm, we can evaluate the reward of every $\pi \in \Pi^{\mathrm{DM}}$ (which is finite) and then select the one with the highest reward as π^*. But this is not a very efficient way for us to find an optimal policy, because we have many policies to evaluate [i.e., $(|A|^{|S|})^N$]. The following algorithm provides an efficient method for solving for an optimal policy:

The Dynamic Programming Algorithm

1. Set $t = N$ and $V_N(s) = r_N(s)$ for all $s \in S$;
2. If $t = 0$, go to Step 6; otherwise go to Step 3;
3. Let $t \leftarrow t - 1$ and compute $Q_t(s,a)$ for each $(s,a) \in S \times A$ and $V_t(s)$ for each $s \in S$ according to
$$Q_t(s,a) = r_t(s,a) + \sum_{s' \in S} p(s'|s,a) V_{t+1}(s')$$
$$V_t(s) = \max_{a \in A} Q_t(s,a)$$
4. Determine d_t^* according to
$$d_t^*(s) \in \operatorname*{argmax}_{a \in A} Q_t(s,a), \quad s \in S$$
5. Return to Step 2;
6. Output $\pi^* = \{d_0^*, d_1^*, ..., d_{N-1}^*\}$.

The equations in Step 3

$$V_t(s) = \max_{a \in A} Q_t(s,a) = \max_{a \in A} r_t(s,a) + \sum_{s' \in S} p(s'|s,a) V_{t+1}(s'), \quad s \in S$$

are called the **optimality equations** (also called the **Bellman equations**, named after Richard E. Bellman). These equations indicate that $V_t(s)$ represents the highest future reward (i.e., cumulative reward over $k = t, ..., N$) that we can get if we are in state s at time t. Therefore, $\mathcal{R}^{\pi^*}(s) = V_0(s)$ and the output of Step 6 is an optimal policy. The optimality equations act like a dynamic system of $V_t(s)$, but backward in time, with boundary condition $V_N(s) = r_N(s)$.

Theorem: Let V_t, $t = 0, ..., N$, be solutions to the optimality equations

$$V_t(s) = \max_{a \in A} Q_t(s,a) = \max_{a \in A} r_t(s,a) + \sum_{s' \in S} p(s'|s,a) V_{t+1}(s')$$

subject to boundary condition $V_N(s) = r_N(s)$, and $\pi^* = \{d_0^*, d_1^*, ..., d_{N-1}^*\}$ be a policy satisfying

$$r_t(s, d_t^*(s)) + \sum_{s' \in S} p(s'|s, d_t^*(s)) V_{t+1}(s') = \max_{a \in A} r_t(s,a) + \sum_{s' \in S} p(s'|s,a) V_{t+1}(s')$$

Then, π^* is an optimal policy such that

$$\mathcal{R}^{\pi^*}(s) \geq \mathcal{R}^{\pi}(s), \quad s \in S$$

for all $\pi \in \Pi^{\text{RH}}$.

Example:

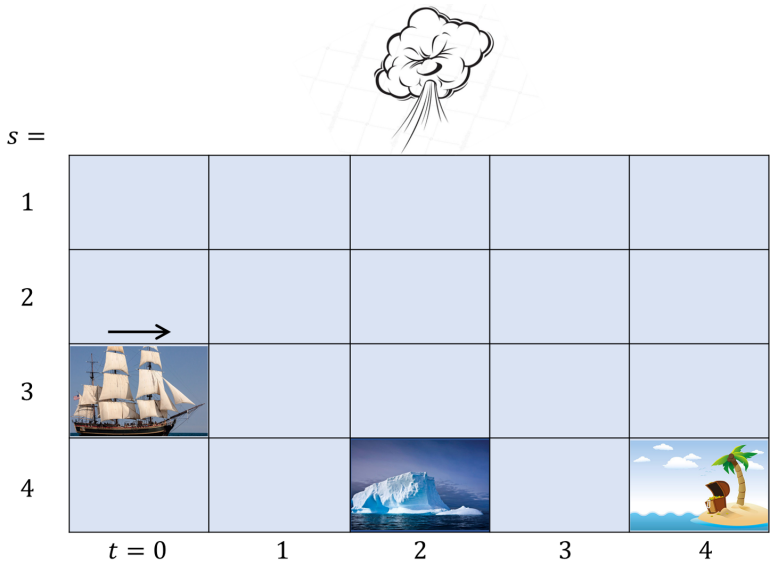

Figure 3: Example of a finite-horizon MDP problem.

Consider the finite-horizon MDP problem shown in the above figure with the following parameters:

Spaces:

$$T = \{0, 1, 2, 3, 4\}, \quad S = \{1, 2, 3, 4\}, \quad A = \{\rightarrow, \uparrow, \downarrow\}$$

Transition probabilities:

$$p(s\,|\,s,\rightarrow)=0.6,\quad p(s+1\,|\,s,\rightarrow)=0.4,\quad \forall s\in\{1,2,3\},\qquad p(4\,|\,4,\rightarrow)=1$$
$$p(s-1\,|\,s,\uparrow)=0.6,\quad p(s\,|\,s,\uparrow)=0.4,\quad \forall s\in\{2,3,4\},\qquad p(1\,|\,1,\uparrow)=1$$
$$p(s+1\,|\,s,\downarrow)=0.6,\quad p(s+2\,|\,s,\downarrow)=0.4,\quad \forall s\in\{1,2\},\qquad p(4\,|\,s',\downarrow)=1,\quad \forall s'\in\{3,4\}$$

Rewards:

$$r_2(4,a)=-10,\quad r_4(4)=10,\quad r_t(s,a)=0 \text{ otherwise}$$

Use the DP algorithm to solve this problem:

$t=4$:

$$V_4(s)=r_4(s)=0\quad \forall s\in\{1,2,3\},\quad V_4(4)=r_4(4)=10$$

$t=3$:

$$\begin{aligned}Q_3(s,\rightarrow)&=r_3(s,\rightarrow)+\sum_{s'\in S}p(s'|s,\rightarrow)V_4(s')=0+p(s\,|\,s,\rightarrow)V_4(s)+p(s+1\,|\,s,\rightarrow)V_4(s+1)\\&=p(s\,|\,s,\rightarrow)V_4(s)+p(s+1\,|\,s,\rightarrow)V_4(s+1)\\Q_3(s,\uparrow)&=r_3(s,\uparrow)+\sum_{s'\in S}p(s'|s,\uparrow)V_4(s')=0+p(s-1\,|\,s,\uparrow)V_4(s-1)+p(s\,|\,s,\uparrow)V_4(s)\\&=p(s-1\,|\,s,\uparrow)V_4(s-1)+p(s\,|\,s,\uparrow)V_4(s)\\Q_3(s,\downarrow)&=r_3(s,\downarrow)+\sum_{s'\in S}p(s'|s,\downarrow)V_4(s')=0+p(s+1\,|\,s,\downarrow)V_4(s+1)+p(s+2\,|\,s,\downarrow)V_4(s+2)\\&=p(s+1\,|\,s,\downarrow)V_4(s+1)+p(s+2\,|\,s,\downarrow)V_4(s+2)\quad \forall s\in\{1,2,3\}\end{aligned}$$

$$\begin{aligned}Q_3(1,\rightarrow)&=p(1\,|\,1,\rightarrow)V_4(1)+p(2\,|\,1,\rightarrow)V_4(2)=0+0=0\\Q_3(1,\uparrow)&=p(1\,|\,1,\uparrow)V_4(1)=0\\Q_3(1,\downarrow)&=p(2\,|\,1,\downarrow)V_4(2)+p(3\,|\,1,\downarrow)V_4(3)=0\\V_3(1)&=\max_{a\in A}Q_3(1,a)=0\qquad d_3^*(1)\in\operatorname{argmax}Q_3(1,a)=\{\rightarrow,\uparrow,\downarrow\}\end{aligned}$$

$$\begin{aligned}Q_3(2,\rightarrow)&=p(2\,|\,2,\rightarrow)V_4(2)+p(3\,|\,2,\rightarrow)V_4(3)=0+0=0\\Q_3(2,\uparrow)&=p(1\,|\,2,\uparrow)V_4(1)+p(2\,|\,2,\uparrow)V_4(2)=0+0=0\\Q_3(2,\downarrow)&=p(3\,|\,2,\downarrow)V_4(3)+p(4\,|\,2,\downarrow)V_4(4)=0+0.4\times 10=4\\V_3(2)&=\max_{a\in A}Q_3(2,a)=4\qquad d_3^*(2)\in\operatorname{argmax}Q_3(2,a)=\{\downarrow\}\end{aligned}$$

$$\begin{aligned}Q_3(3,\rightarrow)&=p(3\,|\,3,\rightarrow)V_4(3)+p(4\,|\,3,\rightarrow)V_4(4)=0+0.4\times 10=4\\Q_3(3,\uparrow)&=p(2\,|\,3,\uparrow)V_4(2)+p(3\,|\,3,\uparrow)V_4(3)=0+0=0\\Q_3(3,\downarrow)&=p(4\,|\,3,\downarrow)V_4(4)=1\times 10=10\\V_3(3)&=\max_{a\in A}Q_3(3,a)=10\qquad d_3^*(3)\in\operatorname{argmax}Q_3(3,a)=\{\downarrow\}\end{aligned}$$

$$\begin{aligned}Q_3(4,\rightarrow)&=p(4\,|\,4,\rightarrow)V_4(4)=1\times 10=10\\Q_3(4,\uparrow)&=p(3\,|\,4,\uparrow)V_4(3)+p(4\,|\,4,\uparrow)V_4(4)=0+0.4\times 10=4\\Q_3(4,\downarrow)&=p(4\,|\,4,\downarrow)V_4(4)=1\times 10=10\\V_3(4)&=\max_{a\in A}Q_3(4,a)=10\qquad d_3^*(4)\in\operatorname{argmax}Q_3(4,a)=\{\rightarrow,\downarrow\}\end{aligned}$$

$t = 2$:

$$
\begin{aligned}
Q_2(s,\rightarrow) &= r_2(s,\rightarrow) + p(s \mid s,\rightarrow)V_3(s) + p(s+1 \mid s,\rightarrow)V_3(s+1) \\
Q_2(s,\uparrow) &= r_2(s,\uparrow) + p(s-1 \mid s,\uparrow)V_3(s-1) + p(s \mid s,\uparrow)V_3(s) \\
Q_2(s,\downarrow) &= r_2(s,\downarrow) + p(s+1 \mid s,\downarrow)V_3(s+1) + p(s+2 \mid s,\downarrow)V_3(s+2) \quad \forall s \in \{1,2,3\}
\end{aligned}
$$

$$
\begin{aligned}
Q_2(1,\rightarrow) &= p(1 \mid 1,\rightarrow)V_3(1) + p(2 \mid 1,\rightarrow)V_3(2) = 0 + 0.4 \times 4 = 1.6 \\
Q_2(1,\uparrow) &= p(1 \mid 1,\uparrow)V_3(1) = 0 \\
Q_2(1,\downarrow) &= p(2 \mid 1,\downarrow)V_3(2) + p(3 \mid 1,\downarrow)V_3(3) = 0.6 \times 4 + 0.4 \times 10 = 6.4 \\
V_2(1) &= \max_{a \in A} Q_2(1,a) = 6.4 \qquad d_2^*(1) \in \operatorname{argmax} Q_2(1,a) = \{\downarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_2(2,\rightarrow) &= p(2 \mid 2,\rightarrow)V_3(2) + p(3 \mid 2,\rightarrow)V_3(3) = 0.6 \times 4 + 0.4 \times 10 = 6.4 \\
Q_2(2,\uparrow) &= p(1 \mid 2,\uparrow)V_3(1) + p(2 \mid 2,\uparrow)V_3(2) = 0 + 0.4 \times 4 = 1.6 \\
Q_2(2,\downarrow) &= p(3 \mid 2,\downarrow)V_3(3) + p(4 \mid 2,\downarrow)V_3(4) = 0.6 \times 10 + 0.4 \times 10 = 10 \\
V_2(2) &= \max_{a \in A} Q_2(2,a) = 10 \qquad d_2^*(2) \in \operatorname{argmax} Q_2(2,a) = \{\downarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_2(3,\rightarrow) &= p(3 \mid 3,\rightarrow)V_3(3) + p(4 \mid 3,\rightarrow)V_3(4) = 0.6 \times 10 + 0.4 \times 10 = 10 \\
Q_2(3,\uparrow) &= p(2 \mid 3,\uparrow)V_3(2) + p(3 \mid 3,\uparrow)V_3(3) = 0.6 \times 4 + 0.4 \times 10 = 6.4 \\
Q_2(3,\downarrow) &= p(4 \mid 3,\downarrow)V_3(4) = 1 \times 10 = 10 \\
V_2(3) &= \max_{a \in A} Q_2(3,a) = 10 \qquad d_2^*(3) \in \operatorname{argmax} Q_2(3,a) = \{\rightarrow,\downarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_2(4,\rightarrow) &= r_2(4,\rightarrow) + p(4 \mid 4,\rightarrow)V_3(4) = -10 + 1 \times 10 = 0 \\
Q_2(4,\uparrow) &= r_2(4,\uparrow) + p(3 \mid 4,\uparrow)V_3(3) + p(4 \mid 4,\uparrow)V_3(4) = -10 + 10 = 0 \\
Q_2(4,\downarrow) &= r_2(4,\downarrow) + p(4 \mid 4,\downarrow)V_3(4) = -10 + 1 \times 10 = 0 \\
V_2(4) &= \max_{a \in A} Q_2(4,a) = 0 \qquad d_2^*(4) \in \operatorname{argmax} Q_2(4,a) = \{\rightarrow,\uparrow,\downarrow\}
\end{aligned}
$$

$t = 1$:

$$
\begin{aligned}
Q_1(1,\rightarrow) &= p(1 \mid 1,\rightarrow)V_2(1) + p(2 \mid 1,\rightarrow)V_2(2) = 0.6 \times 6.4 + 0.4 \times 10 = 7.84 \\
Q_1(1,\uparrow) &= p(1 \mid 1,\uparrow)V_2(1) = 1 \times 6.4 = 6.4 \\
Q_1(1,\downarrow) &= p(2 \mid 1,\downarrow)V_2(2) + p(3 \mid 1,\downarrow)V_2(3) = 10 \\
V_1(1) &= \max_{a \in A} Q_1(1,a) = 10 \qquad d_1^*(1) \in \operatorname{argmax} Q_1(1,a) = \{\downarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_1(2,\rightarrow) &= p(2 \mid 2,\rightarrow)V_2(2) + p(3 \mid 2,\rightarrow)V_2(3) = 10 \\
Q_1(2,\uparrow) &= p(1 \mid 2,\uparrow)V_2(1) + p(2 \mid 2,\uparrow)V_2(2) = 0.6 \times 6.4 + 0.4 \times 10 = 7.84 \\
Q_1(2,\downarrow) &= p(3 \mid 2,\downarrow)V_2(3) + p(4 \mid 2,\downarrow)V_2(4) = 0.6 \times 10 + 0.4 \times 0 = 6 \\
V_1(2) &= \max_{a \in A} Q_1(2,a) = 10 \qquad d_1^*(2) \in \operatorname{argmax} Q_1(2,a) = \{\rightarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_1(3,\rightarrow) &= p(3 \mid 3,\rightarrow)V_2(3) + p(4 \mid 3,\rightarrow)V_2(4) = 0.6 \times 10 + 0.4 \times 0 = 6 \\
Q_1(3,\uparrow) &= p(2 \mid 3,\uparrow)V_2(2) + p(3 \mid 3,\uparrow)V_2(3) = 10 \\
Q_1(3,\downarrow) &= p(4 \mid 3,\downarrow)V_2(4) = 0 \\
V_1(3) &= \max_{a \in A} Q_1(3,a) = 10 \qquad d_1^*(3) \in \operatorname{argmax} Q_1(3,a) = \{\uparrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_1(4,\rightarrow) &= p(4\,|\,4,\rightarrow)V_2(4) = 0 \\
Q_1(4,\uparrow) &= p(3\,|\,4,\uparrow)V_2(3) + p(4\,|\,4,\uparrow)V_2(4) = 0.6\times 10 + 0.4\times 0 = 6 \\
Q_1(4,\downarrow) &= p(4\,|\,4,\downarrow)V_2(4) = 0 \\
V_1(4) &= \max_{a\in A} Q_1(4,a) = 6 \qquad d_1^*(4) \in \operatorname{argmax} Q_1(4,a) = \{\uparrow\}
\end{aligned}
$$

$t = 0$:

$$
\begin{aligned}
Q_0(1,\rightarrow) &= p(1\,|\,1,\rightarrow)V_1(1) + p(2\,|\,1,\rightarrow)V_1(2) = 10 \\
Q_0(1,\uparrow) &= p(1\,|\,1,\uparrow)V_1(1) = 10 \\
Q_0(1,\downarrow) &= p(2\,|\,1,\downarrow)V_1(2) + p(3\,|\,1,\downarrow)V_1(3) = 10 \\
V_0(1) &= \max_{a\in A} Q_0(1,a) = 10 \qquad d_0^*(1) \in \operatorname{argmax} Q_0(1,a) = \{\rightarrow,\uparrow,\downarrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_0(2,\rightarrow) &= p(2\,|\,2,\rightarrow)V_1(2) + p(3\,|\,2,\rightarrow)V_1(3) = 10 \\
Q_0(2,\uparrow) &= p(1\,|\,2,\uparrow)V_1(1) + p(2\,|\,2,\uparrow)V_1(2) = 10 \\
Q_0(2,\downarrow) &= p(3\,|\,2,\downarrow)V_1(3) + p(4\,|\,2,\downarrow)V_1(4) = 0.6\times 10 + 0.4\times 6 = 8.4 \\
V_0(2) &= \max_{a\in A} Q_0(2,a) = 10 \qquad d_0^*(2) \in \operatorname{argmax} Q_0(2,a) = \{\rightarrow,\uparrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_0(3,\rightarrow) &= p(3\,|\,3,\rightarrow)V_1(3) + p(4\,|\,3,\rightarrow)V_1(4) = 0.6\times 10 + 0.4\times 6 = 8.4 \\
Q_0(3,\uparrow) &= p(2\,|\,3,\uparrow)V_1(2) + p(3\,|\,3,\uparrow)V_1(3) = 10 \\
Q_0(3,\downarrow) &= p(4\,|\,3,\downarrow)V_1(4) = 1\times 6 = 6 \\
V_0(3) &= \max_{a\in A} Q_0(3,a) = 10 \qquad d_0^*(3) \in \operatorname{argmax} Q_0(3,a) = \{\uparrow\}
\end{aligned}
$$

$$
\begin{aligned}
Q_0(4,\rightarrow) &= p(4\,|\,4,\rightarrow)V_1(4) = 1\times 6 = 6 \\
Q_0(4,\uparrow) &= p(3\,|\,4,\uparrow)V_1(3) + p(4\,|\,4,\uparrow)V_1(4) = 0.6\times 10 + 0.4\times 6 = 8.4 \\
Q_0(4,\downarrow) &= p(4\,|\,4,\downarrow)V_1(4) = 1\times 6 = 6 \\
V_0(4) &= \max_{a\in A} Q_0(4,a) = 8.4 \qquad d_0^*(4) \in \operatorname{argmax} Q_0(4,a) = \{\uparrow\}
\end{aligned}
$$

Solution:

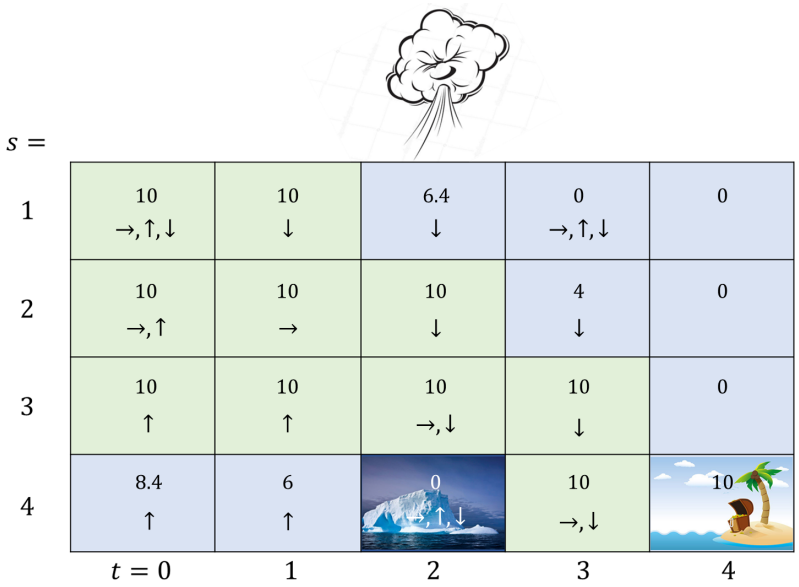

Figure 4: Solution to the finite-horizon MDP example.

11 Infinite-horizon MDP and Value/Policy Iterations

An infinite-horizon MDP is an MDP $\{T, S, A, p(s'|s,a), r_t(s,a)\}$ with $T = \mathbb{Z}_+ = \{0, 1, 2, ...\}$. A policy for an infinite-horizon MDP is an infinite sequence of decision rules, $\pi = \{d_0, d_1, d_2, ...\}$. Similar to the finite-horizon case, most generally, a policy can be randomized history-dependent, $\pi \in \Pi^{\text{RH}}$, or be deterministic Markovian, $\pi \in \Pi^{\text{DM}}$.

We focus on problems with a time-independent reward function $r_t(s,a) = r(s,a)$. We may pursue a policy $\pi \in \Pi^{\text{RH}}$ that maximizes the expected total reward,

$$\mathcal{R}^\pi(s) = \lim_{N\to\infty} \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{N-1} r(s_t, a_t) \right\} \quad \overset{\text{when lim exists}}{=} \quad \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{\infty} r(s_t, a_t) \right\}$$

However, the limit does not always exist [indeed, often does not exist]. For instance, if $r(s,a) > 0$ for all $(s,a) \in S \times A$, we are constantly adding positive numbers that are at least $r_{\min} = \min_{s,a} r(s,a) > 0$ to the sum as N increases. Then, the sum goes to infinity as $N \to \infty$ and the limit does not exist.

Therefore, we need to consider some other performance criteria. The most popular choices are the following two:

- The expected total discounted reward criterion:

$$\mathcal{R}_\lambda^\pi(s) = \lim_{N\to\infty} \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{N-1} \lambda^t\, r(s_t, a_t) \right\} = \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{\infty} \lambda^t\, r(s_t, a_t) \right\}$$

  where $\lambda \in [0, 1)$ is a **discount factor**. The limit exists when $\sup_{s,a} |r(s,a)| = M < \infty$. Therefore, for finite-space MDP, the limit always exists.

- The average reward criterion:

$$\bar{\mathcal{R}}^\pi(s) = \lim_{N\to\infty} \frac{1}{N} \mathbb{E}_s^\pi \left\{ \sum_{t=0}^{N-1} r(s_t, a_t) \right\}$$

  The limit may still not exist. Additional assumptions are needed to guarantee existence of the limit.

Most MDP solution techniques (including most reinforcement learning techniques) are for the expected total discounted reward criterion. Therefore, in what follows we focus on this criterion and all results apply to this criterion.

Recall that a policy π^* is **optimal** if

$$\mathcal{R}_\lambda^{\pi^*}(s) \geq \mathcal{R}_\lambda^\pi(s), \quad s \in S$$

for all $\pi \in \Pi^{\text{RH}}$.

Theorem: For an infinite-horizon MDP with finite S and A, there exists a deterministic, Markovian, and stationary policy, i.e., $\pi \in \Pi^{\text{DM}}$ and $\pi = \{d, d, ...\}$, that is optimal.

Proof: Section 6.2 of *Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.*

Indeed, we can drop "Markovian" from the theorem statement, because a stationary policy, which is a policy that has the same decision rule at every decision epoch, must be Markovian [since $d_0 = d$ uses only the initial state to determine the action].

Based on the above theorem, the goal reduces to computing a deterministic stationary optimal policy.

Recall the optimality/Bellman equations in the finite-horizon case,

$$V_t(s) = \max_{a\in A} Q_t(s,a) = \max_{a\in A}\left[r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V_{t+1}(s')\right], \quad s \in S$$

Passing to the limit, we obtain the following equations

$$V(s) = \max_{a\in A} Q(s,a) = \max_{a\in A}\left[r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V(s')\right], \quad s \in S$$

where $V(s) = \lim_{t\to\infty} V_t(s)$. These are called the **optimality/Bellman equations** in the infinite-horizon case. If we treat $V(s)$, for each $s \in S$, as a variable, then we have n equations for n variables, where $n = \text{card}(S)$.

Theorem: For an infinite-horizon MDP with finite S and A, the following hold:

1. There is a unique $V_\lambda^* : S \to \mathbb{R}$ satisfying the optimality/Bellman equations;
2. A policy $\pi^* \in \Pi^{\text{RH}}$ is optimal if and only if $\mathcal{R}_\lambda^{\pi^*}(s) = V_\lambda^*(s)$ for all $s \in S$;
3. If $d^* : S \to A$ satisfies

$$d^*(s) \in \operatorname*{argmax}_{a\in A} Q_\lambda^*(s,a) = \operatorname*{argmax}_{a\in A}\left[r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V_\lambda^*(s')\right], \quad \forall s \in S$$

   then $\pi^* = \{d^*, d^*, ...\}$ is an optimal policy (which is deterministic and stationary).

Based on the above theorem, the goal of computing an optimal policy reduces to solving the system of optimality/Bellman equations. We then introduce two algorithms to achieve this goal, called the **value iteration** algorithm and the **policy iteration** algorithm.

The Value Iteration Algorithm

1. Initialize V^0 (e.g., $V^0(s) = 0$ for all $s \in S$), specify $\varepsilon > 0$, and set $n = 0$;
2. For each $(s,a) \in S \times A$, compute $Q^n(s,a)$ and $V^{n+1}(s)$ according to

$$Q^n(s,a) = r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V^n(s')$$
$$V^{n+1}(s) = \max_{a\in A} Q^n(s,a)$$

3. If $\|V^{n+1} - V^n\|_\infty < \varepsilon$, go to Step 4; otherwise $n \leftarrow n+1$ and return to Step 2;
4. Choose d_ε^* according to

$$d_\varepsilon^*(s) \in \operatorname*{argmax}_{a\in A} Q^{n+1}(s,a) = \operatorname*{argmax}_{a\in A}\left[r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V^{n+1}(s')\right], \quad \forall s \in S$$

5. Set $\pi_\varepsilon^* = \{d_\varepsilon^*, d_\varepsilon^*, ...\}$.

Propositions:

1. **Linear convergence at rate λ**:

$$\|V^n - V_\lambda^*\|_\infty \leq \frac{\lambda^n}{1-\lambda} \|V^1 - V^0\|_\infty$$

   This is called linear convergence because the bound on the RHS is a linear system $b^{n+1} = \lambda b^n$.

2. **Monotone convergence**:

   If $V^{n+1} \leq (\geq)\ V^n$, then $V^{n+m+1} \leq (\geq)\ V^{n+m}$ for all $m \geq 0$. [Here, $\leq (\geq)$ is pointwise.]

   Special case: If $r(s,a) \leq (\geq)\ 0$ for all $(s,a) \in S \times A$ and $V^0(s) = 0$ for all $s \in S$, then $V^1 \leq (\geq)\ V^0$ and the convergence is monotone.

3. **ε-optimality**:

   Let $d^{n+1}(s) \in \text{argmax}_{a \in A}\ Q^{n+1}(s,a)$ and $\pi^{n+1} = \{d^{n+1}, d^{n+1}, ...\}$, then

$$\|\mathcal{R}_\lambda^{\pi^{n+1}} - V_\lambda^*\|_\infty \leq \frac{2\lambda}{1-\lambda} \|V^{n+1} - V^n\|_\infty$$

Let $\pi = \{d, d, ...\}$ be a deterministic stationary policy. We now use π and d interchangably. Recall that in the policy evaluation algorithm of the finite-horizon case, we used the following recursive equations to compute the values of a policy

$$V_t^\pi(s) = r(s, d(s)) + \lambda \sum_{s' \in S} p(s'|s, d(s)) V_{t+1}^\pi(s')$$

and $\mathcal{R}_\lambda^\pi(s) = V_0^\pi(s)$. Passing to the limit, we obtain the following equations

$$V^\pi(s) = r(s, d(s)) + \lambda \sum_{s' \in S} p(s'|s, d(s)) V^\pi(s')$$

and $\mathcal{R}_\lambda^\pi(s) = V^\pi(s) = V_d(s)$. These equations can be used to evaluate a policy.

Let

$$V_d = \begin{bmatrix} V_d(s_1) \\ \vdots \\ V_d(s_n) \end{bmatrix}, \quad r_d = \begin{bmatrix} r(s_1, d(s_1)) \\ \vdots \\ r(s_n, d(s_n)) \end{bmatrix}, \quad \mathbf{P}_d = \begin{bmatrix} p(s_1|s_1, d(s_1)) & \cdots & p(s_1|s_n, d(s_n)) \\ \vdots & \ddots & \vdots \\ p(s_n|s_1, d(s_1)) & \cdots & p(s_n|s_n, d(s_n)) \end{bmatrix}$$

Then, the above policy evaluation equations can be written as

$$V_d = r_d + \lambda \mathbf{P}_d^\top V_d$$

or

$$(I - \lambda \mathbf{P}_d^\top) V_d = r_d$$

which is a system of linear equations for $V_d \in \mathbb{R}^n$.

The Policy Iteration Algorithm

1. Initialize a decision rule $d^0 : S \to A$ and set $n = 0$;
2. (Policy evaluation) Obtain $V^n = V_{d^n}$ by solving
$$(I - \lambda \mathbf{P}_{d^n}^\top)V = r_{d^n}$$
3. (Policy improvement) Choose d^{n+1} to satisfy
$$d^{n+1}(s) \in \underset{a\in A}{\text{argmax}}\ Q^n(s,a) = \underset{a\in A}{\text{argmax}} \left[r(s,a) + \lambda \sum_{s'\in S} p(s'|s,a)V^n(s') \right], \quad \forall s \in S$$
setting $d^{n+1} = d^n$ if possible.
4. If $d^{n+1} = d^n$, set $d^* = d^n$ and stop; otherwise $n \leftarrow n + 1$ and return to Step 2.

Propositions:

1. **Finite convergence**:
For finite S and A, the algorithm terminates in a finite number of iterations, with a solution d^* ($\pi^* = \{d^*, d^*, ...\}$) that is an optimal policy.
Finite convergence is the fastest convergence.
2. **Monotone convergence**: $V^{n+1} = V_{d^{n+1}} \geq V^n = V_{d^n}$.
The policy is monotonically improving.

Remarks:

1. Although the policy iteration algorithm converges to the optimal value function and an optimal policy in a less number of iterations (finite) than the value iteration algorithm (possibly infinite), one policy iteration is more computationally demanding than one value iteration, due to the need for solving a system of linear equations. Therefore, the value iteration algorithm is more popular.
2. In the policy iteration algorithm, $V^n = V_{d^n}$ is the value function of the policy d^n; in the value iteration algorithm, V^n is not the value function of d^n, where $d^n(s) \in \text{argmax}_{a\in A}\ Q^n(s,a)$. This is because given d^n, there is a unique V_{d^n}, determined by the policy evaluation equation; but different V may lead to the same policy $d(s) \in \text{argmax}_{a\in A}\ Q(s,a)$.

12 A practical procedure to solve a control problem using MDP

Suppose we have the following control system

$$\dot{x} = f_c(x, u, w)$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control input vector, and $w \in \mathbb{R}^d$ is a disturbance input, taking values according to some continuous distribution with pdf $\phi(w)$.

The first step is to convert the system model into discrete-time

$$x^+ = f_d(x, u, w)$$

using a discretization method, such as the forward Euler method

$$x^+ = f_d(x, u, w) = x + f_c(x, u, w)\Delta t$$

where Δt is the sampling interval.

Then, we estimate a normal (bounded) operating range of the system, $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$, and select a bounded set for w, $\mathcal{W} \subset \mathbb{R}^d$, that contains the most probability mass of $\phi(w)$.

Then, we discretize/grid $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{W}$ with certain resolutions/step sizes Δx, Δu, and Δw, to get the discretized/finite spaces $\hat{\mathcal{X}}$, $\hat{\mathcal{U}}$, and $\hat{\mathcal{W}}$. We name the grid points $\{x_i\}_{i=1}^N$, $\{u_j\}_{j=1}^M$, and $\{w_k\}_{k=1}^D$.

We compute a discrete probability distribution of $\{w_k\}_{k=1}^D$ according to

$$\Phi(w_k) = \mathbb{P}(w = w_k) = \frac{\phi(w_k)}{\sum_{k'=1}^D \phi(w_{k'})}$$

We estimate a discrete mapping $\hat{f} : \hat{\mathcal{X}} \times \hat{\mathcal{U}} \times \hat{\mathcal{W}} \to \hat{\mathcal{X}}$ according to

$$\hat{x}^+ = \hat{f}(\hat{x}, \hat{u}, \hat{w}) \in \operatorname*{argmax}_{x_i \in \hat{\mathcal{X}}} \|x_i - x^+\| = \operatorname*{argmax}_{x_i \in \hat{\mathcal{X}}} \|x_i - f_d(\hat{x}, \hat{u}, \hat{w})\|$$

where $\hat{x}$, $\hat{u}$, and $\hat{w}$ are points on the grids. We now have a discrete approximation of the original continuous system.

The last step is to represent the obtained discrete system as an MDP, according to

$$p(x' \,|\, x, u) = \sum_{k=1}^{D} \mathbb{1}_{x'}(\hat{f}(x, u, w_k))\Phi(w_k)$$

where $\mathbb{1}_{x'}(\cdot)$ is an indicator function, taking 1 if $(\cdot) = x'$ and 0 otherwise. The transition probability from $(x, u) \in \hat{\mathcal{X}} \times \hat{\mathcal{U}}$ to $x' \in \hat{\mathcal{X}}$ is the probability mass of $w \in \hat{\mathcal{W}}$ such that $x' = \hat{f}(x, u, w)$.

Then, according to the control objective, we can design reward (cost) functions $r_t(x, u)$ or $r(x, u)$, e.g.,

$$r(x, u) = -\left(x^\top Q x + u^\top R u\right)$$

and use stochastic dynamic programming or value/policy iterations to solve for an optimal policy $\pi^* = \{d_0^*, d_1^*, ...\}$.

The optimal decision rules $d_t^* : \hat{\mathcal{X}} \to \hat{\mathcal{U}}$ can be represented as a lookup table. Then, we can obtain approximate optimal controls $u_t^* : \mathcal{X} \to \mathcal{U}$ of the continuous-space discrete-time system f_d using interpolation.

Finally, we apply controls $u_t^* : \mathcal{X} \to \mathcal{U}$ to the original continuous-time system f_c using zero-order hold (i.e., piecewise-constant control).

Module IV: Reinforcement Learning

Given an infinite-horizon MDP with an expected total discounted reward criterion, we can use the value/policy iteration algorithm to solve for V^* or Q^* when we know the transition probabilities $p(s'|s,a)$. However, in many problems, even though we know that the underlying dynamics of $(s,a) \to s'$ are Markovian, we may not know the transition probabilities $p(s'|s,a)$ explicitly. For instance, the dynamics may involve interactions of multiple sub-systems and there may be multiple uncertainty sources. We may know or be able to model the stochastic characteristics of each source separately. But their cumulative effect may be difficult to calculate. In such a case, we can use **Reinforcement Learning (RL)**. According to Wikipedia:

Reinforcement learning is concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward, where the environment is typically modeled as a Markov decision process (MDP). Many reinforcement learning algorithms use dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

13 Q-Learning

Q-learning is a classical RL algorithm (first proposed in Watkins, 1989) and serves as the foundation of many more advanced RL algorithms. The Q-learning algorithm, as its name suggests, aims to learn the optimal Q function from data (without relying on an explicit MDP model).

According to a previous theorem, the problem of computing an optimal policy for an infinite-horizon MDP with an expected total discounted reward criterion reduces to the problem of solving the following optimality/Bellman equation

$$V(s) = \max_{a \in A} r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V(s')$$

According to the same theorem, the above equation has a unique solution V^*, called the optimal value function. With V^*, we can obtain an optimal policy $\pi^* = \{d^*, d^*, ...\}$ according to

$$d^*(s) \in \operatorname*{argmax}_{a \in A} r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V^*(s')$$

If we define

$$Q^*(s,a) = r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V^*(s')$$

called the optimal state-action value function, or the optimal Q function. Then the above equations can be equivalently expressed as

$$\begin{aligned} V^*(s) &= \max_{a \in A} Q^*(s,a) \\ d^*(s) &\in \operatorname*{argmax}_{a \in A} Q^*(s,a) \\ Q^*(s,a) &= r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) \left(\max_{a' \in A} Q^*(s',a') \right) \end{aligned}$$

Due to the uniqueness of V^*, the third equation above has a unique solution Q^*.

The Q-learning algorithm is based on the following simple update equation,

$$Q_n(s,a) = r(s,a) + \lambda \left(\max_{a' \in A} Q_{n-1}(s',a') \right)$$

where (s,a,s') is a data point of (previous state, previous action, next state)-triple.

Suppose we have a large amount of trajectory data generated by the underlying MDP. Let $N(s,a)$ denote the number of data points with previous state $= s$ and previous action $= a$ and $N(s,a,s')$ the number of data points with previous state $= s$, previous action $= a$, and next state $= s'$. Then, we have

$$\sum_{s' \in S} N(s,a,s') = N(s,a), \quad \frac{N(s,a,s')}{N(s,a)} \approx p(s'|s,a)$$

We now add up the equations for updating $Q(s,a)$,

$$\sum_{n=1}^{N(s,a)} Q_n(s,a) = \sum_{n=1}^{N(s,a)} \left[r(s,a) + \lambda \left(\max_{a' \in A} Q_{n-1}(s',a') \right) \right] = N(s,a)\, r(s,a) + \lambda \sum_{n=1}^{N(s,a)} \left(\max_{a' \in A} Q_{n-1}(s',a') \right)$$

Assume that $Q_n(s,a) = Q'(s,a)$ for all n. Then, the above equation reduces to

$$\begin{aligned}
\sum_{n=1}^{N(s,a)} Q'(s,a) &= N(s,a)\, r(s,a) + \lambda \sum_{n=1}^{N(s,a)} \left(\max_{a' \in A} Q'(s',a') \right) \\
N(s,a)\, Q'(s,a) &= N(s,a)\, r(s,a) + \lambda \sum_{s' \in S} N(s,a,s') \left(\max_{a' \in A} Q'(s',a') \right) \\
Q'(s,a) &= r(s,a) + \lambda \sum_{s' \in S} \frac{N(s,a,s')}{N(s,a)} \left(\max_{a' \in A} Q'(s',a') \right) \\
Q'(s,a) &\approx r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) \left(\max_{a' \in A} Q'(s',a') \right)
\end{aligned}$$

which is the optimality equation. This explains how the update equation (which does not involve transition probabilities) approximates the optimality equation (which involves transition probabilities) through averaging over data. This also implies that if $Q_n(s,a)$ converges, it converges to the solution of the optimality equation $Q^*(s,a)$.

The update equation can be generalized to

$$\begin{aligned}
Q_n(s,a) &= (1-\alpha_n) Q_{n-1}(s,a) + \alpha_n \left[r(s,a) + \lambda \left(\max_{a' \in A} Q_{n-1}(s',a') \right) \right] \\
&= Q_{n-1}(s,a) + \alpha_n \underbrace{\left[r(s,a) + \lambda \left(\max_{a' \in A} Q_{n-1}(s',a') \right) - Q_{n-1}(s,a) \right]}_{= \text{temporal difference (TD)}}
\end{aligned}$$

where $\alpha_n \in [0,1]$ is a learning rate. Based on the second line above, sometimes the learning objective is said to be making the TD error zero

$$\text{TD error} = \underbrace{r(s,a) + \lambda \left(\max_{a' \in A} Q_{n-1}(s',a') \right)}_{= \text{TD target}} - Q_{n-1}(s,a)$$

Theorem (Watkins & Dayan, 1992): Let $n^i(s,a)$ denote the index of the ith time that action a is applied in state s. Then, if

$$\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty, \quad \sum_{i=1}^{\infty} (\alpha_{n^i(s,a)})^2 < \infty, \quad \forall\, (s,a) \in S \times A$$

then $Q_n(s,a) \to Q^*(s,a)$ as $n \to \infty$ for all $(s,a) \in S \times A$ with probability 1.

The above theorem reveals an important fact: In order for $Q_n(s,a)$ to converge to $Q^*(s,a)$, every state-action pair (s,a) in $S \times A$ must be visited infinitely often. This can be interpreted as: In order to evaluate and identify an optimal action among all actions in each state, the RL agent must **explore** every action and every state.

A classical exploration strategy is called ε**-greedy**: During data collection and learning, at each decision epoch/time step t, the RL agent has a large probability of $1-\varepsilon$ to take an action that is optimal according to the latest Q-value estimate, $a_t \in \operatorname{argmax}_{a\in A} Q_t(s_t,a)$, and it has a small probability of ε to take a random action in A.

Algorithm 4 The Q-Learning Algorithm (Abstract)

Initialize $Q_0(s,a)$ for all $(s,a) \in S \times A$ and state s_0;
for $t = 0, 1, ...$ **do**
  Select an action a_t according to

$$a_t \in \begin{cases} \operatorname{argmax}_{a\in A} Q_t(s_t,a) & \text{with probability } 1-\varepsilon \text{ (exploitation)} \\ \text{random action in } A & \text{with probability } \varepsilon \text{ (exploration)} \end{cases}$$

  Perform a_t, observe reward $r_t = r(s_t,a_t)$ and next state s_{t+1};
  Update Q-values according to

$$Q_{t+1}(s,a) = \begin{cases} (1-\alpha_t)Q_t(s,a) + \alpha_t\big[r_t + \lambda \max_{a'\in A} Q_t(s_{t+1},a')\big] & \text{if } (s,a) = (s_t,a_t) \\ Q_t(s,a) & \text{otherwise} \end{cases}$$

end for

Algorithm 5 The Q-Learning Algorithm (Practical)

Initialize $Q_0(s,a)$ for all $(s,a) \in S \times A$ and $\bar{r} = 0$;
for $n = 0, 1, ..., n_{\max} - 1$ (episode) **do**
  Randomly initialize state $s_0 \in S$;
  for $t = 0, 1, ..., t_{\max} - 1$ (time) **do**
    Select an action a_t according to

$$a_t \in \begin{cases} \operatorname{argmax}_{a\in A} Q_{nt_{\max}+t}(s_t,a) & \text{with probability } 1-\varepsilon \\ \text{random action in } A & \text{with probability } \varepsilon \end{cases}$$

    Perform a_t, observe reward $r_t = r(s_t,a_t)$ and next state s_{t+1};
    Update Q-values according to

$$Q_{nt_{\max}+t+1}(s,a) = \begin{cases} (1-\alpha)Q_{nt_{\max}+t}(s,a) + \alpha\big[r_t + \lambda \max_{a'\in A} Q_{nt_{\max}+t}(s_{t+1},a')\big] & \text{if } (s,a) = (s_t,a_t) \\ Q_{nt_{\max}+t}(s,a) & \text{otherwise} \end{cases}$$

    Compute $\bar{r} \leftarrow ((nt_{\max}+t)\bar{r} + r_t)/(nt_{\max}+t+1)$ (used to monitor convergence);
  end for
end for
Output $d^*(s) \in \operatorname{argmax}_{a\in A} Q_{n_{\max}t_{\max}}(s,a)$.

Coverage of the state space is improved by periodically and randomly reset the state.

Another exploration strategy called **softmax**:

$$\mathbb{P}(a_t = a|s_t) = \frac{\exp(\beta Q_t(s_t, a))}{\sum_{a' \in A} \exp(\beta Q_t(s_t, a'))}$$

where $\beta > 0$ is a shaping parameter ($T = 1/\beta$ is sometimes called the **temperature**, and high temperature $\rightarrow$ softer).

14 Representation of Q-function

The Q-learning algorithm updates the estimates of optimal Q-values using the equation

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r(s, a) + \lambda \left(\max_{a' \in A} Q(s', a') \right) \right]$$

where $Q(s, a)$ can be viewed as a function $Q : S \times A \rightarrow \mathbb{R}$. We need to represent the Q-function, due to the following two reasons:

- On the right-hand side, we need to calculate the (un-updated) Q-values for (s, a) and (s', a'), $\forall a' \in A$. For this, we need an (un-updated) representation of the Q-function so that we can evaluate it at (s, a) and (s', a').
- On the left-hand side, we obtain an updated Q-value for (s, a). We need to update our Q-function representation accordingly so that we can use the updated Q-function representation to calculate the right-hand side for future learning steps.

When S and A are finite, one way to represent the Q-function is in the form of a table, called a **tabular representation**, i.e.,

Q-table	a_1	a_2	$\cdots$	a_M
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$		$Q(s_1, a_M)$
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$		$Q(s_2, a_M)$
$\vdots$			$\ddots$	
s_N	$Q(s_N, a_1)$	$Q(s_N, a_2)$		$Q(s_N, a_M)$

Now we assume $s \in S \subset \mathbb{R}^n$ and $a \in A \subset \mathbb{R}^m$, i.e., a state is an n-dimensional real vector and an action is an m-dimensional real vector. Then, if the Q-function is a linear function of (s, a), we can represent it as

$$Q(s, a) = a^\top s + b^\top a$$

where $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are coefficients/parameters. Compared to a tabular representation, such a linear representation has the following advantages:

- Evaluation of this function at (s, a) and (s', a') are very easy.
- Using a tabular representation, we need to learn the Q-values of $M \times N$ cells, which can be viewed as $M \times N$ parameters. Using a linear representation, we only need to learn $m + n$ parameters, which are the entries of a and b. We may need less data to learn their values (for the optimal Q-function).
- Such a linear representation does not care whether S and A are finite, which opens up the opportunity to extend Q-learning from finite-space problems to continuous-space problems.

However, we do not know whether the optimal Q-function is a linear function of (s, a) or not beforehand, and, indeed, it is not in most cases. If the optimal Q-function is not a linear function and we restrict our representation to a linear function, then the error between our learned Q-function and the actual optimal Q-function can be large, causing our learned policy to significantly deviate from optimal.

Therefore, we need to consider nonlinear function representations. For instance, we may consider a polynomial representation

$$Q(s, a) = w^\top \Phi(s, a)$$

where $w = [w_1, w_2, ..., w_p]^\top \in \mathbb{R}^p$ are coefficients and $\Phi(s, a) = [\phi_1(s, a), \phi_2(s, a), ..., \phi_p(s, a)]^\top$ are monomials of entries of s and a. In this case, the coefficients $w = [w_1, w_2, ..., w_p]^\top$ are the parameters that we need to learn.

Suppose we have samples of the actual optimal Q-function, $\{Q^*(s_k, a_k)\}_{k=1}^K$. We want our representation to have the best fit to these data points, i.e.,

$$\min_w \sum_{k=1}^K \left|Q(s_k, a_k) - Q^*(s_k, a_k)\right| = \sum_{k=1}^K \left|w^\top \Phi(s_k, a_k) - Q^*(s_k, a_k)\right|$$

This is a standard **curve fitting** problem. We know that as we increase the degree of the polynomial, we are able to fit more complex functions.

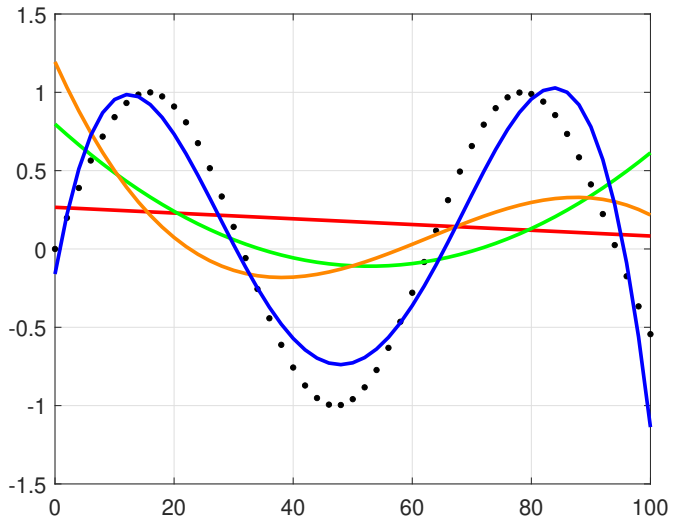

Figure 5: Polynomial curves fitting points generated with a sine function [$y = \sin(0.1x)$]. The black dotted line is the "true" data, the red line is a first degree polynomial, the green line is second degree, the orange line is third degree, and the blue line is fourth degree.

Now we can further relax the restriction that $\Phi(s, a) = (\phi_1(s, a), \phi_2(s, a), ..., \phi_p(s, a))$ must be monomials – we can consider other nonlinear functions as $\phi_i(s, a)$. When we represent $Q(s, a)$ as $Q(s, a) = w^\top \Phi(s, a)$, $\Phi(s, a) = (\phi_1(s, a), \phi_2(s, a), ..., \phi_p(s, a))$ are typically called **features** and $w = (w_1, w_2, ..., w_p)$ called **weights**.

It is clear that the choice of features influences the fitting result. Because the shape of the optimal Q-function is typically not known beforehand, good choices of features are the ones that are able to fit a large range of functions with a small number of features. For instance, **radial basis functions (RBFs)** are shown to be good choices in many problems. A (Gaussian) RBF is defined as

$$\phi_i(s, a) = \exp\left(-(\varepsilon_i \|(s, a) - (s_i, a_i)\|)^2\right)$$

where $(s_i, a_i) \in \mathbb{R}^{m+n}$ is the radial center and $\varepsilon_i > 0$ is a shape parameter. If all features $\Phi(s,a) = (\phi_1(s,a), \phi_2(s,a), ..., \phi_p(s,a))$ are RBFs, a linear combination of them, $w^\top \Phi(s,a)$, is called a **RBF network**, which is a single-layer neural network.

Now we can further relax the restriction that our representation must take the form of $Q(s,a) = w^\top \Phi(s,a)$ – we can consider more general parametric functions, i.e.,

$$Q(s,a) = \Phi_\theta(s,a)$$

where $\Phi_\theta : S \times A \to \mathbb{R}$ is a function parameterized by $\theta \in \mathbb{R}^p$. A multi-layer neural network is such a parametric function, where Φ is determined by the number of layers, the number of neurons, and the choice of activation functions, and θ are the weights of the network. When $Q(s,a) = \Phi_\theta(s,a)$ is a neural network, the fitting process

$$\min_\theta \sum_{k=1}^{K} \big|Q(s_k,a_k) - Q^*(s_k,a_k)\big| = \sum_{k=1}^{K} \big|\Phi_\theta(s_k,a_k) - Q^*(s_k,a_k)\big|$$

is called **training**, and efficient tools have been developed for this process. We will not discuss the details of these tools. They are packaged and ready for us to use.

Now if we can sample the optimal Q-function, the problem reduces to a curve fitting problem. However, we are not able to sample it but need to gradually learn it through Q-learning [this makes reinforcement learning different from supervised learning], during which we need to represent and update our running estimates of the Q-function. This requires a combination of the Q-value update step and the curve fitting step. The algorithm after this combination is called **Neural-Fitted Q-Learning (NFQ)** when a single-layer neural network is used to represent the Q-function and called **Deep Q-Learning** when a multi-layer/deep neural network is used.

Algorithm 6 Neural-Fitted/Deep Q-Learning

Initialize Q-function representation $\tilde{Q}_\theta(s,a)$, create empty buffer $\mathcal{D}$;
for $n = 0, 1, ..., n_{\max} - 1$ (episode) **do**
  for $m = 0, 1, ..., m_{\max} - 1$ (trajectory) **do**
    Randomly initialize state $s_0 \in S$;
    for $t = 0, 1, ..., t_{\max} - 1$ (time) **do**
      Select an action a_t according to

$$a_t \in \begin{cases} \operatorname{argmax}_{a \in A} \tilde{Q}_\theta(s_t,a) & \text{with probability } 1-\varepsilon \\ \text{random action in } A & \text{with probability } \varepsilon \end{cases}$$

      Perform a_t, observe reward $r_t = r(s_t,a_t)$ and next state s_{t+1};
      Calculate updated Q-value of state-action pair (s_t,a_t) according to

$$\hat{Q}(s_t,a_t) = (1-\alpha)\tilde{Q}_\theta(s_t,a_t) + \alpha\big[r_t + \lambda \max_{a' \in A} \tilde{Q}_\theta(s_{t+1},a')\big]$$

      Store $(s_t, a_t, \hat{Q}(s_t,a_t))$ in buffer $\mathcal{D}$;
    end for
  end for
  Update Q-function representation $\tilde{Q}_\theta(s,a)$ according to data in buffer $\mathcal{D}$ (training);
  Empty the buffer $\mathcal{D}$;
end for

Module V: Additional Topics

15 Partially observable Markov decision processes

Recall that an MDP is defined by a 5-tuple,

$$\{T, S, A, p(s'|s,a), r_t(s,a)\}$$

For infinite-horizon MDP, T is (always) equal to $\mathbb{Z}_+$ and r_t is typically time-independent, i.e., $r_t = r$ for all $t \in T$. When considering the expected total discounted reward criterion, the problem also depends on the discount factor $\lambda \in [0, 1)$ as a parameter.

Therefore, an infinite-horizon MDP with an expected total discounted reward criterion can be defined by the following 5-tuple,

$$\{S, A, p(s'|s,a), r(s,a), \lambda\}$$

An important assumption of MDP, which makes a (deterministic Markovian) decision rule in the form of $d_t : S \to A$ and enables all the previously introduced solution techniques (SDP, VI/PI, and Q-learning), is the ability to observe the state s_t at every time step t. However, this may not be the case in many problems.

A **Partially Observable Markov Decision Process (POMDP)** is a decision process model in which it is assumed that the system dynamics are determined by an MDP but the agent cannot directly/fully observe the state. Instead, it receives partial information of the state through a sensor model. A POMDP is defined by the following 7-tuple,

$$\{S, A, p(s'|s,a), r(s,a), O, q(o|s), \lambda\}$$

where $O = \{o_1, o_2, ..., o_l\}$ is a set of **observations** (possible values of the sensor model output), called the **observation space**, and $q(o|s)$ is the **sensor model/observation function** and describes the probability of observing each $o \in O$ conditioned on the current $s \in S$. Typically, the goal of POMDP is to maximize the infinite-horizon expected total discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \lambda^t r(s_t, a_t)\right]$.

At each time step $t \in T$, because the agent cannot observe the state s_t, it can only make decisions using all information it has up to this time, which includes all observations it has received up to this time, $\{o_0, o_1, ..., o_t\}$ and all previous actions it has applied, $\{a_0, a_1, ..., a_{t-1}\}$. Therefore, the information the agent has at time t can be expressed by the following vector

$$h_t = \{o_0, a_0, o_1, a_1, ..., o_{t-1}, a_{t-1}, o_t\}$$

Then, a deterministic decision rule is a mapping from h_t to an action a_t, i.e.,

$$d_t : h_t \mapsto a_t$$

However, a decision rule in this form is not easy to express due to the increasing length of the information vector h_t as t increases.

Suppose the transition probabilities $p(s'|s,a)$ and observation probabilities $q(o|s)$ are known. Then, it is possible for the agent to use the information it has, h_t, to estimate the state s_t. The estimate will be probabilistic – the agent can at most calculate a probability distribution over the state space S that describes the probability of s_t taking each value $s \in S$ given the information vector h_t [think about Kalman filtering]. These probabilities can be written as

$$b_t(s) = \mathbb{P}(s_t = s \,|\, h_t) = \mathbb{P}(s_t = s \,|\, o_0, a_0, o_1, a_1, ..., o_{t-1}, a_{t-1}, o_t), \quad s \in S$$

and is called the agent's **belief** in the state.

Suppose the agent has a previous belief b_{t-1}, applies an action a_{t-1} at the previous step, and receives a new observation o_t at the current step. Then, the agent can update its belief according to the following formula

$$
\begin{aligned}
&b_t(s') = \mathbb{P}(s_t = s' \,|\, h_t) = \mathbb{P}(s_t = s' \,|\, h_{t-1}, a_{t-1}, o_t) \\
&= \frac{\mathbb{P}(s_t = s', o_t \,|\, h_{t-1}, a_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} = \frac{\mathbb{P}(o_t \,|\, s_t = s', h_{t-1}, a_{t-1})\mathbb{P}(s_t = s' \,|\, h_{t-1}, a_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} \qquad \text{Bayes' rule} \\
&= \frac{q(o_t|s')\mathbb{P}(s_t = s' \,|\, h_{t-1}, a_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} = \frac{q(o_t|s')\sum_{s\in S}\mathbb{P}(s_t = s', s_{t-1} = s \,|\, h_{t-1}, a_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} \\
&= \frac{q(o_t|s')\sum_{s\in S}\mathbb{P}(s_t = s' \,|\, s_{t-1} = s, a_{t-1}, h_{t-1})\mathbb{P}(s_{t-1} = s \,|\, h_{t-1}, a_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} \\
&= \frac{q(o_t|s')\sum_{s\in S}p(s'|s, a_{t-1})\mathbb{P}(s_{t-1} = s \,|\, h_{t-1})}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})} \qquad \text{Markov + Causality } (s_{t-1} \text{ does not depend on later action } a_{t-1}) \\
&= \frac{q(o_t|s')\sum_{s\in S}p(s'|s, a_{t-1})b_{t-1}(s)}{\mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1})}
\end{aligned}
$$

or

$$b_t(s') = \frac{1}{\eta_t}\, q(o_t|s') \sum_{s\in S} p(s'|s, a_{t-1})b_{t-1}(s)$$

where $\eta_t = \mathbb{P}(o_t \,|\, h_{t-1}, a_{t-1}) = \sum_{s''\in S} q(o_t|s'') \left(\sum_{s\in S} p(s''|s, a_{t-1})b_{t-1}(s)\right)$ is a normalization factor.

In practice, we will not compute the normalization factor using the above expression. Instead, after computing $\tilde{b}_t(s') = \eta_t b_t(s') = q(o_t|s')\sum_{s\in S} p(s'|s, a_{t-1})b_{t-1}(s)$ for every $s' \in S$, we compute η_t according to $\eta_t = \sum_{s'\in S} \tilde{b}_t(s')$ and then obtain $b_t(s') = \tilde{b}_t(s')/\eta_t$.

Because the above update formula is essentially an application of Bayes' rule, it is called a **recursive Bayesian estimator** or **Bayesian filter**.

The belief vector

$$b_t = \begin{bmatrix} b_t(s_1) \\ b_t(s_2) \\ \vdots \\ b_t(s_l) \end{bmatrix}$$

is called the **belief state** or **information state** of the POMDP. It is a **sufficient statistic** for the POMDP.

A statistic is sufficient with respect to a statistical model and its associated unknown parameter if "no other statistic that can be calculated from the same sample provides any additional information as to the value of the parameter."

The information state b_t is a sufficient statistic for the POMDP because, as it turns out, all information contained in the information vector h_t useful for making optimal decisions is contained in b_t. Consequently, there exists a decision rule in the form of

$$d : b_t \mapsto a_t$$

i.e., it is deterministic, Markovian, and stationary with respect to the information state, that is optimal. Therefore, the problem reduces to computing an optimal decision rule $d : B \to A$, where B represents the space (the set of all possible values) of the information state and is $B = \Delta^{l-1} = \{b \in \mathbb{R}^l : 0 \le b_i \le 1 \text{ for } i = 1, ..., l \text{ and } \sum_{i=1}^{l} b_i = 1\}$, called the "$(l-1)$-dim probability simplex."

It is possible to solve a POMDP "exactly" based on the belief/information state. Specifically, it is possible to convert a POMDP to an MDP whose state s_t is the belief state b_t of the original POMDP, called "belief MDP" (https://en.wikipedia.org/wiki/Partially_observable_Markov_decision_process). After this conversion, MDP solution techniques (e.g., value/policy iteration) can be used to solve the belief MDP and thus the original POMDP. However, this is computationally very demanding.

To see this, let's consider a model with only 10 states, $S = \{s_1, ..., s_{10}\}$. This is a very small-sized problem. To this model, the belief space B is the 9-dimensional continuous space $\Delta^9 = \{b \in \mathbb{R}^{10} : 0 \leq b_i \leq 1 \text{ for } i = 1, ..., 10 \text{ and } \sum_{i=1}^{10} b_i = 1\}$. Note that this is a 9-dimensional space (not 10) because the last entry b_{10} can be treated as a function of $b_1, ..., b_9$ according to the equality condition, and thus, there are only 9 independent variables. In order to run VI/PI on B, we need to discretize B. Let's discretize each dimension of B (which is the interval $[0, 1]$) with a step size $\Delta b_i = 0.1$. Then, we obtain 11 grid points for each dimension. After the discretization, we end up with a space of $11^9 = 2,357,947,691$ belief states, and we need to run VI/PI on this number of belief states. We can see that the complexity increases drastically from MDP to POMDP. This is also why Texas hold'em and Mah-jongg are considered to be harder games than Chess and Go (at least for computers). POMDP is an active research area (e.g., approximate solutions, RL for POMDP) and is making slow progresses.

An interesting fact is that an optimal POMDP decision rule/policy automatically balances **exploration and exploitation** (in an optimal way). The optimal policy will take actions for the purpose of information gathering (to improve the agent's estimate of the state) so that it can make better decisions in the future. Such behavior does not rely on any term of the reward function specially designed to motivate information gathering.

16 Safe reinforcement learning

When a simulation model is not available, RL may be performed directly on hardware (i.e., run hardware experiments to obtain the data for RL). In certain cases, it is desirable to run RL on hardware in an online manner, e.g., to adapt control policy to a priori uncertain or changing operating environment.

However, the operation of a physical system typically needs to satisfy certain safety requirements (e.g., variable bounds, collision avoidance, etc.) and conventional RL algorithms do not have the ability to satisfy such requirements (e.g., due to random exploration). Safe RL is a technique for running RL while satisfying prescribed safety requirements. Safe RL is an active research area. We introduce one safe RL approach below:

Suppose we are dealing with a system represented by the following discrete-time model

$$s_{t+1} = f(s_t, a_t, w_t)$$

where s_t denotes the system state at time t, a_t denotes the control action at t, and w_t denotes uncertainties, which can include system parameter uncertainty, unmeasured external disturbances, and unmodeled dynamics. We assume we know w_t takes values necessarily in a known set W for all t, but we do not know which value it takes – it may take values randomly according to some (unknown) probability distribution over W, $w_t \sim \mathcal{P}_w$, where the probability distribution may even be state-and-action dependent, $\mathcal{P}_w = \mathcal{P}_w(s_t, a_t)$. Note that the case where w_t is some unknown constant (such as an unknown system parameter) or is some unknown deterministic function of (s_t, a_t) can be viewed as w_t following some (unknown) singular distribution, and thus is covered.

Many safety requirements can be represented as

$$s_t \in S_{\text{safe}}, \quad \forall t$$

i.e., the system state must stay inside some safe set, $S_{\text{safe}} \subset S$, for all time.

Suppose the current state is s_t. To ensure that the next state s_{t+1} is necessarily inside S_{safe} whichever value of W the uncertainty w_t takes, we must take an action a_t that satisfies

$$f(s_t, a_t, w) \in S_{\text{safe}}, \quad \forall w \in W$$

or

$$f(s_t, a_t, W) \subset S_{\text{safe}}$$

Depending on the value of s_t, it is possible that all actions in the action space A satisfy the above requirement, some satisfy it and some do not, or none of them satisfy it. If no actions in A satisfy this requirement, then we are not able to ensure safety at $t+1$.

We now consider a set of states, $S_{\text{feasible}} \subset S$, such that for any $s \in S_{\text{feasible}}$, there exists at least one $a \in A$ satisfying $f(s, a, W) \subset S_{\text{safe}}$. Then, if s_t is in S_{feasible}, we are able to find an action $a_t \in A$ that ensures $s_{t+1} \in S_{\text{safe}}$. Meanwhile, we definitely hope that, at $t+1$, we are still able to find an action $a_{t+1} \in A$ that ensures $s_{t+2} \in S_{\text{safe}}$, i.e., satisfies $f(s_{t+1}, a_{t+1}, W) \subset S_{\text{safe}}$. One way to make this necessarily happen is to enforce s_{t+1} to be, not only in S_{safe}, but also in S_{feasible}, i.e.,

$$f(s_t, a_t, W) \subset S_{\text{safe}} \cap S_{\text{feasible}}$$

However, $s_t \in S_{\text{feasible}}$ only guarantees the existence of $a_t \in A$ such that $f(s_t, a_t, W) \subset S_{\text{safe}}$ and there may not be an action satisfying $f(s_t, a_t, W) \subset S_{\text{safe}} \cap S_{\text{feasible}}$, because $S_{\text{safe}} \cap S_{\text{feasible}}$ is a smaller set than S_{safe}.

It turns out that in order to ensure the existence of $a_t \in A$ that guarantees $s_{t+1} \in S_{\text{safe}}$ at every t, we need a set, $S_\infty \subset S$, that satisfies the following two properties:

- $S_\infty \subset S_{\text{safe}}$: Therefore, if s_{t+1} is in S_∞, it is in S_{safe}.
- For any $s \in S_\infty$, there exists $a \in A$ such that $f(s, a, W) \subset S_\infty$: Therefore, if $s_t \in S_\infty$ and we take an action a_t satisfying $f(s_t, a_t, W) \subset S_\infty$, s_{t+1} is necessarily in S_∞ and there will exist $a_{t+1} \in A$ satisfying $f(s_{t+1}, a_{t+1}, W) \subset S_\infty$. That is, the requirement $f(s_t, a_t, W) \subset S_\infty$ is **recursively feasible**.

The second property above is called **(robust) positively invariant**.

Suppose we can compute a set S_∞ that satisfies these two properties. Then, an RL process is safe if we modify the action selection step to

$$a_t \in \begin{cases} \text{argmax}_{a \in A} Q_t(s_t, a) \ \text{ s.t. } \ f(s_t, a, W) \subset S_\infty & \text{with probability } 1-\varepsilon \\ \text{random action in } A \ \text{ s.t. } \ f(s_t, a, W) \subset S_\infty & \text{with probability } \varepsilon \end{cases}$$

i.e., we only take actions that satisfy $f(s_t, a, W) \subset S_\infty$ for exploration and exploitation.

Methods for computing S_∞ typically depend on the function f (linear, piecewise-affine, nonlinear, etc.). In what follows we introduce a method that is particularly suitable when spaces S, A, and W are all finite.

Let $S_0 = S_{\text{safe}}$ and define S_1 according to

$$S_1 = S_0 \cap \{s \in S : \exists a \in A, f(s, a, W) \subset S_0\}$$

This set is similar to $S_{\text{safe}} \cap S_{\text{feasible}}$ considered above – any s in S_1 is safe (since $S_1 \subset S_0$) and admits an action a such that the next state is necessarily in S_0.

Then, we define S_2 according to

$$S_2 = S_0 \cap \{s \in S : \exists a \in A, f(s, a, W) \subset S_1\}$$

Any s in S_2 is safe and admits an action a such that the next state is necessarily in S_1 (therefore, the next state is safe and admits an action a' such that the further next state is necessarily in S_0).

We define S_k, for $k = 1, 2, ...$, recursively according to

$$S_k = S_0 \cap \{s \in S : \exists a \in A, f(s, a, W) \subset S_{k-1}\}$$

and consider S_∞ defined as

$$S_\infty = \bigcap_{k=0}^{\infty} S_k$$

Firstly, it can be easily shown that S_k is a decreasing sequence of sets, i.e., $S_{k+1} \subset S_k$ for all k. Secondly, if there exists k' such that $S_{k'+1} = S_{k'}$, then

$$\begin{aligned} S_{k'+2} &= S_0 \cap \{s \in S : \exists a \in A, f(s,a,W) \subset S_{k'+1}\} \\ &= S_0 \cap \{s \in S : \exists a \in A, f(s,a,W) \subset S_{k'}\} = S_{k'+1} \end{aligned}$$

i.e., $S_k = S_{k'}$ for all $k' \geq k$. In this case,

$$S_\infty = \bigcap_{k=0}^{\infty} S_k = \bigcap_{k=0}^{k'} S_k = S_{k'}$$

and it can be shown that $S_\infty = S_{k'}$ has the two desired properties above. Finally, when S is finite, such a k' necessarily exists.

When S, A, and W are all finite, S_∞ can be computed using the following algorithm:

Algorithm 7 Computation of S_∞

Let $k = 0$ and $S_0 = S_{\text{safe}}$;
while $S_k \neq S_{k-1}$ **do**
    $S_{k+1} = \emptyset$;
    for $s \in S_k$ **do**
        Let $F_s = 0$ (flag for $s \in S_{k+1}$);
        for $a \in A$ **do**
            Let $F_a = 1$ (flag for $f(s,a,W) \subset S_k$);
            for $w \in W$ **do**
                if $f(s,a,w) \notin S_k$ **then**
                    $F_a = 0$ and end the for-loop for w (because this a does not satisfy $f(s,a,W) \subset S_k$);
                end if
            end for
            if $F_a = 1$ **then**
                $F_s = 1$ and end the for-loop for a (because this s admits a such that $f(s,a,W) \subset S_k$);
            end if
        end for
        if $F_s = 1$ **then**
            $S_{k+1} \leftarrow S_{k+1} \cup \{s\}$;
        end if
    end for
    Let $k \leftarrow k + 1$;
end while
Output $S_\infty = S_k$.

17 Inverse reinforcement learning

An MDP/RL problem is to compute an optimal policy with respect to a given reward function. Sometimes we are interested in the inverse problem: Given a policy, we would like to know with respect to what reward function the policy is optimal. This problem is useful in various settings. For instance,

- We want to develop human-like artificial intelligence (AI) such as autonomous driving systems that behave like human drivers. We may view a human as an optimal decision-maker/controller, but we do not know the internal reward function a human is maximizing when making decisions. In this case, we may want to learn the human reward function from human behavior. With the learned reward function, not only will we gain better understanding of human behavior but we will also be able to compute an AI policy that reproduces human behavior (by solving the forward MDP/RL problem).
- When upgrading a control system, we may want to replace a legacy controller (such as a PID) with an optimization-based controller (such as a model predictive controller). We may want the new optimization-based controller to behave like the legacy controller for ranges where the legacy controller is well-tuned. For instance, we may want the new controller to have similar time-domain specifications (e.g., rise time, settling time, etc.) as the legacy controller for small signals, while require the new controller to have the ability to enforce constraints for large signals which the legacy controller may not have. In this case, we may want to know the reward function with respect to which the legacy controller is optimal. We can then use it as the reward function for the new optimization-based controller so that it will have similar behavior as the legacy controller when constraints are not active.

This inverse problem is called **inverse optimal control** or **inverse reinforcement learning (IRL)**. In what follows we introduce IRL in a finite state and action spaces setting.

Recall that in the policy iteration algorithm, we use the following equation to evaluate a policy (indeed, a decision rule $d : S \to A$),

$$(I - \lambda \mathbf{P}_d^\top) V_d = r_d$$

where V_d is the vector of V-values corresponding to the decision rule d, $\mathbf{P}_d$ and r_d are the transition matrix and reward vector corresponding to d. Specifically,

$$V_d = \begin{bmatrix} V_d(s_1) \\ \vdots \\ V_d(s_n) \end{bmatrix}, \quad \mathbf{P}_d = \begin{bmatrix} p(s_1|s_1, d(s_1)) & \cdots & p(s_1|s_n, d(s_n)) \\ \vdots & \ddots & \vdots \\ p(s_n|s_1, d(s_1)) & \cdots & p(s_n|s_n, d(s_n)) \end{bmatrix}, \quad r_d = \begin{bmatrix} r(s_1, d(s_1)) \\ \vdots \\ r(s_n, d(s_n)) \end{bmatrix}$$

The MDP/RL problem can be understood as: Given a reward function $r(s, a)$, find a policy/decision rule d^* such that $V_{d^*} \geq V_d$ for all d, where V_d is the solution to $(I - \lambda \mathbf{P}_d^\top) V_d = r_d$. The IRL problem can be understood as: Given a policy/decision rule d^*, find a reward function $r(s, a)$ such that $V_{d^*} \geq V_d$ for all d, where V_d is the solution to $(I - \lambda \mathbf{P}_d^\top) V_d = r_d$.

We now assume that reward only depends on state, i.e., $r : S \to \mathbb{R}$, and further discuss the IRL problem based on this assumption. When r only depends on s, the policy evaluation equation reduces to

$$(I - \lambda \mathbf{P}_d^\top) V_d = r$$

i.e., the reward vector on the right-hand side no longer depends on the decision rule d. This equation can also be written as

$$V_d = (I - \lambda \mathbf{P}_d^\top)^{-1} r$$

For a given r, suppose d^* is an optimal decision rule and V_{d^*} is its corresponding value vector. Then, according to the optimality equation, we have

$$d^*(s) \in \operatorname*{argmax}_{a \in A} \left[r(s) + \lambda \sum_{s' \in S} p(s'|s, a) V_{d^*}(s') \right] = \operatorname*{argmax}_{a \in A} \left[\sum_{s' \in S} p(s'|s, a) V_{d^*}(s') \right], \quad \forall s \in S$$

which implies

$$\sum_{s' \in S} p(s'|s, d^*(s)) V_{d^*}(s') \geq \sum_{s' \in S} p(s'|s, d(s)) V_{d^*}(s'), \quad \forall s \in S$$

for all decision rules d. In vector form, this can be written as

$$\mathbf{P}_{d^*}^\top V_{d^*} \geq \mathbf{P}_d^\top V_{d^*} \qquad \Longleftrightarrow \qquad (\mathbf{P}_{d^*} - \mathbf{P}_d)^\top V_{d^*} \geq \mathbf{0}$$

for all d, where $\geq$ is entrywise. Using $V_{d^*} = (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r$, the above inequality can be further written as

$$(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}$$

The above analysis leads to the following result:

Theorem (Ng & Russell, 2000): A given policy/decision rule $d^* : S \to A$ is optimal if and only if the reward function $r : S \to \mathbb{R}$ satisfies

$$(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}$$

for all decision rules $d : S \to A$.

According to the above theorem, the IRL problem reduces to a **feasibility problem**: Given a policy/decision rule d^*, find a reward function $r : S \to \mathbb{R}$ (or, a reward vector $r \in \mathbb{R}^n$) such that it satisfies

$$(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}$$

for all decision rules $d : S \to A$.

Given d^*, $\mathbf{P}_{d^*}$ can be computed according to

$$\mathbf{P}_{d^*} = \begin{bmatrix} p(s_1|s_1, d^*(s_1)) & \cdots & p(s_1|s_n, d^*(s_n)) \\ \vdots & \ddots & \vdots \\ p(s_n|s_1, d^*(s_1)) & \cdots & p(s_n|s_n, d^*(s_n)) \end{bmatrix}$$

For each $d : S \to A$, $\mathbf{P}_d$ can be computed similarly. Then, $(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1}$ can be computed. Therefore, $(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}$ are linear inequalities on the unknown variable r.

For each d, $(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}$ gives n linear inequalities on r. When S and A are finite, there are a finite number of (distinctive) decision rules d. Therefore, there are a finite number of linear inequalities for the reward vector r to satisfy. According to the above theorem, any reward vector that satisfies all these linear inequalities is a solution. Consequently, the IRL problem can be solved as a **linear programming** feasibility problem.

However, not all feasible solutions to

$$(\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}, \quad \forall d$$

are equally good (for our IRL purpose). Some solutions are less "meaningful" than others. For instance, it can be easily seen that $r = \mathbf{0}$ is always a solution. But we can learn nothing about the given policy d^* from such a trivial solution. Therefore, we want to introduce some additional criterion to distinguish between more "meaningful" and less "meaningful" solutions.

One proposal is called **margin maximization**, where we want to maximize

$$Q_{d^*}(s, d^*(s)) - \max_{a \in A \setminus d^*(s)} Q_{d^*}(s, a)$$

with $Q_{d^*}(s,a) = r(s) + \lambda \sum_{s' \in S} p(s'|s,a) V_{d^*}(s')$. The idea is to favor solutions of r that make any single-step deviation from $d^*(s)$ as costly as possible. Therefore, if the underlying actual reward function is such a solution r, the agent is more likely to develop a policy equal to d^* (because the second-best policy is significantly worse than d^*). In other words, such a solution r explains the given d^* better than alternative policies by a margin.

With the margin maximization criterion, the IRL problem is formulated as

$$\begin{aligned} \max_{r} \quad & \sum_{s \in S} \Big(Q_{d^*}(s, d^*(s)) - \max_{a \in A \setminus d^*(s)} Q_{d^*}(s,a) \Big) \\ \text{subject to} \quad & (\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}, \quad \forall d : S \to A \\ & |r(s)| \leq r_{\max}, \quad \forall s \in S \end{aligned}$$

where constraints in the second line are introduced to scale the reward function to a magnitude of $r_{\max}$. It can be easily seen that, without these constraints, αr is a feasible solution if r is a feasible solution and a larger α leads to a larger objective function value, causing the problem to be unbounded.

Using $Q_{d^*}(s,a) = r(s) + \lambda \sum_{s' \in S} p(s'|s,a) V_{d^*}(s')$ and $V_{d^*} = (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r$, the objective function can be expressed as

$$\begin{aligned} & \sum_{s \in S} \Big(Q_{d^*}(s, d^*(s)) - \max_{a \in A \setminus d^*(s)} Q_{d^*}(s,a) \Big) \\ =& \lambda \sum_{s \in S} \left(\sum_{s' \in S} p(s'|s, d^*(s)) V_{d^*}(s') - \max_{a \in A \setminus d^*(s)} \sum_{s' \in S} p(s'|s,a) V_{d^*}(s') \right) \\ =& \lambda \sum_{j=1}^{n} \Big(\mathbf{P}_{d^*}(j)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r - \max_{a \in A \setminus d^*(s)} \mathbf{P}_a(j)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \Big) \\ =& \lambda \sum_{j=1}^{n} \Big(\min_{a \in A \setminus d^*(s_j)} (\mathbf{P}_{d^*}(j) - \mathbf{P}_a(j))^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \Big) \end{aligned}$$

where $\mathbf{P}_{d^*}(j)$ denotes the jth column of $\mathbf{P}_{d^*}$, and $\mathbf{P}_a$ is the transition matrix corresponding to $d(s) = a$ for all $s \in S$. Therefore, the IRL problem can be re-expressed as

$$\begin{aligned} \max_{r} \quad & \sum_{j=1}^{n} \Big(\min_{a \in A \setminus d^*(s_j)} (\mathbf{P}_{d^*}(j) - \mathbf{P}_a(j))^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \Big) \\ \text{subject to} \quad & (\mathbf{P}_{d^*} - \mathbf{P}_d)^\top (I - \lambda \mathbf{P}_{d^*}^\top)^{-1} r \geq \mathbf{0}, \quad \forall d : S \to A \\ & |r(s)| \leq r_{\max}, \quad \forall s \in S \end{aligned}$$

which can be solved as a **linear program**.

The IRL method introduced above is based on the assumption of finite state and action spaces, and explicit knowledge of transition model $p(s'|s,a)$ and policy $d^* : S \to A$. Extensions of the above method that handle continuous spaces and learn from trajectory data (called **demonstrations**) instead of relying on explicit knowledge of d^* are developed (see *Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In ICML (Vol. 1, p. 2).*) In general, IRL is an active research area.