蒙特卡洛方法

定义

蒙特卡洛（Monte Carlo, MC）方法是用于**估计价值函数**和**发现最优策略**的第一类学习方法。不同于**DP**，MC 方法**不假设对环境有完全了解**，它仅依赖**经验**，从与环境的实际或模拟交互中获得的状态、动作和奖励的样本序列。其中的经验可以来自实际经验和模拟经验。通过**实际经验**学习则无需环境动态的先验知识，仍可实现最优行为；通过**模拟经验**学习则只需一个能生成样本转移的模型，无需像DP那样提供所有可能转移的完整概率分布。但在许多情况下，**生成样本经验容易**，**显式表达概率分布困难。**

> [!NOTE]
>
> 复习
>
> **状态价值函数**
>
> 对于一个给定的策略 π，**状态 s 的价值**定义为：
> 从状态 s 开始，按照策略 π 采取动作，未来累积回报的**期望值**。记作 $v_\pi(s)$，数学表达式为：
>
> $$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$
>
> 其中：
>
> - $v_\pi(s)$：策略 π 下状态 s 的价值；
> - $\mathbb{E}_\pi[\cdot]$：在策略 π 下的期望（即所有可能的动作选择、状态转移和奖励序列的加权平均）；
> - G_t：从时间步 t 开始的**回报**（return），定义为未来折扣奖励之和：
>
> $$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$
>
> - $\gamma \in [0, 1]$：折扣因子，用于控制未来奖励的重要性；
> - $S_t = s$：当前时刻 t 所处的状态为 s。

蒙特卡洛预测的目标是估计在给定策略 π 下的状态价值函数 $v_\pi(s)$。主要通过对访问该状态后观察到的**回报取平均**实现。随着样本增多，平均值收敛到期望值。

经验可以被划分成多个**幕**，**一个“幕”是指从某个初始状态开始，到一个终止状态结束的完整状态-动作-奖励序列。价值估计**以及**策略改进**在整个幕结束时才进行，蒙特卡洛算法是**逐幕做出改进**的,而非在每 一步 (在线) 都有改进。

形式上，一个幕表示为：

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$$

其中：

- S_0：起始状态；
- S_T 是**终止状态**（terminal state）；
- T 是该回合的总步数（随机变量，可能每回合不同）；
- 所有后续状态和奖励都定义在 $t = 0$ 到 $T - 1$ 之间。

多臂老虎机通过采样并平均每个动作的收益来估计最优动作，适用于**单状态环境**，其中每个动作的**收益独立且固定。蒙特卡洛算法**扩展到多状态场景，通过采样完整幕并平均每个“状态-动作”二元组的回报来估计状态-动作值函数。

主要区别在于，蒙特卡洛**处理多个状态**，每个状态可视为一个独立的上下文相关赌博机问题（即关联搜索），但这些状态并非孤立：**在某一状态采取动作后的回报受同一幕中后续状态动作选择的影响，因为状态转换和策略共同决定未来奖励。**此外，由于策略在学习过程中随时间不断更新，从较早状态的视角观察，后续状态的动作分布会变化，导致状态-动作回报的分布不稳定，因此问题呈现**非平稳性。**这种非平稳性源于策略演进对早期状态估计的间

接影响，增加了学习难度。

> [!NOTE]
>
> **非平稳性**（non-stationarity）在强化学习和决策问题中指的是环境动态特性（如状态转移概率、奖励函数或最优策略）随时间变化，导致学习目标不稳定。具体来说，当问题的统计特性（例如，某个状态-动作对的期望回报）不是固定的，而是依赖于时间或学习过程本身时，该问题被称为非平稳的。关键点在于：
>
> - 如果环境动态固定不变（例如，马尔可夫决策过程的转移概率和奖励函数恒定），问题为平稳（stationary）。
> - 如果环境动态随时间变化（例如，奖励分布漂移或策略更新导致后续状态行为改变），问题为非平稳。此时，学习算法需适应变化（如使用衰减步长或遗忘机制），否则估计值会过时。

为了处理蒙特卡洛方法中的非平稳性，采**用广义策略迭代**（GPI）的思想进行迭代优化。

> [!TIP]
>
> GPI的核心在于交替进行策略评估（预测）和策略改进：首先对固定策略 π 估计其状态值函数 $v_\pi(s)$ 和动作值函数 $q_\pi(s,a)$（预测问题），然后基于当前价值函数改进策略以提升性能，最后通过反复迭代解决控制问题（寻找最优策略 π^*）。

与动态规划（DP）中的GPI不同，DP依赖已知的马尔可夫决策过程（MDP）模型（如状态转移概率和奖励函数）精确计算价值函数，而蒙特卡洛方法仅通过采样完整幕（episodes）的经验回报来学习价值函数，无需环境模型。

具体而言，在预测阶段，蒙特卡洛通过平均从状态 s 或状态-动作对 (s,a) 开始的所有后续回报样本估计 $v_\pi(s)$ 或 $q_\pi(s,a)$；在策略改进阶段，基于更新后的 $q_\pi(s,a)$ 采用贪心策略（如 $\pi'(s) = \arg\max_a q_\pi(s,a)$）生成更优策略；控制问题则通过GPI循环使策略和价值函数逐步收敛至最优。尽管DP和蒙特卡洛均遵循GPI框架以保证最优性，但关键区别在于数据来源：DP使用模型驱动的全状态空间迭代，而蒙特卡洛仅依赖采样经验，这使其适用于未知模型的环境，但也引入了由非平稳性导致的估计偏差，需通过足够多次的幕采样或加权平均技术（如衰减步长）来缓解。

蒙特卡洛预测

同一个状态 s **可能在一个回合中出现多次**；每一次出现都称为一次**访问**；回合中第一次访问 s 称为对 s 的**首次访问**。蒙特卡洛预测有两种形式，首次访问蒙特卡洛方法将 $v_\pi(s)$ 估计为**首次访问** s **后回报的平均值**；**每次访问蒙特卡洛方法**则对所有访问 s 后的回报取平均。

首次访问

$V(s)$ 状态 s 的价值估计，逼近 $v_\pi(s)$

输入

- 待评估的策略 π

初始化

- 对所有 $s \in \mathcal{S}$:
  - $V(s) \leftarrow$ 任意实数
  - $Returns(s) \leftarrow$ 空列表

循环（每个回合）： // 每次循环处理一个完整的回合

1. 按照策略 π 生成一个完整回合：

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$$

2. $G \leftarrow 0$
3. 对 $t = T-1, T-2, \ldots, 0$： // 倒序遍历：从回合末尾向前处理每个时间步

- $G \leftarrow \gamma G + R_{t+1}$
- 若 S_t 未在 $S_0, S_1, \ldots, S_{t-1}$ 中出现过（即首次访问）：
  - 将 G 添加到 $Returns(S_t)$
  - $V(S_t) \leftarrow \text{average}(Returns(S_t))$

> [!NOTE]
>
> $Returns(S_t)$ 中的内容随着循环会不断增加，每个新回合若首次访问 s，就添加一个新回报。蒙特卡洛预测的本质就是**用一个不断增长的经验库 $Returns(s)$ 来逼近状态价值的期望**，而大数定律保证了这个平均值终将收敛到真实 $v_\pi(s)$。

> [!CAUTION]
>
> 在强化学习中，从时间步 t 开始的**回报**（return）定义为：
>
> $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$
>
> 这是一个**从 $t+1$ 到 T 的未来奖励的折扣和。**
>
> 这个公式是**向前看的**（look-ahead），但我们是在**回合结束后**才开始计算，此时所有 $R_{t+1}, R_{t+2}, \ldots, R_T$ 都已知。
>
> 回报有一个非常重要的**递归性质**：
>
> $$G_t = R_{t+1} + \gamma G_{t+1}$$
>
> 也就是说：
>
> - 当前时刻的回报 = 当前奖励 + 折扣后的下一时刻回报。
>
> 这说明：**只要我们知道 G_{t+1}，就能算出 G_t。**
>
> 而最后一个非终止步的回报最容易算：
>
> $$G_{T-1} = R_T + \gamma \cdot 0 = R_T \quad (\text{因为 } G_T = 0)$$
>
> 所以我们可以：
>
> 1. 从 G_{T-1} 开始；
> 2. 然后算 $G_{T-2} = R_{T-1} + \gamma G_{T-1}$；
> 3. 再算 G_{T-3}，依此类推；
> 4. 一直倒推到 G_0。
>
> 所以在计算回报时要使用倒序遍历
>
> 倒序遍历是为了**利用回报的递归性质**，在回合结束后，**从终点反向传播真实回报值**，从而为每个状态提供准确的更新依据。这是蒙特卡洛方法能够"基于完整经验"学习的核心机制。

随着对状态 s 的访问次数趋于无穷：首次访问MC 和 每次访问MC 都收敛到 $v_\pi(s)$。

> [!NOTE]
>
> 在**首次访问MC**中，我们只考虑一个状态 s 在**每个回合中的第一次出现**。对于每一次这样的"首次访问"，记录从该时刻开始直到回合结束的**回报** G_t。这个 G_t 是对真实状态价值 $v_\pi(s)$ 的一个**独立同分布估计**
>
> | 概念 | 解释 |
> | --- | --- |
> | **独立**（Independent） | 不同回合中获得的回报 $G^{(1)}, G^{(2)}, \ldots$ 是相互独立的。因为每个回合是独立生成的（遵循相同策略 π），前一局的结果不影响下一局。 |
> | **同分布**（Identically Distributed） | 所有这些回报都来自同一个概率分布——即在策略 π 下，从状态 s 出发所能获得的回报的分布。 |

概念	解释
有限方差	该分布的方差是有限的（不是无穷大），意味着回报不会极端发散，具有统计稳定性。

由大数定律，**平均值收敛到期望值。**

设 $X_1, X_2, \ldots, X_n$ 是独立同分布的随机变量，均值为 $\mu = \mathbb{E}[X_i]$，方差有限。

令样本均值为：

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

则当 $n \to \infty$ 时，$\bar{X}_n$ **依概率收敛**于 μ，即：

$$\bar{X}_n \xrightarrow{P} \mu$$

> 同时**估计无偏，误差标准差以 $1/\sqrt{n}$ 速度下降（n 为样本数）**

根据无偏估计的定义有，一个估计量 $\hat{\theta}$ 是参数 θ 的**无偏估计**，如果：

$$\mathbb{E}[\hat{\theta}] = \theta$$

第一次访问状态 s 得到的回报 G_t 满足：

$$\mathbb{E}[G_t \mid S_t = s] = v_\pi(s)$$

所以单个 G_t 是 $v_\pi(s)$ 的**无偏估计。多个无偏估计的平均仍然是无偏**的：

$$\mathbb{E}\left[\frac{1}{n}\sum_{k=1}^{n} G^{(k)}\right] = \frac{1}{n}\sum_{k=1}^{n}\mathbb{E}[G^{(k)}] = \frac{1}{n}\sum_{k=1}^{n} v_\pi(s) = v_\pi(s)$$

所以 $V(s)$ 是 $v_\pi(s)$ 的**无偏估计量。**

对于每个回报 $G^{(k)}$ 有，均值：$\mu = v_\pi(s)$；方差：$\sigma^2 = \mathrm{Var}(G^{(k)}) < \infty$

样本均值：

$$\bar{G}_n = \frac{1}{n}\sum_{k=1}^{n} G^{(k)}$$

则 $\bar{G}_n$ 的方差为：

$$\mathrm{Var}(\bar{G}_n) = \frac{\sigma^2}{n}$$

所以标准差为：

$$\mathrm{SD}(\bar{G}_n) = \sqrt{\frac{\sigma^2}{n}} = \frac{\sigma}{\sqrt{n}}$$

所以**误差标准差以 $1/\sqrt{n}$ 速度下降**

[!NOTE]

$$\bar{G}_n = \frac{1}{n}\sum_{k=1}^{n} G^{(k)}$$

利用方差的线性性质

> 方差的一个重要性质是：

$$\mathrm{Var}(aX+bY) = a^2\mathrm{Var}(X) + b^2\mathrm{Var}(Y) + 2ab\,\mathrm{Cov}(X,Y)$$

但如果 X 和 Y **独立**（或不相关），则 $\mathrm{Cov}(X,Y)=0$，所以：

$$\mathrm{Var}(aX+bY) = a^2\mathrm{Var}(X) + b^2\mathrm{Var}(Y)$$

推广到 n 个**相互独立**的随机变量：

$$\mathrm{Var}\left(\sum_{k=1}^{n} X_k\right) = \sum_{k=1}^{n}\mathrm{Var}(X_k)$$

$$\mathrm{Var}(\bar{G}_n) = \mathrm{Var}\left(\frac{1}{n}\sum_{k=1}^{n} G^{(k)}\right)$$

把常数 $\frac{1}{n}$ 提出来（注意是平方）：

$$= \left(\frac{1}{n}\right)^2 \cdot \mathrm{Var}\left(\sum_{k=1}^{n} G^{(k)}\right)$$

由于 $G^{(k)}$ 是**独立**的：

$$= \frac{1}{n^2}\sum_{k=1}^{n}\mathrm{Var}(G^{(k)})$$

又因为它们是**同分布**的，每个的方差都是 σ^2：

$$= \frac{1}{n^2}\sum_{k=1}^{n}\sigma^2 = \frac{1}{n^2}\cdot n\cdot\sigma^2 = \frac{\sigma^2}{n}$$

$$\boxed{\mathrm{Var}(\bar{G}_n) = \frac{\sigma^2}{n}}$$

[!CAUTION]

在统计学中，**误差标准差（Standard Error, SE）** 是指**估计量的标准差**，也就是对某个参数（比如状态价值 $v_\pi(s)$）进行估计时，这个估计值本身的不确定性有多大。

在当前的场景中：

- 不知道真实的状态价值 $v_\pi(s)$。
- 可以通过多次"首次访问"状态 s，得到多个回报 $G^{(1)}, G^{(2)}, \ldots, G^{(n)}$。
- 然后用这些回报的平均值 $\bar{G}_n = \frac{1}{n}\sum_{k=1}^{n} G^{(k)}$ 来估计 $v_\pi(s)$。

这个 $\bar{G}_n$ 就是一个**估计量**，它本身是一个随机变量（因为每次生成的回合是随机的），所以它有分布，也有方差。

因为 $\bar{G}_n$ 是对 $v_\pi(s)$ 的估计，两者之间会有偏差（误差）：

$$\text{误差} = \bar{G}_n - v_\pi(s)$$

因为这是一个无偏估计，所以这个误差的期望是 0，但因为方差存在，所以误差存在波动。

因此使用标准差：

$$\mathrm{SE} = \sqrt{\mathrm{Var}(\bar{G}_n)} = \frac{\sigma}{\sqrt{n}}$$

估计值围绕真实值波动的典型幅度。

二十一点

二十一点（Blackjack）是一个经典的回合制决策问题，目标是使手中牌面点数之和**尽可能接近21点但不超过21点。**

牌面计分规则：

- 数字牌：按面值计分（2–10）；
- 花牌（J/Q/K）：计为 10 点；
- A（Ace）：可计为 **1 或 11 点**，取决于是否导致爆牌。

> 若 A 计为 11 不会导致总和超过 21，则称为**可用 A**（usable ace），此时总是按 11 计算更优。

游戏流程

1. **初始发牌**：
   - 玩家和庄家各发两张牌；
   - 庄家一张牌明示（upcard），另一张暗藏。
2. **天成（Natural）判定**：
   - 若玩家初始即有 21 点（A + 10点牌），称为“天成”；
     - 若庄家无天成 → 玩家胜（+1）；
     - 若庄家也有天成 → 平局（0）；
   - 否则游戏继续。
3. **玩家行动阶段**：
   - 动作空间：`要牌`（hit）或 `停牌`（stick）；
   - 若玩家选择 `要牌`，继续发牌；
   - 若总和 > 21 → **爆牌**（bust），立即失败（奖励 -1）；
   - 若选择 `停牌`，轮到庄家行动。
4. **庄家行动策略（固定）**：
   - 总点数 < 17：继续要牌；
   - ≥ 17：停牌；
   - 若庄家爆牌 → 玩家获胜（+1）；
   - 否则比较双方最终点数，决定胜负或平局。

奖励设定

结果	**奖励**
玩家赢	$+1$
玩家输	-1
平局	0

> 所有中间步骤无奖励（reward = 0），仅在游戏结束时给出终止奖励。
> 因此，**回报 G_t 就等于最终的胜负奖励。**

此外，不进行折扣（$\gamma = 1$），因为这是一个短回合任务，且目标是最大化最终结果。

状态空间建模

我们假设使用**无限牌堆**（infinite deck），即每次抽牌独立同分布，无需记忆历史出牌。

每个状态由以下三个变量决定：

1. **玩家当前点数**：12–21（低于12时必会要牌，无需决策）；
2. **庄家明示牌**：A, 2, 3, ..., 10（共10种）；
3. **是否有可用 A**：是 / 否（boolean）

> 总状态数：$10 \times 10 \times 2 = 200$ 个有效决策状态。 注意：点数小于12的状态不会出现在策略决策中（因为必须继续要牌），因此不计入策略评估的状态集合。

待评估策略（Policy π）

> “当玩家点数为 20 或 21 时停牌，否则继续要牌。”

形式化表示为：

$$\pi(s) = \begin{cases} \text{stick}, & \text{if player sum is 20 or 21} \\ \text{hit}, & \text{otherwise} \end{cases}$$

这是一个简单的启发式策略，易于模拟，适合用于蒙特卡洛预测实验。

蒙特卡洛预测过程

目标：估计该策略下的状态价值函数 $v_\pi(s)$。

方法：首次访问蒙特卡洛（First-Visit MC）

1. **初始化**：
   - 对每个状态 s：
     - $V(s) \leftarrow 0$
     - $Returns(s) \leftarrow [\,]$（空列表）
2. **重复以下步骤若干回合**（例如 50 万局）：
   a. **生成一个完整游戏回合**：
   - 遵循策略 π 进行玩家决策；
   - 庄家按固定规则行动；
   - 记录完整序列：

$$S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$$

   - $R_T \in \{-1, 0, +1\}$ 是最终奖励（也是回报）。

   b. **计算每个时间步的回报 G_t**：
   - 因为 $\gamma = 1$，且中间奖励为 0，所以：

$$G_t = R_T \quad (\text{所有 } t < T)$$

   - 实际上，整个回合的所有非终止状态共享同一个回报值！

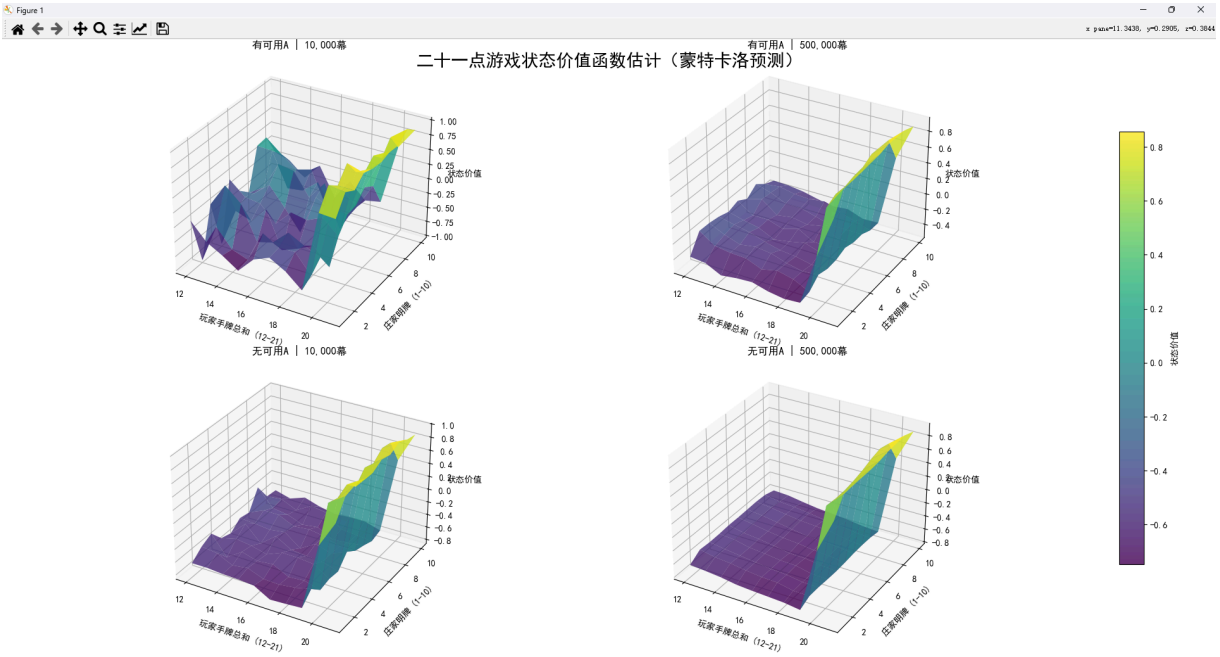
   c. **倒序遍历，更新首次访问的状态**：
   - 初始化 $G = R_T$
   - 对 $t = T-1, T-2, \ldots, 0$：
     - 如果 S_t 是该回合中第一次出现：
       - 将 G 添加到 $Returns(S_t)$
       - 更新 $V(S_t) = \text{average}(Returns(S_t))$

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
from mpl_toolkits.mplot3d import Axes3D
import time
from matplotlib import font_manager

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    # 用来正常显示负号

# 定义牌堆 (13张牌: A=1, 2-9, 10/J/Q/K=10)
DECK = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]

def draw_card():
    """从牌堆随机抽一张牌（有放回）"""
    return np.random.choice(DECK)

def get_hand_value(hand):
    """计算手牌总和及是否有可用的A"""
    total = sum(hand)
    has_usable_ace = 1 in hand  # 检查是否有A
    # 尝试将A当作11（仅当不爆牌时）
    if has_usable_ace and total + 10 <= 21:
        total += 10
        return total, True
    return total, False

def init_hand():
    """初始化玩家和庄家手牌（各发两张牌）"""
    player = [draw_card(), draw_card()]
    dealer = [draw_card(), draw_card()]
    return player, dealer

def is_natural_blackjack(hand):
    """检查是否为天和（21点）"""
    total, _ = get_hand_value(hand)
    return total == 21 and len(hand) == 2

def player_policy(player_sum):
    """玩家策略：手牌<20时要牌，否则停牌"""
    return "hit" if player_sum < 20 else "stick"

def play_episode():
    """模拟一局二十一点游戏，返回(状态序列, 回报)"""
    # 初始化手牌
    player_hand, dealer_hand = init_hand()

    # 检查天和
    if is_natural_blackjack(player_hand):
        dealer_value, _ = get_hand_value(dealer_hand)
        return [], 0 if is_natural_blackjack(dealer_hand) else 1

    # 记录状态序列（仅12~21的非终止状态）
    states = []
    dealer_show = dealer_hand[0]  # 庄家明牌

    # 玩家行动阶段
    while True:
        player_sum, usable_ace = get_hand_value(player_hand)

        # 仅记录12~21的决策状态
        if 12 <= player_sum <= 21:
```

```
            states.append((player_sum, dealer_show, usable_ace))

        # 执行玩家策略
        action = player_policy(player_sum)
        if action == "hit" and player_sum < 20:
            player_hand.append(draw_card())
            player_sum, _ = get_hand_value(player_hand)
            if player_sum > 21:  # 爆牌
                return states, -1
        else:
            break

    # 庄家行动（固定策略：<17要牌）
    dealer_sum, _ = get_hand_value(dealer_hand)
    while dealer_sum < 17:
        dealer_hand.append(draw_card())
        dealer_sum, _ = get_hand_value(dealer_hand)
        if dealer_sum > 21:  # 庄家爆牌
            return states, 1

    # 比较结果
    if player_sum > dealer_sum:
        return states, 1
    elif player_sum < dealer_sum:
        return states, -1
    else:
        return states, 0

def monte_carlo_prediction(num_episodes=500000, progress_interval=10000):
    """蒙特卡洛预测算法主函数（带进度显示）"""
    # 初始化价值函数和计数器
    V = {}
    N = {}

    start_time = time.time()

    for episode in range(1, num_episodes + 1):
        # 生成一幕
        states, reward = play_episode()

        # 更新每个首次访问的状态
        for s in states:
            if s not in V:
                V[s] = 0.0
                N[s] = 0
            N[s] += 1
            # 增量平均更新
            V[s] += (reward - V[s]) / N[s]

        # 定期打印进度
        if episode % progress_interval == 0 or episode == num_episodes:
            elapsed = time.time() - start_time
            print(f"幕数: {episode:,} | 已用时间: {elapsed:.1f}秒 | 平均幕/秒:
{episode/elapsed:.1f}")

    return V

def plot_value_function(V, usable_ace, num_episodes, ax):
    """可视化状态价值函数（集成到子图中）"""
    # 准备数据网格
```

```
    player_sum = np.arange(12, 22)
    dealer_show = np.arange(1, 11)
    X, Y = np.meshgrid(player_sum, dealer_show)
    Z = np.zeros_like(X, dtype=float)

    # 填充价值数据
    for i, ps in enumerate(player_sum):
        for j, ds in enumerate(dealer_show):
            key = (ps, ds, usable_ace)
            Z[j, i] = V.get(key, 0)  # 未访问状态默认0

    # 绘制3D曲面
    surf = ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none', alpha=0.8)
    ax.set_xlabel('玩家手牌总和 (12-21)')
    ax.set_ylabel('庄家明牌 (1-10)')
    ax.set_zlabel('状态价值')

    title = f'{"有" if usable_ace else "无"}可用A | {num_episodes:,}幕'
    ax.set_title(title)

    return surf

# 分别运行不同幕数的实验
print("开始10,000幕实验...")
V_10k = monte_carlo_prediction(num_episodes=10000)

print("\n开始500,000幕实验...")
V_500k = monte_carlo_prediction(num_episodes=500000)

# 创建2x2的子图布局
fig = plt.figure(figsize=(20, 16))

# 有可用A - 10,000幕
ax1 = fig.add_subplot(2, 2, 1, projection='3d')
surf1 = plot_value_function(V_10k, usable_ace=True, num_episodes=10000, ax=ax1)

# 有可用A - 500,000幕
ax2 = fig.add_subplot(2, 2, 2, projection='3d')
surf2 = plot_value_function(V_500k, usable_ace=True, num_episodes=500000, ax=ax2)

# 无可用A - 10,000幕
ax3 = fig.add_subplot(2, 2, 3, projection='3d')
surf3 = plot_value_function(V_10k, usable_ace=False, num_episodes=10000, ax=ax3)

# 无可用A - 500,000幕
ax4 = fig.add_subplot(2, 2, 4, projection='3d')
surf4 = plot_value_function(V_500k, usable_ace=False, num_episodes=500000, ax=ax4)

# 添加颜色条
cbar_ax = fig.add_axes([0.92, 0.15, 0.02, 0.7])
fig.colorbar(surf4, cax=cbar_ax, label='状态价值')

plt.tight_layout(rect=[0, 0, 0.9, 1])  # 为颜色条留出空间
plt.suptitle('二十一点游戏状态价值函数估计（蒙特卡洛预测）', fontsize=20)
plt.savefig('blackjack_monte_carlo.png', dpi=300, bbox_inches='tight')
plt.show()
```



在当前例子中，已经知道二十一点的所有规则：发牌机制、庄家策略、胜负判定、奖励设置等。从理论上讲，这是一个“**完全已知环境**”，似乎非常适合使用**动态规划**（Dynamic Programming, DP）来求解状态价值函数 $v_\pi(s)$。然而，**理论上的可解性 ≠ 实际上的易解性。**

DP 方法（如策略迭代或价值迭代）要求我们明确写出环境的**四参数函数**（环境动态）：

$$p(s', r \mid s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

也就是说，对于每一个状态-动作对 (s, a)，我们必须计算出：

- 所有可能的下一个状态 s'；
- 对应的即时奖励 r；
- 以及它们发生的**精确概率。**

假设当前状态为：

- 玩家点数 = 14
- 有可用 A？否
- 庄家明示牌 = 10
- 动作为：`stick`（停牌）

现在的问题是： **在这个状态下选择 `stick` 后，玩家最终获胜（获得奖励 +1）的概率是多少？**

要回答这个问题，DP 方法需要我们**穷举庄家后续所有可能的抽牌路径**，直到其停止或爆牌：

- 庄家当前总和未知（只知明牌为10，暗牌可能是 A–10）；
- 若暗牌是 A → 当前为 20 → 停牌；
- 若是 2 → 总和 12 → 要牌 → 再抽一张……
- 必须递归模拟庄家策略（≥17 停），计算其最终点数分布；
- 然后比较玩家（14）与庄家最终点数，判断胜负。

这涉及：条件概率（暗牌分布）、多步抽牌的组合爆炸、不同终止点数的概率加权；每种结果对应的奖励等等。虽然数学上可行，但这类计算**极其繁琐、容易出错、难以编码实现。**

蒙特卡洛只需做一件事：**模拟真实游戏过程。**

1. 在该状态下（玩家=14, 庄家明牌=10），执行 `stick`；
2. 让庄家按规则自动行动（<17 要牌，≥17 停）；

3. 记录最终结果：赢（+1）、输（−1）或平局（0）；
4. 重复此过程 1000 次；
5. 统计胜率 → 即为该状态价值的估计。

不需要任何概率公式，不需要枚举所有路径。

维度	动态规划（DP）	蒙特卡洛（MC）
是否需要模型	需要完整环境动态 $p(s', r \mid s, a)$	不需要，仅需样本交互
计算方式	解析式期望： $v(s) = \sum_{s',r} p(\cdots)\,[r + \gamma v(s')]$	经验平均：$V(s) = \frac{1}{n} \sum G^{(k)}$
更新方向	向前（forward in time）：依赖未来状态的价值	向后（backward in time）：从终点反向传播回报
数据来源	数学建模 + 概率推导	实际采样 + 回合经验
实现难度	高（尤其在复杂环境中）	低（易于编程模拟）
学习时机	可以每步更新（同步迭代）	必须等到回合结束
适用任务类型	回合制、持续任务均可	仅适用于回合制任务

蒙特卡洛方法是一种**非自举**（non-bootstrapping）的学习方式，其核心特点是：在更新某个状态的价值时，**完全依赖该状态之后实际观测到的完整回报**，而不借助其他任何状态的价值估计。换句话说，MC 方法通过实际采样的轨迹从终点反向回传真实回报，直接作为目标来更新状态价值，整个过程不涉及对后续状态价值的“引用”或“预测”。这与动态规划（DP）形成鲜明对比——DP 方法基于贝尔曼方程进行**自举**（bootstrapping），即当前状态的价值更新依赖于其后继状态的当前估计值（如 $v(s) \leftarrow \mathbb{E}[r + \gamma v(s')]$）。因此，MC 更加“经验主义”，仅靠真实经验序列学习，无需模型也无需依赖现有估值，但代价是必须等到回合结束才能更新；而 DP 虽然可以提前预测、快速传播价值信息，但高度依赖准确的环境模型和初始估计。

练习

题目：考虑图 5.1 右侧的图。为什么估计的价值函数在最后两行突然增高? 为什么它在最靠左侧的一列降低? 为什么上方图中靠前的值比下方图中对应位置要高?

答案：

1. **最后两行突然增高**：
   - 最后两行对应玩家手牌总和为 20 和 21
   - 根据策略（仅当点数和 ≥ 20 时停牌），玩家在此状态下必然停止要牌
   - 20 和 21 是游戏中胜率最高的点数：
     - 21 点：除非庄家也是 21 点（天和），否则玩家直接获胜
     - 20 点：庄家需达到 17 才能停牌，但很难超过 20 而不爆牌
   - 尤其是当庄家明牌为 2-6 时，庄家爆牌概率高达 35-42%，使玩家胜率显著提升
   - 因此，这两个状态的价值（约 +0.5 到 +0.9）明显高于 19 点及以下状态
2. **最靠左侧一列降低**（庄家明牌为 A）：
   - 最左侧一列对应庄家明牌为 A（数值 1）
   - 庄家明牌为 A 时具有极高优势：
     - 若第二张牌为 10/J/Q/K（概率 4/13），庄家直接获得天和（21 点）
     - 即使不是天和，A 也可灵活计为 1 或 11，使庄家更容易达到 17-21 点

- 统计数据显示：
  - 当庄家明牌为 A 时，玩家胜率约为 40%（vs 明牌为 5 时的 60%）
  - 无可用 A 时，玩家在庄家明牌为 A 的状态下价值普遍为负
- 这种"庄家优势"导致最左侧一列的价值显著低于其他列

3. **上方图比下方图对应位置值高**：
   - 上方图为有可用 A 的状态，下方图为无可用 A 的状态
   - 有可用 A 时的关键优势：
     - 玩家手牌具有双重可能性（例如 A+6 可表示 7 或 17）
     - 可安全要牌逼近 21 点而不易爆牌（如 16 点有 A 时实际为 6 或 16）

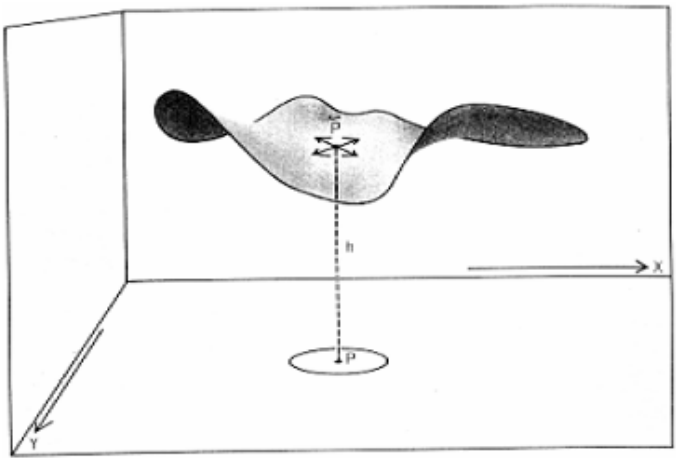
题目：假如我们在解决二十一点问题时使用的不是首次访问型蒙特卡洛算法而是每次访问型蒙特卡洛算法，那么你觉得结果会非常不一样吗？为什么？

答案：

结果**不会显著不同**，原因如下：

1. **状态不可重复性**：
   - 在二十一点游戏中，每个状态在单一幕中**不可能重复访问**
   - 玩家每次要牌后手牌总和单调递增（或爆牌结束）
   - 庄家明牌在游戏开始时即固定不变
   - 可用 A 的状态特性：一旦要牌导致 A 不再"可用"（如总和超过 21 后 A 被计为 1），则后续状态不再具备该特性
2. **算法本质等价**：
   - 首次访问型 MC：仅对幕中首次出现的状态更新价值
   - 每次访问型 MC：对幕中每次出现的状态均更新价值
   - 由于状态不可重复，两种算法在二十一点中：
     - 对每个状态的更新次数完全相同（均为 1 次/幕）
     - 使用的回报值完全相同（幕的最终回报）
     - 价值更新公式完全一致：$V(s) \leftarrow V(s) + \frac{1}{N(s)}[G - V(s)]$
3. **数学证明**：
   - 设状态 s 在幕 i 中出现，回报为 G_i
   - 首次访问型：$V(s) = \frac{1}{n}\sum_{i=1}^{n} G_i$
   - 每次访问型：$V(s) = \frac{1}{n}\sum_{i=1}^{n} G_i$
   - 二者数学表达式完全相同

肥皂泡



A bubble on a wire loop.

将一个已知几何形状的金属丝框架浸入肥皂水中，形成一张**肥皂膜**（或称"肥皂泡表面"）。这张表面具有以下两个关键特性：

1. **内部平衡性**（局部平均性质）：
   - 表面上任意一点所受的合力为零；
   - 数学上等价于：**该点的高度等于其周围邻近点高度的平均值**；
   - 这被称为**调和函数**（harmonic function）的性质，满足拉普拉斯方程 $\nabla^2 h = 0$。
2. **边界约束**：
   - 肥皂膜在边界处必须与金属丝框架完全贴合；
   - 即：**边界点的高度是固定的**，由金属丝形状决定。

目标：计算肥皂膜上任意一点的**高度** $h(x, y)$。

方法一：迭代法（类比 DP）

1. **问题建模**
   将肥皂膜覆盖的区域离散化为一个二维网格，每个网格点 (i, j) 对应一个高度 $h_{i,j}$。边界点位于金属丝框架上，其高度由框架形状确定，记为已知常数。内部点的高度未知，需要求解。
2. **物理规律转化为数学条件**
   肥皂膜在静止状态下受力平衡，满足拉普拉斯方程 $\nabla^2 h = 0$。在离散网格中，该方程等价于：

   > 每个内部点的高度等于其上下左右四个相邻点高度的算术平均值。
   > 即：

   $$h_{i,j} = \frac{1}{4}(h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1})$$

   这是对所有内部点必须满足的条件。
3. **构造方程组**
   每个内部点对应一个如上形式的线性方程，所有内部点构成一个大型线性方程组。该方程组的解即为所求的表面高度分布。
4. **直接求解困难**
   当网格较密时，方程数量庞大，直接求解（如矩阵求逆）计算代价高。因此采用迭代法求近似解。
5. **迭代更新规则**
   初始化所有内部点高度为任意值（如 0）。然后重复以下步骤：
   - 遍历每一个内部点；
   - 将该点的高度更新为其当前四个邻点高度的平均值；

- 所有点更新一轮后，进入下一轮迭代。

6. **信息从边界向内传播**
   初始时内部点高度为猜测值，不满足物理规律。但由于边界点高度固定，在第一次更新时，紧邻边界的内部点会将其值调整为包含边界高度的平均值。这一变化在后续迭代中逐步影响更远的内部点，使得边界条件所携带的信息逐渐向网格中心传播。
7. **逐步逼近平衡状态**
   每次更新都在减小当前点与其邻居之间的不一致性。随着迭代进行，各点高度不断调整，局部偏差逐渐减小。
8. **收敛性保证**
   该迭代过程对应于求解线性方程组的**雅可比迭代法**。由于系数矩阵具有对角占优性和正定性，数学上可证明：无论初始值如何，迭代序列都将收敛到唯一满足边界条件和离散拉普拉斯方程的解。
9. **最终结果**
   经过多轮迭代后，所有内部点的高度变化趋于停止，即满足：

$$h_{i,j} \approx \frac{1}{4}(h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1})$$

> **虽然初始值是错的，但边界值是对的、固定的；每一次更新都在把"边界的信息"往内部传递一步；经过足够多轮，边界的信息就传遍了整个区域，所有点都"知道"了自己该是多少。**该算法可以从任意点开始进行。

方法二：随机游走法（类比 MC）

解法思路

设想我们只关心某个特定起点 P 的表面高度。

可以采用一种完全不同的策略：

1. 从点 P 出发，进行一次**随机游走**（random walk）：
   - 每一步以相等概率移动到上下左右四个相邻网格点；
   - 继续前进，直到到达边界。
2. 记录**首次触达边界时的高度** h_{boundary}。
3. 重复此过程 n 次，得到 n 个边界高度：$h^{(1)}, h^{(2)}, \ldots, h^{(n)}$
4. 取平均：

$$\hat{h}(P) = \frac{1}{n}\sum_{k=1}^{n} h^{(k)}$$

随机游走法之所以能算出点 P 的高度，是因为：

> **系统本身的平衡规则（局部平均）决定了：从 P 出发的所有可能路径，其终点高度的加权平均（按路径概率加权），正好就是 P 的真实高度。**
> 而随机游走正是在模拟这些路径，用频率逼近概率，用样本平均逼近期望

动作值的蒙特卡洛估计

> 在无模型环境下，估计**动作值**（即状态-动作对的价值）比估计状态值更具实用性。

- **有模型时**：仅凭状态值函数 $v_\pi(s)$ 即可确定策略。
  只需向前看一步，结合环境动态 $p(s', r \mid s, a)$，选择能带来最高期望回报的动作：

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

  这正是动态规划（DP）中的策略改进方法。

- **无模型时**：无法获取 $p(s', r \mid s, a)$，因此无法通过 $v_\pi(s)$ 推导出最优动作。
  此时，必须**显式估计每个动作的值**，以便比较不同动作的优劣。

> [!NOTE]
>
> 场景设定：**“悬崖边的宝藏”游戏**
>
> 想象一个智能体（如机器人）站在悬崖边（状态 S），面前有两个选择：
>
> - **动作** A：向左走（安全路径）
> - **动作** B：向右走（危险路径）
>
> 环境规则（**智能体初始不知道这些规则，但作为我们知晓**）：
>
> | 动作 | 结果（概率分布） | 即时奖励 |
> | --- | --- | --- |
> | A | 80% 概率到达宝藏（安全） | +5 |
> | | 20% 概率滑下小坡（轻微受伤） | -2 |
> | B | 60% 概率快速到达宝藏（高效） | +3 |
> | | 40% 概率坠入悬崖（重伤） | -10 |
>
> - **关键假设**：
>   - 所有路径都直接到达**终止状态**（宝藏或悬崖），无后续状态（即 s' 是终止状态，值恒为 0）。
>   - 折扣因子 $\gamma = 1$（简化计算）。
> - **当前策略** π：
>   智能体随机选择动作（50% 概率选 A，50% 概率选 B）。
>
> 有模型环境：仅用状态值 $v_\pi(s)$ 即可改进策略
>
> 假设智能体**知道环境动态** $p(s', r \mid s, a)$（即上表中的概率和奖励）。
>
> 步骤 1：计算当前策略的状态值 $v_\pi(S)$
>
> - $v_\pi(S)$ 是遵循策略 π 时，从状态 S 开始的期望回报：
>
> $$\begin{aligned} v_\pi(S) &= \pi(A|S) \cdot q_\pi(S, A) + \pi(B|S) \cdot q_\pi(S, B) \\ &= 0.5 \cdot \underbrace{[0.8 \times 5 + 0.2 \times (-2)]}_{q_\pi(S,A)} + 0.5 \cdot \underbrace{[0.6 \times 3 + 0.4 \times (-10)]}_{q_\pi(S,B)} \\ &= 0.5 \cdot (4 - 0.4) + 0.5 \cdot (1.8 - 4) \\ &= 0.5 \times 3.6 + 0.5 \times (-2.2) = 1.8 - 1.1 = 0.7 \end{aligned}$$
>
> **结论**：当前策略的期望回报为 $v_\pi(S) = 0.7$。
>
> 步骤 2：用 $v_\pi(S)$ 和环境动态改进策略
>
> - 虽然我们不知道 $q_\pi(S, A)$ 和 $q_\pi(S, B)$，但**利用环境动态** $p(s', r \mid s, a)$，我们可以计算每个动作的期望回报：
>
> 动作 A： $\sum_{r,s'} p(s', r \mid S, A) \left[r + \gamma v_\pi(s') \right] = 0.8 \times (5 + 0) + 0.2 \times (-2 + 0) = 4 - 0.4 = 3.6$
>
> 动作 B： $\sum_{r,s'} p(s', r \mid S, B) \left[r + \gamma v_\pi(s') \right] = 0.6 \times (3 + 0) + 0.4 \times (-10 + 0) = 1.8 - 4 = -2.2$
>
> > 这里只需 $v_\pi(s')$（终止状态值为 0）和 $p(s', r \mid s, a)$，**无需预先知道** q_π。
>
> - 比较结果：$3.6 > -2.2$，因此新策略应为 $\pi'(S) = A$（总是向左走）。
>
> **总结（有模型）**：

> 知道环境动态后，**仅用状态值 $v_\pi(s)$ 即可推导出最优动作**。无需存储动作值函数。

无模型环境：必须显式估计动作值 $q_\pi(s,a)$

现在假设智能体**不知道环境动态**（即不知道概率和奖励的分布），只能通过与环境交互收集经验（如试错）。

步骤 1：通过交互估计状态值 $v_\pi(S)$

- 智能体执行当前策略 π（50% 选 A，50% 选 B）100 次：
  - 选择 A 的 50 次中：
    - 40 次成功（奖励 +5）→ 回报 = +5
    - 10 次滑坡（奖励 -2）→ 回报 = -2
    - A 的平均回报 = $\frac{40\times5+10\times(-2)}{50} = \frac{200-20}{50} = 3.6$
  - 选择 B 的 50 次中：
    - 30 次成功（奖励 +3）→ 回报 = +3
    - 20 次坠崖（奖励 -10）→ 回报 = -10
    - B 的平均回报 = $\frac{30\times3+20\times(-10)}{50} = \frac{90-200}{50} = -2.2$
- **状态值 $v_\pi(S)$ 的估计值**：

  $$v_\pi(S) \approx \frac{\text{所有 } 100 \text{ 次的总回报}}{100} = \frac{(40\times5+10\times(-2))+(30\times3+20\times(-10))}{100} = \frac{180-110}{100} = 0.7$$

  结果：智能体得到 $v_\pi(S) \approx 0.7$。

步骤 2：仅靠 $v_\pi(S)$ 无法确定最优动作！

- **问题**：智能体知道 $v_\pi(S) = 0.7$，但**无法区分动作 A 和 B 的优劣。**
  - 可能性 1：动作 A 很好（平均回报 3.6），动作 B 很差（-2.2），当前策略因随机混合导致平均值为 0.7。
  - 可能性 2：动作 A 一般（平均回报 1.0），动作 B 很好（平均回报 0.4），混合后平均值为 0.7。
  - **仅凭 $v_\pi(S) = 0.7$，无法判断哪种可能性成立。**

步骤 3：必须显式估计动作值 $q_\pi(s,a)$

- 智能体**单独记录每个动作的回报**：
  - 对于动作 A：平均回报 $\approx 3.6 \rightarrow q_\pi(S,A) \approx 3.6$
  - 对于动作 B：平均回报 $\approx -2.2 \rightarrow q_\pi(S,B) \approx -2.2$
- **比较动作值**：$q_\pi(S,A) > q_\pi(S,B)$，因此新策略应为 $\pi'(S) = A$。
- **关键实现**：
  这正是**蒙特卡洛动作值估计**的核心——
  - 每次访问状态-动作对 (S,A)，记录后续回报（如 +5 或 -2）。
  - 对所有访问的回报取平均，得到 $q_\pi(S,A)$。
  - 无需环境动态，仅需交互数据。

总结（无模型）：

> 在不知道环境动态时，**状态值 $v_\pi(s)$ 是"模糊的平均值"，无法指导动作选择**；
> **必须显式估计动作值 $q_\pi(s,a)$**，才能比较不同动作的优劣。

因此，蒙特卡洛方法的主要目标之一是估计最优动作值函数 $q_*(s,a)$。
首先需解决的问题是：**在给定策略 π 下，估计动作值函数 $q_\pi(s,a)$。**

动作值函数 $q_\pi(s,a)$ 定义为：
从状态 s 出发，执行动作 a 后，遵循策略 π 所能获得的未来回报的期望值。

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

与状态值估计类似，蒙特卡洛通过**对实际观测到的回报取平均**来估计 $q_\pi(s,a)$。

- 若在某回合中，时刻 t 满足 $S_t = s$ 且 $A_t = a$，则称**状态-动作对** (s,a) 在该回合中被**访问**。
- **每次访问MC**：将所有访问 (s,a) 后的回报取平均。
- **首次访问MC**：仅对每回合中 (s,a) 的**第一次访问**后的回报取平均。

> 随着每个状态-动作对的访问次数趋于无穷，两种方法均收敛到真实 $q_\pi(s,a)$，且收敛速度为 $\mathcal{O}(1/\sqrt{n})$。

但是相关的操作存在探索不足的问题。若策略 π 是**确定性策略**，则在遵循 π 的过程中，每个状态 s 只会选择一个动作。因此，其他动作 (s,a')（$a' \neq \pi(s)$）**永远不会被执行**，也就没有对应的回报可用于估计 $q_\pi(s,a')$。这会导致这些动作的值无法被评估，进而无法判断是否存在更优动作。

为确保所有状态-动作对都能被访问，引入**探索性起点**假设：每个回合从某个**状态-动作对** (s,a) 开始；要求每个 (s,a) 都有**非零概率**被选为起始对。

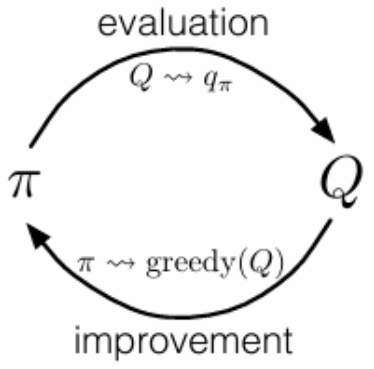
探索性起点在某些理论分析中很有用，但在实际交互学习中通常不可行，因为环境不一定允许从任意状态-动作对开始；初始条件受限，无法自由设定起始动作因此，**探索性起点不能普遍依赖。**

蒙特卡洛控制

定义

蒙特卡洛方法通过完整回合的经验回报平均来估计状态价值函数 $v_\pi(s)$ 和动作值函数 $q_\pi(s,a)$；在无模型环境下，估计动作值函数 $q_\pi(s,a)$ 比状态值函数更重要，因为它能直接指导策略选择；蒙特卡洛控制利用蒙特卡洛方法进行**策略优化**（control），即找到近似最优策略 $\pi \approx \pi_*$。

蒙特卡洛控制遵循**广义策略迭代**（Generalized Policy Iteration, GPI）的思想：**同时维护一个近似策略 π 和一个近似值函数 Q；值函数不断更新以更接近当前策略的值；策略则根据当前值函数不断改进；两者相互促进，共同趋近于最优。**评估和改进过程相互对抗（每种变化都为另一种创造了移动目标），但共同促使系统向最优收敛。



$$\pi_0 \xrightarrow{\mathbb{E}} q_{\pi_0} \xrightarrow{\mathbb{I}} \pi_1 \xrightarrow{\mathbb{E}} q_{\pi_1} \xrightarrow{\mathbb{I}} \pi_2 \xrightarrow{\mathbb{E}} \cdots \xrightarrow{\mathbb{I}} \pi_* \xrightarrow{\mathbb{E}} q_*$$

- $\xrightarrow{\mathbb{E}}$：**策略评估**（Evaluation）— 计算 q_{π_k}
- $\xrightarrow{\mathbb{I}}$：**策略改进**（Improvement）— 从 q_{π_k} 获取 π_{k+1}

> [!IMPORTANT]
>
> 动作值函数 $q_{\pi_k}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ 定义为：
>
> $$q_{\pi_k}(s,a) = \mathbb{E}_{\pi_k}\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_{\pi_k}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

> 该函数量化了在状态 s 执行动作 a 后，遵循策略 π_k 所能获得的期望累积折扣回报。
>
> **计算 q_{π_k} 的根本目的在于实现策略改进**，给定策略 π_k 及其对应的动作值函数 q_{π_k}，若策略 π_{k+1} 满足：
>
> $$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}(s)} q_{\pi_k}(s,a), \quad \forall s \in \mathcal{S}$$
>
> 则对所有状态 $s \in \mathcal{S}$，有：
>
> $$v_{\pi_{k+1}}(s) \geq v_{\pi_k}(s)$$
>
> 且当且仅当 π_k 为最优策略时，等号成立。

策略评估即通过观察多个回合，估计动作值函数 $q_{\pi_k}(s,a)$；在**探索性起点**和**无限多回合**的假设下，蒙特卡洛方法将精确计算 q_{π_k}。

策略改进通过使策略相对于当前动作值函数**贪婪化**完成：

$$\pi_{k+1}(s) \doteq \arg\max_a q_{\pi_k}(s,a) \tag{5.1}$$

根据策略改进定理，对所有 $s \in \mathcal{S}$：

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s) \end{aligned}$$

保证了 π_{k+1} 比 π_k 一致地更好（或相等），整个过程收敛到最优策略和最优值函数。

> [!CAUTION]
>
> **探索性起点**假设：每个回合从某个**状态-动作对** (s,a) 开始；要求每个 (s,a) 都有**非零概率**被选为起始对。

为了能保证收敛，我们做出了两个假设：

假设	问题
探索性起点	实际环境中通常无法从任意状态-动作对开始
无限多回合的策略评估	实际应用中不可能运行无限多回合

但在实际应用中，**不可能运行无限多的回合**。以下提出了两种解决方法：

方案一：**近似评估**

近似估计不要求完全精确评估策略，只需评估到误差在可接受范围内，然后就进行策略改进。

1. 设定一个误差容忍度（如 ε = 0.01）
2. 生成回合，不断更新 $Q(s,a)$ 估计值
3. 当估计值的变化小于 ε 时，停止评估
4. 基于当前 Q 进行策略改进

但为了达到小的误差，可能需要**非常多**的回合。尤其是当状态-动作空间很大时，计算量可能大到不实用。

方案二：**回合交替评估与改进**

回合交替评估与改进，**放弃等待完成完整策略评估**；每个回合后**立即**改进策略；评估和改进**交替进行**，而非顺序进行。

1. 生成**一个完整回合**
2. 用这个回合的数据**更新** $Q(s,a)$（评估）
3. 立即用更新后的 Q **改进策略** π

4. 重复步骤1-3

虽然单个回合提供的信息有限但**长期累积**，$Q(s,a)$ 会越来越接近真实值；策略也会**逐步改进**，最终收敛到最优。

特性	方案一：近似评估	方案二：回合交替评估与改进
执行顺序	评估 → 评估 → ... → 评估 → **改进** (顺序执行)	评估 → **改进** → 评估 → **改进** → ... (交替执行)
评估深度	需要**大量回合**达到精度要求(ε)	每次仅用**单个回合**数据
策略更新频率	仅在**充分评估后**更新一次	**每个回合后立即更新**策略
计算效率	评估阶段计算量大，尤其在大状态空间	更高效，避免"过度评估"浪费
收敛特性	每次改进后策略**严格提升**	单次改进不一定提升，但**长期收敛**
实现复杂度	需监控评估精度(如Δ<ε)	逻辑简单：**生成-评估-改进**循环
实际应用	适合小规模、需高精度问题	**更适合大规模实际问题**

试探性出发算法

蒙特卡洛ES (Monte Carlo ES with Exploring Starts) 是一种基于**每次访问**蒙特卡洛方法的策略迭代算法，用于在无模型环境下求解最优策略 $\pi_* \approx \pi$ 和最优动作值函数 $q_* \approx Q$。

- $\mathcal{S}$: 状态空间
- $\mathcal{A}(s)$: 状态 s 下的可行动作集合
- $\pi : \mathcal{S} \to \mathcal{A}$: 确定性策略
- $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: 动作值函数估计
- G_t: 从时间步 t 开始的回报，$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$
- $\gamma \in [0,1]$: 折扣因子
- $N(s,a)$: 状态-动作对 (s,a) 的访问计数器

$$
\begin{aligned}
&\text{初始化：} \\
&\quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s): \\
&\quad\quad Q(s,a) \leftarrow \text{任意实数} \\
&\quad\quad N(s,a) \leftarrow 0 \\
&\quad \forall s \in \mathcal{S}: \\
&\quad\quad \pi(s) \leftarrow \text{任意 } a \in \mathcal{A}(s) \\
&\\
&\text{循环 (对每个回合)：} \\
&\quad \text{随机选择 } S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0) \text{ 使得 } P(S_0, A_0) > 0 \\
&\quad \text{生成遵循 } \pi \text{ 的完整回合:} \\
&\quad\quad \tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T) \\
&\quad G \leftarrow 0 \\
&\quad \text{对 } t = T-1, T-2, \ldots, 0 \text{ 执行：} \\
&\quad\quad G \leftarrow \gamma \cdot G + R_{t+1} \\
&\quad\quad N(S_t, A_t) \leftarrow N(S_t, A_t) + 1 \\
&\quad\quad Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} [G - Q(S_t, A_t)] \\
&\quad\quad \pi(S_t) \leftarrow \arg\max_{a \in \mathcal{A}(S_t)} Q(S_t, a)
\end{aligned}
$$

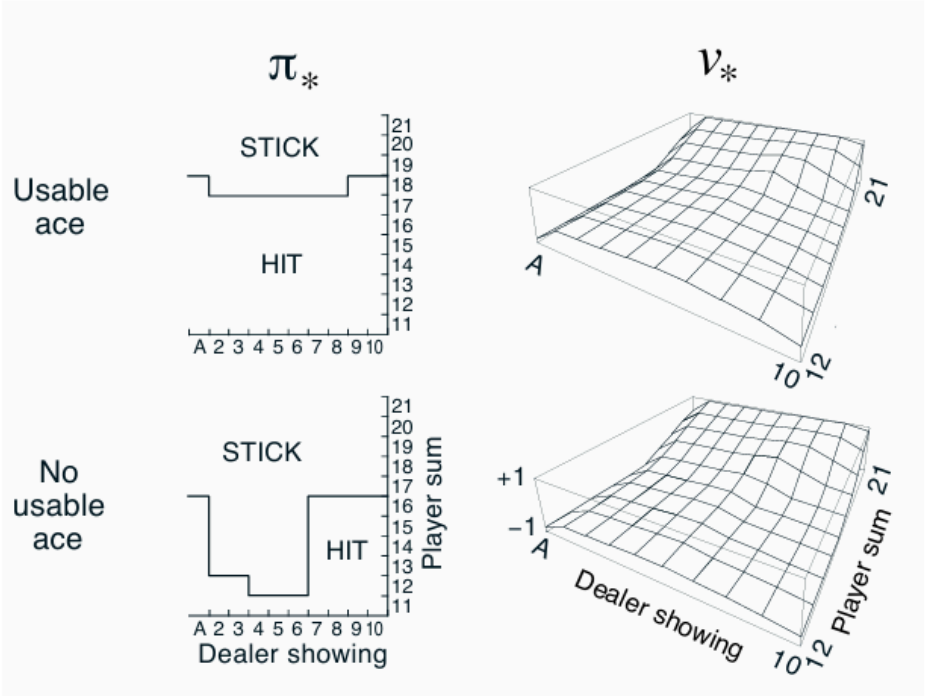
1. **初始化**:
   - 动作值函数 $Q(s,a)$ 可初始化为任意实数值

    - 访问计数器 $N(s,a)$ 初始化为 0
    - 初始策略 $\pi(s)$ 可任意选择（如随机选择或固定策略）
2. **回合生成**：
    - 利用"探索性起点"条件，随机选择起始状态-动作对 (S_0, A_0)
    - 从该起点开始，按照当前策略 π 生成完整回合 τ
3. **回报计算与更新**：
    - 从回合末尾向前遍历（$t = T-1, T-2, \ldots, 0$）
    - 递归计算回报 $G \leftarrow \gamma \cdot G + R_{t+1}$
    - **对每个出现的状态-动作对 (S_t, A_t) 都进行更新**
    - **增量式更新**动作值函数：$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}[G - Q(S_t, A_t)]$
4. **策略改进**：
    - **对每个状态 S_t**，将策略更新为贪婪策略
    - 即选择具有最大动作值的动作：$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

- **每次访问**：对回合中状态-动作对的**每次出现**都进行更新
- **探索性起点**：确保所有状态-动作对都能被访问（通过随机选择起始状态-动作对）
- **收敛保证**：在探索性起点和无限回合假设下，收敛到最优策略
- **非自举**：不依赖其他状态的估计值，仅使用实际观测到的回报
- **高效实现**：采用增量式更新，无需存储所有历史回报，只需维护 $Q(s,a)$ 和 $N(s,a)$

特性	首次访问 (First-visit)	每次访问 (Every-visit)
更新条件	仅当状态-动作对首次出现时更新	每次出现都更新
样本利用率	低（仅使用 1/N 的样本）	高（使用全部样本）
收敛速度	较慢	**更快**（更高效利用数据）
理论保证	收敛到最优策略	收敛到最优策略
ES 设置适用性	不必要（ES 已保证探索）	**完全适用**

使用上述方法重复进行实验有：



找到的最优策略与之前的策略相同，但唯一例外可用A策略中最左边出现了一个凹口和原来不同。

没有探索性起点的蒙特卡洛控制

> [!NOTE]
>
> **探索性起点**（Exploring Starts）是蒙特卡洛方法中的一个**理论假设**，它要求：每个回合可以从**任意状态-动作对** (s, a) 开始，并且每个状态-动作对都有**非零概率**被选为起始点。
>
> **探索性起点假设的局限性**：
>
> - 在实际应用中，通常无法从任意状态-动作对开始
> - 环境往往对初始条件有限制
> - 无法控制环境的初始状态和动作

在线策略与离线策略

同轨策略中**用于生成采样数据序列的策略和用于实际决策的待评估和改进的策略**是**相同**的。**离轨策略**用于评估或者改进的策略与生成采样数据的策略是不同的, 即生成的数据"离开"了待优化的策略所决定的决策序列轨迹。

> [!IMPORTANT]
> 两种方法的根本区别在于：**用于与环境交互生成数据的策略**是否与**被评估或改进的策略**相同。

> [!NOTE]
> 蒙特卡洛ES方法属于**在线策略方法**，但它依赖于探索性起点假设。

ε-贪婪策略

在同轨策略控制方法中，策略通常是**软**的（soft），即对所有状态 $s \in \mathcal{S}$ 和动作 $a \in \mathcal{A}(s)$，满足 $\pi(a|s) > 0$(即**每个可能的动作在每个状态下都有被选择的机会**)，但逐渐向确定性最优策略靠拢。

ε-贪婪策略以概率 $1-\varepsilon$ 选择具有最大估计动作值的动作实现利用，以概率 ε 随机选择一个动作**实现探索**。所有非贪婪动作获得最小选择概率 $\frac{\varepsilon}{|A(s)|}$，剩余概率 $1-\varepsilon+\frac{\varepsilon}{|A(s)|}$ 分配给贪婪动作。

> [!NOTE]

ε-贪婪策略由两部分组成：

1. **贪婪部分**（概率 $1-\varepsilon$）：
   - 以 $1-\varepsilon$ 的概率选择当前估计值最高的动作（贪婪动作）
   - 这部分**只分配给贪婪动作**
2. **随机探索部分**（概率 ε）：
   - 以 ε 的概率均匀随机选择一个动作
   - 这部分**均匀分配给所有 $|A(s)|$ 个可能动作**
   - 每个动作（包括贪婪动作）获得 $\frac{\varepsilon}{|A(s)|}$ 的概率

贪婪动作 a^* 的概率

- 来自贪婪部分：$1-\varepsilon$
- 来自随机探索部分：$\frac{\varepsilon}{|A(s)|}$
- **总计**：$1-\varepsilon+\frac{\varepsilon}{|A(s)|}$

非贪婪动作 $a \neq a^*$ 的概率

- 来自贪婪部分：0（因为不是最优动作）
- 来自随机探索部分：$\frac{\varepsilon}{|A(s)|}$
- **总计**：$\frac{\varepsilon}{|A(s)|}$

$$
\begin{aligned}
\text{总概率} &= P(a^*|s) + \sum_{a \neq a^*} P(a|s) \\
&= \left(1-\varepsilon+\frac{\varepsilon}{|A(s)|}\right) + (|A(s)|-1) \times \frac{\varepsilon}{|A(s)|} \\
&= 1-\varepsilon+\frac{\varepsilon}{|A(s)|} + \varepsilon - \frac{\varepsilon}{|A(s)|} \\
&= 1
\end{aligned}
$$

ε-软策略要求对所有状态和动作满足 $\pi(a|s) \geq \frac{\varepsilon}{|A(s)|}$ 的策略（其中 $\varepsilon > 0$），ε-贪婪策略是ε-软策略的一个特例。在ε-软策略中，ε-贪婪策略在某种意义上是最接近贪婪的策略，保证了对所有动作的持续探索。

[!IMPORTANT]
没有探索性起点的假设，我们不能简单地通过使策略相对于当前值函数完全贪婪化来改进策略，因为这将阻止对非贪婪动作的进一步探索。幸运的是，GPI（广义策略迭代）并不要求策略完全变为贪婪策略，只需要向贪婪策略**靠近**即可。

在线策略首次访问MC控制算法

在线策略首次访问MC控制算法的**目标**是估计最优策略 $\pi \approx \pi_*$，无需探索性起点假设。其**核心思想**是使用ε-软策略保持探索，同时向贪婪策略靠近。通过首次访问MC方法估计动作值函数，策略改进为ε-贪婪策略。

算法参数：一个小的 $\varepsilon > 0$

初始化：

- $\pi \leftarrow$ 任意 ε-软策略
- $Q(s,a) \leftarrow$ 任意实数值，对所有 $s \in \mathcal{S}$，$a \in \mathcal{A}(s)$
- $Returns(s,a) \leftarrow$ 空列表，对所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$

循环（对每个回合）：

1. **生成回合**：
   - 按照当前策略 π 生成完整回合：$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
2. **计算回报**：

- $G \leftarrow 0$
- 对 $t = T-1, T-2, \ldots, 0$：
  - $G \leftarrow \gamma G + R_{t+1}$

3. **更新动作值函数**：
   - 如果 (S_t, A_t) 是该回合中首次访问：
     - 将 G 添加到 $Returns(S_t, A_t)$
     - $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
     - $A^* \leftarrow \text{argmax}_a Q(S_t, a)$（平局任意打破）
4. **策略改进**：
   - 对所有 $a \in \mathcal{A}(S_t)$：
     - $\pi(a|S_t) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$

与蒙特卡洛ES相比，**关键区别**在于策略改进步骤：不是完全贪婪化，而是变为ε-贪婪策略。这种方法确保了对所有动作的持续探索，同时向最优策略靠近。算法在无限回合下会收敛到ε-软策略中的最优策略。

策略改进定理

为证明**使用ε-贪婪策略进行改进确实能保证策略不会变差**，并且最终会收敛到ε-软策略中的最优策略。引入策略改进定理。

> [!IMPORTANT]
>
> 如果对于所有状态 s，有 $q_\pi(s, \pi'(s)) \geq v_\pi(s)$，则策略 π' 至少与策略 π 一样好，即 $v_{\pi'}(s) \geq v_\pi(s)$ 对所有 s 成立。

> [!NOTE]
>
> 1. $v_\pi(s)$：状态值函数
>
> $$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \sum_a \pi(a|s) q_\pi(s, a)$$
>
> - **含义**：在状态 s 下遵循策略 π 的期望回报
>
> 2. $q_\pi(s, a)$：动作值函数
>
> $$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$
>
> - **含义**：在状态 s 下执行动作 a，然后遵循策略 π 的期望回报
>
> 3. $q_\pi(s, \pi'(s))$：新策略在状态 s 的期望值
>
> $$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s, a)$$
>
> - **含义**：在状态 s 下遵循新策略 π' 选择动作，然后继续遵循原策略 π 的期望回报

证明策略改进定理

> [!IMPORTANT]
> **核心思想**：如果在每个状态下，新策略 π' 选择的动作至少与原策略 π 一样好，那么整体上 π' 就不会比 π 差。

考虑从状态 s 开始，比较两条路径：

1. **完全遵循 π**：获得回报 $v_\pi(s)$
2. **仅第一步用 π'，之后用 π**：获得回报 $q_\pi(s, \pi'(s))$

如果对所有 s 都有 $q_\pi(s, \pi'(s)) \geq v_\pi(s)$，这意味着：

- **只做一次改变**（第一步用 π'）就能获得不低于原策略的回报
- **持续使用** π' 只会更好，不会更差

从递归角度看：

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

$$q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

如果 $q_\pi(s, \pi'(s)) \geq v_\pi(s)$，则：

- 在每个状态 s，π' 选择的动作导致的即时回报加上后续状态值不低于 π 的选择
- 由于 $v_\pi(S_{t+1})$ 本身也是基于 π 的，如果 π' 在每一步都至少一样好，那么整体回报必然不低于 π

证明ε-贪婪策略满足策略改进定理

证明：任何相对于 q_π 的ε-贪婪策略 π' 满足 $q_\pi(s, \pi'(s)) \geq v_\pi(s)$，因此 $\pi' \geq \pi$。

> **定理**：任何相对于 q_π 的ε-贪婪策略 π' 优于或等于任何ε-软策略 π。
>
> **证明**：
>
> 对任何状态 $s \in \mathcal{S}$：
>
> $$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) \\ &= v_\pi(s) \end{aligned}$$
>
> 因此，根据策略改进定理，$\pi' \geq \pi$。

步骤 1：展开 $q_\pi(s, \pi'(s))$

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s, a)$$

含义：这是ε-贪婪策略 π' 在状态 s 的期望动作值。

步骤 2：代入ε-贪婪策略定义

$$= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a)$$

解释：

- **随机探索部分**（概率 ε）：
  - 以均匀概率 $\frac{\varepsilon}{|\mathcal{A}(s)|}$ 选择任意动作
  - 贡献：$\frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a)$
- **贪婪利用部分**（概率 $1 - \varepsilon$）：
  - 选择当前估计值最高的动作
  - 贡献：$(1 - \varepsilon) \max_a q_\pi(s, a)$

步骤 3：建立关键不等式

$$\geq \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

[!CAUTION]

要证

$$\frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\max_a q_\pi(s,a) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

两边都有 $\frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a)$，可以从两边消去：

$$(1-\varepsilon)\max_a q_\pi(s,a) \geq (1-\varepsilon)\sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

由于 $1-\varepsilon > 0$（ε 是一个小的正数，通常在 0 到 1 之间），两边同时除以 $1-\varepsilon$：

$$\max_a q_\pi(s,a) \geq \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

考虑权重项：

$$w(a) = \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon}$$

我们需要证明 $w(a)$ 是一个**有效的概率分布**：

由于 π 是 ε-软策略，根据定义：

$$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|} \quad \text{对所有 } a \in \mathcal{A}(s)$$

因此：

$$\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|} \geq 0$$

且 $1-\varepsilon > 0$，所以：

$$w(a) = \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} \geq 0$$

计算权重总和：

$$\begin{aligned}
\sum_a w(a) &= \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} \\
&= \frac{1}{1-\varepsilon}\sum_a \left(\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}\right) \\
&= \frac{1}{1-\varepsilon}\left(\sum_a \pi(a|s) - \sum_a \frac{\varepsilon}{|\mathcal{A}(s)|}\right) \\
&= \frac{1}{1-\varepsilon}(1-\varepsilon) \\
&= 1
\end{aligned}$$

所以$w(a)$ 是一个有效的概率分布（非负且和为1）。

最大值与加权平均的关系

> [!DEFINITION]
> **基本不等式**：对于任何一组数值 $\{x_1, x_2, \ldots, x_n\}$ 和任何有效的概率分布 $\{p_1, p_2, \ldots, p_n\}$（$p_i \geq 0$ 且 $\sum p_i = 1$），有：
>
> $$\max_i x_i \geq \sum_i p_i x_i$$

证明：

设 $x^* = \max_i x_i$，则对所有 i，有 $x_i \leq x^*$。

因此：

$$\sum_i p_i x_i \leq \sum_i p_i x^* = x^* \sum_i p_i = x^* \cdot 1 = x^*$$

即：

$$\max_i x_i \geq \sum_i p_i x_i$$

在我们的不等式中：

- $x_a = q_\pi(s, a)$
- $p_a = w(a) = \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon}$

由于 $w(a)$ 是一个有效的概率分布，根据基本不等式：

$$\max_a q_\pi(s, a) \geq \sum_a w(a) q_\pi(s, a) = \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a)$$

即证

$$w(a) = \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon}$$

这个权重表示：**在排除了强制探索部分后，原策略如何分配其"决策权重"。**

步骤 4：简化表达式

$$\begin{aligned}
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \left(\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|} \right) q_\pi(s, a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) \\
&= \sum_a \pi(a|s) q_\pi(s, a) \\
&= v_\pi(s)
\end{aligned}$$

关键转换：

- 将 $(1 - \varepsilon)$ 与分母抵消
- 展开求和并合并同类项
- 最终得到 $v_\pi(s)$ 的定义

ε-软策略中的最优性

虽然策略改进定理证明了ε-贪婪策略改进能保证策略不会变差，但它**没有告诉我们何时达到最优。**最优性内容提供了**判断是否达到最优的精确标准**，完成了策略迭代的理论闭环。

> [!IMPORTANT]
> **核心思想**：考虑一个新环境 $\widetilde{M}$，它与原始环境 M 相同，但将ε-软约束"内建"到环境中。"将ε-软约束'内建'到环境中"是指**将原本需要在策略层面强制满足的约束条件，转变为环境本身的固有特性。**

修改版环境工作原理：

- 在状态 s 采取动作 a 时：
  - 以概率 $1-\varepsilon$：行为与原环境 M 相同
  - 以概率 ε：随机重新选择动作（均匀分布），然后使用新动作

在修改版环境中，最优值函数 $\tilde{v}_*$ 满足：

$$\tilde{v}_*(s) = (1-\varepsilon)\max_a \tilde{q}_*(s,a) + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a \tilde{q}_*(s,a)$$

解释：

- 以概率 $1-\varepsilon$：选择最优动作 $\max_a \tilde{q}_*(s,a)$
- 以概率 ε：随机选择动作，期望值为 $\frac{1}{|A(s)|}\sum_a \tilde{q}_*(s,a)$

当ε-软策略 π 不再能被改进时（即相对于 q_π 的ε-贪婪策略等于 π 本身）：

$$v_\pi(s) = (1-\varepsilon)\max_a q_\pi(s,a) + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a)$$

这两个方程形式完全相同，由于贝尔曼方程有唯一解，因此 $v_\pi = \tilde{v}_*$。

回忆策略改进定理证明中的关键步骤：

$$v_\pi(s) = \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

当策略 π 不再能被改进时，π 已经是相对于 q_π 的ε-贪婪策略，即：

- 对于贪婪动作 a^*：$\pi(a^*|s) = 1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}$
- 对于非贪婪动作 $a \neq a^*$：$\pi(a|s) = \frac{\varepsilon}{|\mathcal{A}(s)|}$

代入上式：

$$\begin{aligned} v_\pi(s) &= \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\cdot\max_a q_\pi(s,a) \end{aligned}$$

这正是修改版环境中的最优性方程。

基于重要度采样的离轨策略

离策略学习

所有学习控制方法都面临一个两难问题，希望学习基于**后续最优行为**的动作价值，但为了**探索所有动作**（以找到最优动作），需要进行非最优行为。同轨策略方法与离轨策略方法的根本区别在于：**用于与环境交互生成数据的策略**（行为策略）是否与**被评估或改进的策略**（目标策略）相同。 上部分的**没有探索性起点的蒙特卡洛控制方法**就是一种同策略方法，其学习的不是最优策略的动作价值，而是**学习仍保持探索的近似最优策略的动作价值。**

离策略学习提供了一种解决方案：使用**两种策略**协同工作。**目标策略**（target policy）是要学习的策略，目标是使其成为最优策略。**行为策略**（behavior policy）：用于生成行为的策略，通常更具探索性。这种方法允许算法"**基于偏离目标策略的数据进行学习**"，从而解决探索与利用的矛盾。

> [!CAUTION]
> **离策略学习的局限性**
>
> - 离策略方法需要**额外的概念和符号**
> - 由于数据来自不同策略，离策略方法通常具有**更大的方差**
> - **收敛速度更慢**
> - 需要满足**覆盖假设**（coverage assumption）：

在离策略**预测**问题中，目标策略 π 和行为策略 b 都是**固定且已知**的。我们希望估计 v_π 或 q_π，但只有遵循行为策略 b 的片段（其中 $b \neq \pi$）可用。

> [!NOTE]
>
> **知道策略** π，就是我们完全了解策略 π 的定义，即对于每个状态 s，我们知道 $\pi(a|s)$（在状态 s 选择动作 a 的概率）。**执行策略** b就是我们实际与环境交互时使用的是行为策略 b，因此只收集了 b 生成的片段（状态-动作-奖励序列）。在离策略预测中，核心目标就是：**使用行为策略** b **生成的数据（片段）来估计目标策略** π **的价值函数** v_π **或** q_π，即使 $b \neq \pi$。

覆盖假设

为了使用来自行为策略 b 的片段来估计目标策略 π 的值，我们需要确保一个关键条件：**在** π **下采取的每个动作在** b **下也至少偶尔被采取。**

$$\pi(a|s) > 0 \implies b(a|s) > 0$$

覆盖假设确保了重要性采样能够有效工作。如果目标策略 π 在某个状态下可能会选择某个动作，那么行为策略 b 也必须以正概率选择该动作。如果 $\pi(a|s) > 0$ 但 $b(a|s) = 0$，则重要性采样比率中会出现除以零的情况，导致无法进行有效估计。覆盖假设保证了所有与目标策略相关的状态-动作对都有数据支持。

由覆盖假设可知，**在与** π **不完全相同的状态下，** b **必须是随机的。**如果 π 是确定性策略（如总是选择最优动作），那么 b 必须**在所有状态上都是随机**的。行为策略 b 需要探索那些目标策略 π 可能选择的动作。**目标策略** π **可以是确定性的**，例如，最优策略通常是确定性的（贪婪选择最优动作）。

> [!NOTE]
>
> “在与 π 不完全相同的状态下，b 必须是随机的”
>
> 如果策略 π 是确定性的（比如在某个状态 s 只选一个动作 a^*），而行为策略 b 在该状态下不是完全跟着 π 走（即 $b \neq \pi$），那么为了满足**覆盖假设**，b 就不能只选 a^*，它必须有一定的随机性，去尝试其他动作。否则，如果 b 也是确定性地只选 a^*，那就和 π 完全一样了，这就变成了同轨设置。所以，“不完全相同”意味着 b 和 π 不一致；此时为了让 b 仍能产生 π 所关心的动作（尤其是非贪婪动作），就必须引入**随机性**（比如 ε-greedy 中的探索部分）。
>
> 举个例子：
>
> - π: 总是选择最优动作（确定性贪婪策略）
> - b: 使用 ε-greedy 策略，在大部分时间跟随 π，但有 ε 概率随机探索
>
> → 这样 b 是随机的，满足覆盖假设，可以用于学习 π
>
> “如果 π 是确定性策略……那么 b 必须在所有状态上都是随机的”
>
> 进一步推论，若 π 是**完全确定性**的（每个状态只选一个动作），而你想用一个不同的 b 来收集数据训练这个 π（即 off-policy 学习），那么 b 必须在**每个状态**都能以一定概率选择 π 所选择的那个动作。更重要的是，为了让学习过程中能持续获得关于 π 动作的数据，并且允许探索其他可能性，b 通常需要在所有状态都具备一定的**探索能力**（也就是随机性）。这里说“必须是随机的”，并不是说 b 必须均匀随机，而是指它不能是确定性的（否则一旦偏离 π，就可能永远采不到 π 关心的动作）。

> “行为策略 b 需要探索那些目标策略 π 可能选择的动作”
>
> 这正是覆盖假设的直观体现，我们想学的目标策略 π 可能在某些状态下选择特定动作（比如最优动作）。但如果行为策略 b 从来没试过这些动作，我们就得不到这些动作的反馈（奖励、转移结果），也就无法知道它们好不好。因此，b 必须“探索”这些动作，即使它自己并不偏爱它们。
>
> “目标策略 π 可以是确定性的”
>
> 在很多算法中（如 Q-learning），我们的**目标策略 π 是确定性的**，比如：
>
> $$\pi(s) = \arg\max_a Q(s,a)$$
>
> 即贪婪策略。但我们使用的**行为策略 b 是随机的**，比如：
>
> $$b(a|s) = \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}|, & a = \arg\max Q(s,a) \\ \varepsilon/|\mathcal{A}|, & \text{otherwise} \end{cases}$$
>
> 即 ε-greedy。
>
> 这就是典型的 **离轨学习框架**，用一个随机的行为策略 b 收集数据，来学习一个确定性的目标策略 π。

重要性采样

重要性采样是一种**给定一个分布的样本来估计另一个分布下期望值**的通用技术。在离策略学习中，它通过根据目标策略和行为策略下轨迹发生的**相对概率**（重要性采样比率）对回报进行加权。

给定起始状态 S_t，在策略 π 下后续状态-动作轨迹 $A_t, S_{t+1}, A_{t+1}, \dots, S_T$ 发生的概率为：

$$\begin{aligned} &\Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t|S_t)p(S_{t+1}|S_t,A_t)\pi(A_{t+1}|S_{t+1})\cdots p(S_T|S_{T-1},A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k) \end{aligned}$$

其中 p 是状态转移概率函数。

S_t: 时间步 t 的状态

- A_t: 时间步 t 的动作
- $S_{t+1}, A_{t+1}, \dots, S_T$: 从 $t+1$ 到终止时刻 T 的状态和动作序列
- π: 策略，通常是策略函数 $\pi(a|s)$，表示在状态 s 下选择动作 a 的概率
- $A_{t:T-1} \sim \pi$: 表示从时间 t 到 $T-1$ 的所有动作都是根据策略 π 采样得到的

> 注意：最终状态是 S_T，所以最后一个动作是 A_{T-1}

这个表达式描述了在策略 π 下，从状态 S_t 开始的完整轨迹发生的概率。

$$\Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\}$$

这表示：**给定起始状态为 S_t，在策略 π 下，后续动作序列 $A_t, A_{t+1}, \dots, A_{T-1}$ 和状态序列 $S_{t+1}, S_{t+2}, \dots, S_T$ 发生的概率。**

这个概率可以分解为：

$$\pi(A_t|S_t)p(S_{t+1}|S_t,A_t)\pi(A_{t+1}|S_{t+1})\cdots p(S_T|S_{T-1},A_{T-1})$$

目标策略和行为策略下轨迹的**相对概率（重要性采样比率）**为：

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

> $\rho_{t:T-1}$ 从时间t到终止时刻$T-1$的**重要性采样比**

尽管轨迹概率依赖于MDP的转移概率（通常未知），但它们在分子和分母中完全相同，因此**相互抵消**。重要性采样比率最终**只依赖于两个策略和序列，而不依赖于MDP。**

> [!NOTE]
>
> 复习MDP
>
> 在马尔可夫决策过程（MDP）中，**转移概率**（transition probability）是指：
>
> > 在状态 s 下执行动作 a 后，环境转移到状态 s' 的概率，通常记为 $p(s'|s,a)$ 或 $P(s'|s,a)$。
>
> 数学上严格定义为：
>
> $$p(s'|s,a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$$
>
> 这表示：**给定当前状态为 s，执行动作 a 的条件下，下一时刻状态为 s' 的条件概率。**
>
> 对于任何固定的状态-动作对 (s,a)，所有可能的下一状态 s' 的转移概率之和为1：
>
> $$\sum_{s' \in \mathcal{S}} p(s'|s,a) = 1$$
>
> 这保证了转移概率构成一个有效的概率分布。
>
> 转移概率体现了MDP的核心特性——**马尔可夫性**：
>
> - 下一状态 s' 的分布**只依赖于当前状态 s 和动作 a**
> - 与之前的状态和动作历史无关
> - 数学表达：$\Pr\{S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots\} = \Pr\{S_{t+1}|S_t, A_t\}$

行为策略生成的回报 G_t 具有期望 $\mathbb{E}[G_t|S_t = s] = v_b(s)$，但这个回报是具有错误的。通过乘以重要性采样比率 $\rho_{t:T-1}$，将期望值从 $v_b(s)$ 转换为 $v_\pi(s)$，这使得我们可以使用行为策略生成的数据来估计目标策略的价值函数。**重要性采样的核心作用就是将行为策略下的回报转换为目标策略下的期望值。**

$$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

在离策略蒙特卡洛预测中，我们希望估计目标策略 π 的状态价值函数 $v_\pi(s)$，但只有遵循行为策略 b 的片段数据可用。为了有效组织这些数据，我们采用**跨越片段边界的编号方式**标记时间步：

- 如果第一个片段在时间 100 结束于终止状态，则下一个片段从时间 $t = 101$ 开始
- 这种连续编号使我们能够用单一时间索引指代特定片段中的特定步骤

定义关键概念：

- $\mathcal{T}(s)$：访问状态 s 的所有时间步的集合
  - **首次访问方法**：$\mathcal{T}(s)$ 只包含每个片段中首次访问 s 的时间步
  - **每次访问方法**：$\mathcal{T}(s)$ 包含所有访问 s 的时间步
- $T(t)$：时间 t 之后的首次终止时间
- G_t：从时间 t 到 $T(t)$ 的回报
- $\rho_{t:T(t)-1}$：从 t 到 $T(t)-1$ 的重要性采样比率，$\rho_{t:T(t)-1} = \prod_{k=t}^{T(t)-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

普通重要性采样

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

普通重要性采样对**缩放后的回报**进行**算术平均**：

1. 对于每个访问状态 s 的时间步 $t \in \mathcal{T}(s)$：
   - 计算重要性采样比率 $\rho_{t:T(t)-1}$
   - 将回报 G_t 乘以该比率，得到缩放回报 $\rho_{t:T(t)-1} G_t$
2. 将所有缩放回报相加
3. 除以访问次数 $|\mathcal{T}(s)|$，得到最终估计

单次回报情况下，假设在状态 s 只观察到一次回报 G_t，重要性采样比率为 ρ；估计值为 ρG_t。例如，若 $\rho = 10$，则估计值是观察到回报的 10 倍。**虽然是无偏的，但是可能比较极端。**

> [!NOTE]
>
> 要证明普通重要性采样是无偏的，我们需要证明：
>
> $$\mathbb{E}_b[\rho G_t | S_t = s] = v_\pi(s)$$
>
> 其中 $\mathbb{E}_b$ 表示期望是在行为策略 b 下计算的。
>
> 重要性采样比率定义为：
>
> $$\rho = \rho_{t:T(t)-1} = \prod_{k=t}^{T(t)-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$
>
> 计算期望：
>
> $$\begin{aligned} \mathbb{E}_b[\rho G_t | S_t = s] &= \mathbb{E}_b \left[\left(\prod_{k=t}^{T(t)-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \right) G_t \middle| S_t = s \right] \\ &= \sum_\tau p(\tau | b, S_t = s) \cdot \rho(\tau) \cdot G_t(\tau) \end{aligned}$$
>
> 其中 τ 表示从 S_t 开始的完整轨迹，$p(\tau|b, S_t = s)$ 是在行为策略 b 和起始状态 $S_t = s$ 下轨迹 τ 发生的概率。
>
> 根据重要性采样比率的定义：
>
> $$\rho(\tau) = \frac{p(\tau|\pi, S_t = s)}{p(\tau|b, S_t = s)}$$
>
> 其中 $p(\tau|\pi, S_t = s)$ 是在目标策略 π 和起始状态 $S_t = s$ 下轨迹 τ 发生的概率。
>
> 代入期望公式：
>
> $$\begin{aligned} \mathbb{E}_b[\rho G_t | S_t = s] &= \sum_\tau p(\tau | b, S_t = s) \cdot \frac{p(\tau|\pi, S_t = s)}{p(\tau|b, S_t = s)} \cdot G_t(\tau) \\ &= \sum_\tau p(\tau | \pi, S_t = s) \cdot G_t(\tau) \\ &= v_\pi(s) \end{aligned}$$
>
> **关键步骤**：$p(\tau|b, S_t = s)$ 在分子和分母中抵消，剩下的正是目标策略 π 下的期望回报。

这样合理的原因是比率 ρ 表示目标策略下该轨迹发生的可能性是行为策略下的 ρ 倍，因此，该回报对目标策略价值的贡献应放大 ρ 倍，这确保了估计的期望值正确：$\mathbb{E}[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$。

加权重要性采样

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}}$$

（如果分母为零则为零）

加权重要性采样对**缩放后的回报**进行**加权平均**：

1. 对于每个访问状态 s 的时间步 $t \in \mathcal{T}(s)$：
   - 计算重要性采样比率 $\rho_{t:T(t)-1}$
   - 将回报 G_t 乘以该比率，得到缩放回报 $\rho_{t:T(t)-1}G_t$
2. 将所有缩放回报相加作为分子
3. 将所有比率相加作为分母
4. 分子除以分母，得到最终估计

单次回报时，假设在状态 s 只观察到一次回报 G_t，设其重要性采样比率为 ρ，则估计值为 $\frac{\rho G_t}{\rho} = G_t$；**估计等于观察到的回报，与比率无关。**但它的期望是$v_b(s)$而不是$v_\pi(s)$，从这个统计意义上说它是**有偏的。**

> [!IMPORTANT]
>
> 在统计学中，一个估计量 $\hat{\theta}$ 被称为**有偏**的，如果它的期望值不等于被估计的参数 θ：
>
> $$\mathbb{E}[\hat{\theta}] \neq \theta$$
>
> 在离策略预测问题中：
>
> - 我们想要估计的参数是 $v_\pi(s)$（目标策略 π 下状态 s 的真实价值）
> - 我们的估计量是 $V(s)$
>
> 因此，如果 $\mathbb{E}[V(s)] \neq v_\pi(s)$，则 $V(s)$ 是有偏估计。
>
> 在加权重要性采样中，当在状态 s 只观察到一次回报时：
>
> $$V(s) = \frac{\rho G_t}{\rho} = G_t$$
>
> 表面上看，估计值等于观察到的回报 G_t，与比率 ρ 无关。
>
> G_t 是**从行为策略 b 生成的片段**中得到的回报，因此，G_t 的期望是行为策略下的状态价值：
>
> $$\mathbb{E}_b[G_t|S_t = s] = v_b(s)$$
>
> 但我们要估计的是**目标策略 π 下的状态价值** $v_\pi(s)$
>
> 因此：
>
> $$\mathbb{E}[V(s)] = \mathbb{E}[G_t] = v_b(s) \neq v_\pi(s)$$
>
> **它的期望值是 $v_b(s)$，而不是我们想要估计的 $v_\pi(s)$。**所以这是有偏的。

特性	**普通重要性采样 (Ordinary Importance Sampling)**	**加权重要性采样 (Weighted Importance Sampling)**
数学定义	$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\|\mathcal{T}(s)\|}$	$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}}$
平均方式	算术平均（简单平均）	加权平均（比率归一化）

特性	普通重要性采样 (Ordinary Importance Sampling)	加权重要性采样 (Weighted Importance Sampling)
首次访问方法的偏差	无偏：$\mathbb{E}[V(s)] = v_\pi(s)$	有偏：$\mathbb{E}[V(s)] \neq v_\pi(s)$，但偏差渐近收敛到零
单次回报情况	估计值 = ρG_t（可能极端，如$\rho = 10$时为回报的10倍）	估计值 = G_t（等于观察到的回报，与比率无关）
方差特性	通常无界：比率的方差可能是无界的 当轨迹包含循环时尤其明显	通常有界：任何单次回报的最大权重为1 即使比率方差无限，估计器方差也会收敛到零
收敛性	理论上收敛到真实值，但收敛可能不稳定	通常更稳定地收敛到真实值
实践表现	通常误差较高，尤其在学习初期 例5.4中学习曲线显示较高误差	通常具有显著更低的方差 例5.4中学习曲线显示开始时误差低得多
每次访问方法	有偏，但偏差随样本增加渐近降至零	有偏，但偏差随样本增加渐近降至零
实现复杂度	相对简单	稍复杂（需计算分母和）
对异常值敏感度	高：单个高比率片段可能主导估计	低：归一化处理限制了单个片段的影响
理论优势	严格无偏，理论分析更简洁	方差特性更好，更符合实际应用需求
扩展性	更容易扩展到函数逼近方法	在函数逼近中实现更复杂
适用场景	需要严格无偏性的理论分析 某些特定算法中	大多数实际应用 存在循环的MDP 小样本情况

1. **覆盖假设要求**：两种方法都要求满足覆盖假设 $\pi(a|s) > 0 \implies b(a|s) > 0$
2. **单次回报的特殊行为**：
    - 普通重要性采样：单次估计可能极端（如比率为10时，估计值为回报的10倍）
    - 加权重要性采样：单次估计等于观察到的回报，但期望值是 $v_b(s)$ 而非 $v_\pi(s)$
3. **实践建议**：
    - 在大多数实际应用中，**加权重要性采样通常是首选**
    - 例5.4显示：加权方法在学习初期误差显著更低
    - 例5.5显示：普通重要性采样可能在某些MDP上完全失败（方差无限）

练习

考虑一个最简单的马尔可夫决策过程（MDP），其中：

- 只有一个非终止状态，记为 s；
- 只有一个可用动作，因此策略是确定的（无需选择）；
- 执行该动作后：
    - 以概率 p 返回状态 s（继续）；
    - 以概率 $1-p$ 转移到终止状态（结束 episode）；
- 每次状态转移（包括最后一次到终止状态）都获得即时奖励 $r = +1$；
- 折扣因子 $\gamma = 1$；

- 观察到一幕（episode）持续了 10 个时间步，即从 $t=0$ 到 $t=9$ 处于状态 s，在 $t=10$ 进入终止状态。

这意味着这幕序列中状态 s 被访问了 10 次（分别在 $t=0,1,\ldots,9$），并在第 10 步转移到终止状态。总的回报为：

$$G=\sum_{t=0}^{9}\gamma^t\cdot 1=\sum_{t=0}^{9}1=10$$

因为 $\gamma=1$，所以回报就是奖励次数的总和。

> 问题：对这个非终止状态 s 的价值估计，在**首次访问型蒙特卡罗方法**（first-visit MC）和**每次访问型蒙特卡罗方法**（every-visit MC）下分别是多少？

状态 s 的真实价值 $v_\pi(s)$ 是从 s 开始、遵循策略 π 的期望回报。由于只有一个动作，策略固定，我们可以直接计算：

从状态 s 出发，持续获得奖励直到终止。令 V 表示从 s 出发的期望回报，则：

$$V=1+p\cdot V$$

解释：当前步获得奖励 1，然后以概率 p 回到 s，后续期望仍为 V；以概率 $1-p$ 终止，后续回报为 0。

解方程：

$$V=1+pV\Rightarrow V(1-p)=1\Rightarrow V=\frac{1}{1-p}$$

这是状态 s 的真实价值。

但本题不是要求真实价值，而是问：**基于这一幕长度为 10 的经验数据**，使用首次访问和每次访问蒙特卡罗方法，会得到怎样的价值估计？

注意：这里只观察了一幕（one episode），且该幕中状态 s 被访问了 10 次。

首次访问型蒙特卡罗（First-Visit MC）

在这种方法中，对于某一状态 s，我们只考虑它在一幕中**第一次被访问时的回报**，并用这个回报来更新其价值估计。

在这幕中，状态 s 第一次出现在 $t=0$，之后每次都在 s，直到 $t=9$。但由于系统始终处于同一个状态 s，而这是唯一的状态，所以“首次访问”发生在 $t=0$。

关键点是：**从首次访问开始，一直到终止，所获得的总回报是多少？**

从 $t=0$ 开始，共经历 10 步转移，获得 10 次奖励，总回报为：

$$G_{\text{first}}=10$$

因此，在首次访问型 MC 中，我们会将状态 s 的价值估计更新为这个回报值（如果之前没有数据，或这是第一幕，则估计值就是 10）。

所以，**首次访问型的价值估计为 10。**

每次访问型蒙特卡罗（Every-Visit MC）

在这种方法中，状态 s 每次出现都会被视为一个独立的“起点”，我们计算从该次访问开始到 episode 结束的回报，并将所有这些回报用于估计 $v(s)$。

在这幕中，状态 s 出现了 10 次，分别在时间步 $t=0,1,2,\ldots,9$。

我们计算每一次访问对应的回报（即从该时刻起，直到终止的累计奖励）：

- 在 $t=0$ 访问：后续有 10 次奖励 → 回报 = 10

- 在 $t = 1$ 访问：后续有 9 次奖励 → 回报 = 9
- 在 $t = 2$ 访问：后续有 8 次奖励 → 回报 = 8
- ...
- 在 $t = 9$ 访问：后续有 1 次奖励 → 回报 = 1

所以，共有 10 个访问，对应 10 个回报：$10, 9, 8, \ldots, 1$

在每次访问型 MC 中，状态 s 的价值估计是**所有这些回报的平均值。**

计算平均回报：

$$\frac{1}{10}\sum_{k=1}^{10} k = \frac{1}{10}\cdot\frac{10\cdot 11}{2} = \frac{55}{10} = 5.5$$

因此，每次访问型 MC 对状态 s 的价值估计为 5.5。

- 首次访问型蒙特卡罗估计：10
- 每次访问型蒙特卡罗估计：5.5

实例

- **目标状态** s：二十一点游戏中一个特定状态，由三要素确定：
  - 庄家明示牌（upcard）：**2点**
  - 玩家当前点数总和：**13点**
  - 玩家是否有可用A（usable ace）：**有**（即A被计为11点不会爆牌）
- **目标策略** π：
  - 仅在**玩家点数为20或21**时选择"停牌"（stick）
  - 其他情况下选择"要牌"（hit）
  - 这是一个确定性策略：$\pi(\text{stick}|s) = 1$ 当且仅当 玩家点数=20或21
- **行为策略** b：
  - 从**目标状态** s **开始**，以**相等概率**随机选择"要牌"或"停牌"
  - 即：$b(\text{hit}|s) = b(\text{stick}|s) = 0.5$
  - **注意**：行为策略只在目标状态 s 是随机的；一旦离开 s，后续动作由目标策略 π 决定（因为实验只关注估计 s 的价值）

1. **初始化**：每次实验从**目标状态** s 开始
   - 庄家明示牌设为2
   - 玩家点数设为13
   - 玩家有可用A
2. **行为策略执行**：
   - 在状态 s，行为策略 b 以50%概率选择"要牌"，50%概率选择"停牌"
   - 一旦离开状态 s（即执行了动作），后续动作由**目标策略** π 决定
   - 游戏继续直到终止（玩家爆牌、停牌后比较点数等）
3. **片段收集**：
   - 每次从状态 s 开始到游戏结束构成一个片段
   - 每个片段提供状态 s 的一次访问（首次访问方法）
   - 记录从 s 开始的回报 G_t（+1赢，0平，-1输）

- 为了确定状态 s 在目标策略 π 下的真实价值 $v_\pi(s)$：
  - 生成**一亿个**遵循目标策略 π 的完整游戏片段
  - 仅关注从状态 s 开始的片段
  - 计算这些片段回报的平均值
  - 结果：$v_\pi(s) \approx 0.27726$

> [!NOTE]
>
> $v_\pi(s) \approx 0.27726$代表，如果从上述特定局面开始，**严格按照"20或21点才停牌"的策略**玩二十一点，**长期重复多次**后，玩家的**平均每局收益**约为0.27726。

实验结果解释

- **两种方法最终都收敛**：随着片段数量增加，两种方法的估计都接近真实值0.27726
- **加权重要性采样初期表现更好**：
  - 在少量片段时，加权方法的误差显著低于普通方法
  - 因为普通方法将每个"要牌"片段的回报放大2倍，导致早期估计波动更大
- **符合理论预期**：
  - 普通重要性采样无偏但方差大
  - 加权重要性采样有偏但方差小，尤其在样本量少时

这个实验清晰地验证了理论分析：尽管加权重要性采样在单次回报时有偏，但在实际应用中通常提供更稳定、误差更低的估计，特别是在学习初期。

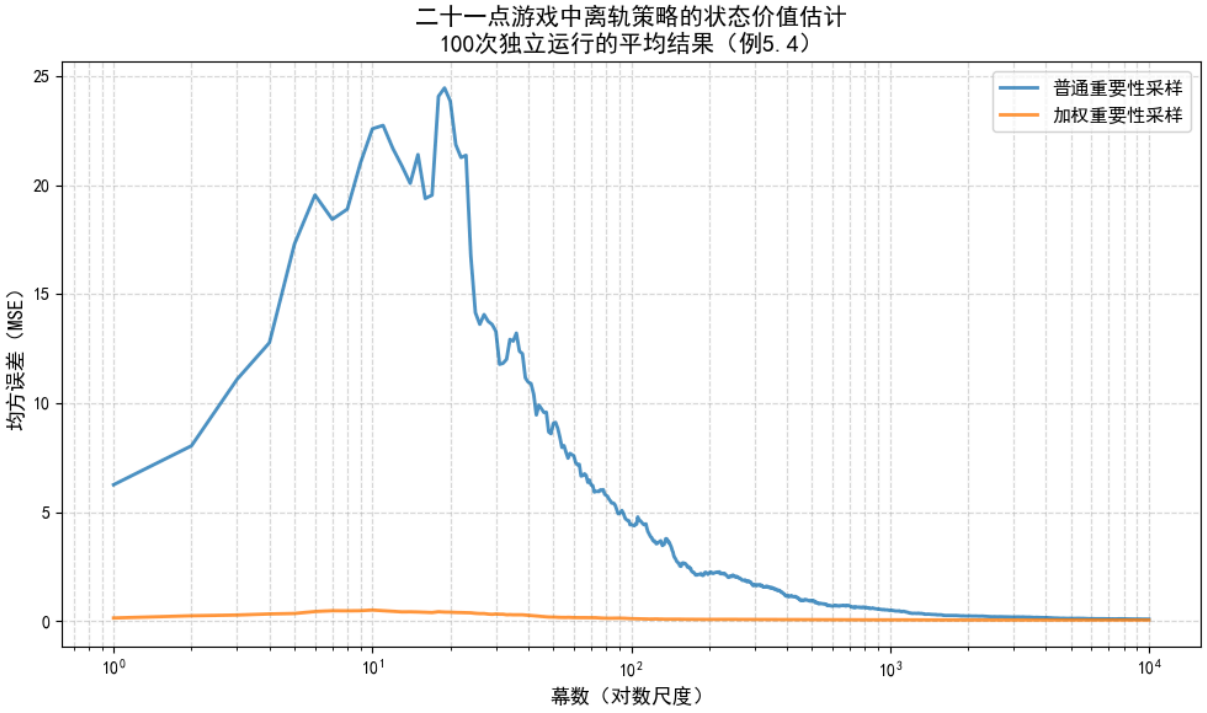
```
import numpy as np
import matplotlib.pyplot as plt
from typing import List
import random

# 设置中文字体（防止中文乱码）
plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS', 'DejaVu Sans']  # 支持中文的字体
plt.rcParams['axes.unicode_minus'] = False  # 正确显示负号

# ================== 工具函数 ==================

def draw_card() -> int:
    """抽一张牌，J/Q/K=10，A=1"""
    card = random.randint(1, 13)
    return min(card, 10)

def has_usable_ace(hand: List[int]) -> bool:
    """是否有可用的A（即A可作11不爆牌）"""
    return 1 in hand and sum(hand) + 10 <= 21

def sum_hand(hand: List[int]) -> int:
    """计算手牌总点数，优先使用可用A"""
    total = sum(hand)
    return total + 10 if has_usable_ace(hand) and total + 10 <= 21 else total

def is_bust(hand: List[int]) -> bool:
    """是否爆牌"""
    return sum_hand(hand) > 21

def score(hand: List[int]) -> int:
    """返回该手牌的得分（用于胜负判断）"""
```

```python
    return 0 if is_bust(hand) else sum_hand(hand)

def dealer_strategy(hand: List[int]) -> bool:
    """庄家策略：小于17必须要牌（软17规则）"""
    return sum_hand(hand) < 17

# ================== 策略定义 ==================

def behavior_policy(player_hand) -> str:
    """行为策略：等概率选择 hit 或 stick"""
    return random.choice(['hit', 'stick'])

def target_policy(player_hand) -> str:
    """目标策略：仅当点数 >= 20 时停牌"""
    return 'stick' if sum_hand(player_hand) >= 20 else 'hit'

# ================== 模拟一局游戏 ==================

def play_episode(start_from_target: bool = False):
    """
    模拟一局二十一点游戏。
    若 start_from_target=True，则强制进入目标状态：
        - 玩家：A + 2 → 13点，有可用A
        - 庄家明牌：2
    返回：(回报 G, 重要性权重 ρ, 是否有效)
    """
    if start_from_target:
        player_hand = [1, 2]          # A 和 2 → 13点，有可用A
        dealer_hand = [2, draw_card()] # 庄家明牌为2
    else:
        player_hand = [draw_card(), draw_card()]
        dealer_hand = [draw_card(), draw_card()]

    weight_ratio = 1.0  # 重要性采样比累积值

    # 玩家回合
    while True:
        action = behavior_policy(player_hand)
        target_action = target_policy(player_hand)

        # 目标策略是确定性的
        pi_prob = 1.0 if action == target_action else 0.0
        if pi_prob == 0:
            weight_ratio = 0.0
            break

        b_prob = 0.5  # 行为策略概率
        weight_ratio *= (pi_prob / b_prob)

        if action == 'hit':
            player_hand.append(draw_card())
            if is_bust(player_hand):
                break
        else:
            break

    # 庄家行动
    while dealer_strategy(dealer_hand):
        dealer_hand.append(draw_card())
```

```
    # 计算回报
    player_score = score(player_hand)
    dealer_score = score(dealer_hand)

    if player_score > dealer_score:
        reward = 1.0
    elif player_score == dealer_score:
        reward = 0.0
    else:
        reward = -1.0

    valid = (weight_ratio > 0)
    return reward, weight_ratio, valid

# ================== 价值估计器 ==================

def estimate_value_over_episodes(num_episodes: int, method: str = 'weighted'):
    """
    使用指定方法估计目标状态的价值。
    :param num_episodes: 幕数
    :param method: 'ordinary' 或 'weighted'
    :return: 每一幕后的价值估计序列
    """
    estimates = []

    if method == 'ordinary':
        total_rhoG = 0.0
        count_valid = 0
    elif method == 'weighted':
        weighted_sum = 0.0  # \sum \rho_i G_i
        weight_sum = 0.0    # \sum \rho_i
    else:
        raise ValueError("method must be 'ordinary' or 'weighted'")

    for i in range(num_episodes):
        G, rho, valid = play_episode(start_from_target=True)

        if not valid:
            current_V = estimates[-1] if estimates else 0.0
            estimates.append(current_V)
            continue

        if method == 'ordinary':
            total_rhoG += rho * G
            count_valid += 1
            V = total_rhoG / count_valid if count_valid > 0 else 0.0
        elif method == 'weighted':
            weighted_sum += rho * G
            weight_sum += rho
            V = weighted_sum / weight_sum if weight_sum > 0 else 0.0

        estimates.append(V)

    return np.array(estimates)

# ================== 主实验：运行多次取平均 MSE ==================

def run_simulation(n_runs: int = 100, max_episodes: int = 10000):
    """
    运行 n_runs 次独立实验，每次学习 max_episodes 幕，
```

```
    返回平均后的 MSE 曲线。
    """
    TRUE_VALUE = -0.27726
    mse_ordinary = np.zeros(max_episodes)
    mse_weighted = np.zeros(max_episodes)

    print(f"开始运行 {n_runs} 次独立实验，每次 {max_episodes} 幕...")

    for run in range(n_runs):
        if (run + 1) % 10 == 0:
            print(f"已完成 {run + 1}/{n_runs} 次实验")

        est_ordinary = estimate_value_over_episodes(max_episodes, method='ordinary')
        est_weighted = estimate_value_over_episodes(max_episodes, method='weighted')

        mse_ordinary += (est_ordinary - TRUE_VALUE) ** 2
        mse_weighted += (est_weighted - TRUE_VALUE) ** 2

    # 取平均
    mse_ordinary /= n_runs
    mse_weighted /= n_runs

    return mse_ordinary, mse_weighted

# ================== 绘图函数：对数横轴 + 中文标注 ==================

def plot_mse_log_scale_chinese(mse_ordinary: np.ndarray, mse_weighted: np.ndarray):
    """
    绘制均方误差曲线，横轴为对数尺度，所有文字为中文
    """
    episodes = np.arange(1, len(mse_ordinary) + 1)

    plt.figure(figsize=(10, 6))
    plt.semilogx(episodes, mse_ordinary,
                 label='普通重要性采样',
                 color='tab:blue', linewidth=2, alpha=0.8)
    plt.semilogx(episodes, mse_weighted,
                 label='加权重要性采样',
                 color='tab:orange', linewidth=2, alpha=0.8)

    plt.xlabel('幕数（对数尺度）', fontsize=12)
    plt.ylabel('均方误差（MSE）', fontsize=12)
    plt.title('二十一点游戏中离轨策略的状态价值估计\n'
              '100次独立运行的平均结果（例5.4）',
              fontsize=14)
    plt.legend(fontsize=11, loc='upper right')
    plt.grid(True, which="both", linestyle='--', alpha=0.5)
    plt.tight_layout()

    # 保存图像（中文兼容路径）
    plt.savefig("blackjack_off_policy_mse_zh.png", dpi=200, bbox_inches='tight',
format='png')
    plt.show()

# ================== 主程序入口 ==================

if __name__ == "__main__":
    # 设置参数
    N_RUNS = 100         # 独立运行次数
    MAX_EPISODES = 10000  # 每次运行的最大幕数
```

```
    # 运行仿真
    mse_ordinary, mse_weighted = run_simulation(n_runs=N_RUNS, max_episodes=MAX_EPISODES)

    # 绘图（中文）
    plot_mse_log_scale_chinese(mse_ordinary, mse_weighted)

    # 输出最终MSE（控制台仍可用英文）
    print(f"\n最终均方误差（{N_RUNS}次平均）:")
    print(f"普通重要性采样: {mse_ordinary[-1]:.6f}")
    print(f"加权重要性采样: {mse_weighted[-1]:.6f}")
```



在估计同一个真实价值的过程中，**加权重要性采样比普通重要性采样具有更低的均方误差**，尤其是在学习初期，体现了其更高的样本效率和更小的方差。

该例题表明普通重要性采样的估计通常具有**无限方差**，当轨迹包含循环时尤其明显。

- **状态空间**：只有**一个非终止状态** s 和一个终止状态
  - 这是最简单的非平凡MDP结构
  - 终止状态无特殊标识，达到后游戏结束
- **动作空间**：两个动作
  - **左动作**：
    - 以**0.9概率**转移到**自身状态** s（形成循环）
    - 以**0.1概率**转移到**终止状态**，并获得**+1奖励**
  - **右动作**：
    - **确定性**转移到终止状态
    - **无奖励**（隐含为0）
- **奖励机制**：
  - 仅当通过左动作终止时获得+1奖励
  - 其他所有转移的奖励均为0
  - 折扣因子 $\gamma = 1$（无折扣）

- **目标策略** π：
  - **总是选择"左"动作**：$\pi(\text{左}|s) = 1$，$\pi(\text{右}|s) = 0$
  - 这是一个**确定性策略**，也是该MDP的**最优策略**
  - 在该策略下，状态 s 的真实价值为：$v_\pi(s) = 1$
- **行为策略** b：
  - **随机选择**：$b(\text{左}|s) = b(\text{右}|s) = 0.5$
  - 这是一个**随机策略**，用于生成数据

1. **初始化**：每次实验从**非终止状态** s 开始
2. **行为策略执行**：
   - 在状态 s，行为策略 b 以50%概率选择"左"，50%概率选择"右"
   - 如果选择"右"：立即终止，回报为0
   - 如果选择"左"：
     - 以90%概率回到 s，继续游戏
     - 以10%概率终止，回报为+1
     - 如果回到 s，重复此过程
3. **片段收集**：
   - 每次从 s 开始到终止构成一个片段
   - 片段长度可能为1（选择右，或选择左后立即终止）
   - 片段长度也可能很长（多次选择左并回到 s）
   - 记录从 s 开始的回报 G_t（0或1）

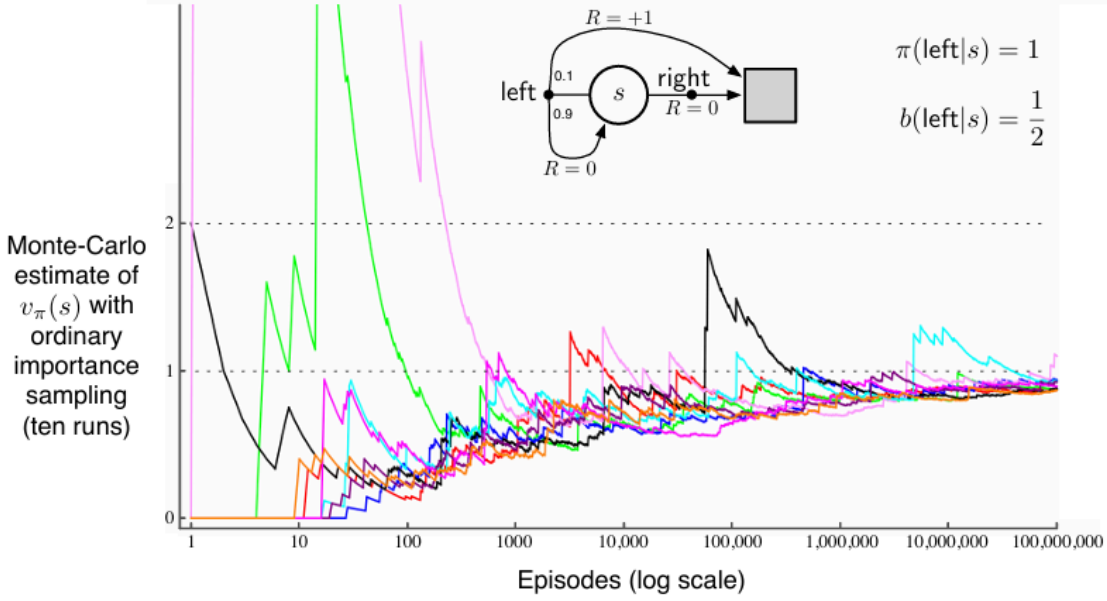
- 在目标策略 π（总是选择左）下：
  - 有0.1概率立即获得+1回报
  - 有0.9×0.1概率经过一次循环后获得+1回报
  - 有0.9^2×0.1概率经过两次循环后获得+1回报
  - 以此类推...
- 状态价值计算：

$$v_\pi(s) = 0.1 \times 1 + 0.9 \times 0.1 \times 1 + 0.9^2 \times 0.1 \times 1 + \cdots = 0.1 \sum_{k=0}^{\infty} 0.9^k = \frac{0.1}{1 - 0.9} = 1$$

- **直观解释**：长期来看，平均每10次尝试会有1次成功终止并获得+1回报，因此期望回报为1。

实验结果

- **图5.4下部**显示了十次独立运行的结果
- 即使在数百万个片段后，估计值仍在剧烈波动
- 估计值从未稳定收敛到真实值1
- 有时接近1，有时突然飙升到100、1000甚至更高



在离轨策略蒙特卡洛方法中，使用**普通重要性采样**（Ordinary Importance Sampling）估计目标策略价值时，即使其期望无偏，也可能因**方差无穷**而导致估计值永不收敛。

考虑行为策略 b 生成的数据用于估计目标策略 π 的价值，若某幕以“右”动作结束，则其回报为 0；但由于目标策略 π 永远不会选择“右”动作，因此该轨迹的重要性采样比：

$$\rho = \prod_{t=0}^{T-1} \frac{\pi(A_t|S_t)}{b(A_t|S_t)} = 0$$

因为存在一步满足 $\pi(\text{右}|s) = 0$。 这类轨迹对估计无贡献，可完全忽略。

唯一能贡献估计的轨迹是那些全程执行“左”动作并最终终止的路径，其共同特征为：

- 所有动作均为“左”
- 回报恒为 $G_0 = 1$
- 经历若干次状态循环后终止

加权重要性采样采用如下形式估计价值：

$$V = \frac{\sum \rho_i G_i}{\sum \rho_i}$$

其中ρ_i是重要性采样比

由于所有有效轨迹的回报都是 1，该估计等价于：

$$V = \frac{\sum \rho_i \cdot 1}{\sum \rho_i} = 1$$

即：只要权重一致归一化，估计值始终为 1 —— **准确且稳定。**

因此，加权重要性采样在此例中表现良好。

普通重要性采样使用未归一化的平均：

$$V_n = \frac{1}{n} \sum_{i=1}^{n} \rho_i G_i$$

虽然大多数轨迹较短、ρ_i 较小，但长轨迹的 ρ_i 随长度指数增长，导致个别极端样本剧烈扰动整体估计。

更严重的是，**这种估计的方差实际上是无穷大。**

> [!NOTE]
>
> > **G_0 表示从时间步 $t=0$ 开始，到 episode 终止为止的总回报（return）。**
>
> 数学上，它定义为：
>
> $$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots + \gamma^{T-1} R_T$$
>
> 其中：
>
> - $R_1, R_2, \ldots, R_T$：后续各步的奖励
> - γ：折扣因子（本例中 $\gamma = 1$）
> - T：episode 的终止时间步
>
> G_0 就是从初始状态出发所能获得的**累积奖励**，也就是预测的目标值。因为：
>
> $$\mathbb{E}_b[\rho G_0] = v_\pi(s)$$

我们考察随机变量 ρG_0 的方差。由方差定义：

$$\mathrm{Var}[\rho G_0] = \mathbb{E}_b\left[(\rho G_0)^2\right] - \left(\mathbb{E}_b[\rho G_0]\right)^2$$

已知 $\mathbb{E}_b[\rho G_0] = v_\pi(s) = 1$（无偏性），故若要 $\mathrm{Var} = \infty$，只需证明：

$$\mathbb{E}_b\left[(\rho G_0)^2\right] = \infty$$

由于 $G_0 = 1$ 对所有有效轨迹成立，只需计算：

$$\mathbb{E}_b[\rho^2] = \sum_{\text{有效轨迹}} P_b(\tau) \cdot \rho(\tau)^2$$

设某轨迹经历 k 次循环（共执行 $k+1$ 次“左”动作）后终止：

- 行为策略下发生概率：

$$P_b(k) = (0.5)^{k+1} \cdot (0.9)^k \cdot 0.1$$

- 重要性采样比：

$$\rho_k = \left(\frac{1}{0.5}\right)^{k+1} = 2^{k+1}$$

- 贡献项为：

$$P_b(k) \cdot \rho_k^2 = \left[(0.5)^{k+1} \cdot 0.9^k \cdot 0.1\right] \cdot \left(2^{k+1}\right)^2$$

$$(0.5)^{k+1} \cdot 4^{k+1} = (0.5 \cdot 4)^{k+1} = 2^{k+1}$$

所以：

$$P_b(k) \cdot \rho_k^2 = 0.1 \cdot 0.9^k \cdot 2^{k+1}$$

求和得：

$$\mathbb{E}_b[\rho^2] = \sum_{k=0}^{\infty} 0.1 \cdot 0.9^k \cdot 2^{k+1} = 0.1 \cdot 2 \cdot \sum_{k=0}^{\infty} (0.9 \cdot 2)^k = 0.2 \sum_{k=0}^{\infty} 1.8^k$$

由于公比 $1.8 > 1$，级数发散：

$$\Rightarrow \mathbb{E}_b[\rho^2] = \infty$$

因此：

$$\mathrm{Var}_b[\rho G_0] = \infty$$

> 在存在循环或潜在长轨迹的环境中，**普通重要性采样不可靠。**
> 即使其理论期望正确，**无限方差意味着实际估计永远不会稳定。**
> 因此，在现代离轨算法中，普遍采用**加权、截断或递归形式的重要性采样**（如 Retrace, V-trace）来保证数值稳定性。

练习2

1.**给定用 b 生成的回报，类比公式 (5.6)，用动作价值函数 $Q(s,a)$ 替代状态价值函数 $V(s)$，得到的式子是什么?**

> 原书公式 (5.6) 指的是普通重要性采样的离轨策略价值估计：
>
> $$V(s) \doteq \frac{1}{n}\sum_{i=1}^{n} \rho^{(i)}_{t_i:T(t_i)-1} G_{t_i}$$
>
> 其中 ρ 是从首次访问 s 到终止的重要性采样比。

要将状态价值 $V(s)$ 推广到**动作价值函数 $Q(s,a)$**。

在离轨策略下估计 $Q_\pi(s,a)$，即：当我们在状态 s 执行动作 a 后，遵循目标策略 π 的期望回报。

但由于数据由行为策略 b 生成，我们必须对轨迹加权。关键点是：

- **只有那些在状态 s 执行了动作 a 的轨迹才可用于估计 $Q_\pi(s,a)$**
- 对于这些轨迹，我们计算从该时刻开始的重要性采样比（从动作发生处到终止）

定义：

- 设第 i 条轨迹在某个时间步 t_i 处于状态 $S_{t_i} = s$ 并执行动作 $A_{t_i} = a$
- 令 G_{t_i} 为从该步开始的回报
- 重要性采样比从 t_i 开始计算：

$$\rho_{t_i:T-1} = \prod_{k=t_i}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

类比公式 (5.6)，使用普通重要性采样估计动作价值函数的形式为：

$$Q(s,a) \doteq \frac{1}{n}\sum_{i=1}^{n} \rho^{(i)}_{t_i:T-1} G_{t_i}$$

其中求和是对所有满足 $S_t = s, A_t = a$ 的访问进行的，$\rho^{(i)}_{t_i:T-1}$ 是从该次 (s,a) 对被访问起至 episode 终止的重要性采样比。

> 这是**普通重要性采样**形式；若使用加权版本，则应为：
>
> $$Q(s,a) \doteq \frac{\sum \rho^{(i)}_{t_i:T-1} G_{t_i}}{\sum \rho^{(i)}_{t_i:T-1}}$$

2.**实际在使用普通重要度采样时，与图 5.3 所示的学习曲线一样，错误率随着训练一般下降。但对于加权重要度采样，错误率会先上升然后下降。为什么会这样?**

> 图 5.3 显示的是两种方法在二十一点任务中的 MSE 学习曲线。你观察到的现象——“加权重要性采样初期 MSE 上升”——确实在某些运行中可能出现，原因如下。

虽然加权重要性采样（Weighted IS）长期来看收敛更快、方差更低，但在**早期阶段可能出现 MSE 上升**，主要原因如下：

初始估计偏差（Initial Bias）

加权重要性采样是**有偏估计器**（biased），其偏差随样本数增加而趋于零。

- 初始阶段只有少数几条有效轨迹（与 π 一致）
- 这些轨迹往往具有较高的重要性权重 ρ
- 因此早期估计严重依赖这几个高权重样本
- 若它们对应的 G_0 偏离真实值（如过早终止导致回报偏低），则估计值会产生较大偏差
- 导致 MSE 暂时升高

类比：一开始只看到几个极端案例，就以为整体如此。

小分母问题（Small Denominator Effect）

加权估计形式为：

$$V = \frac{\sum \rho_i G_i}{\sum \rho_i}$$

在前几幕中：

- $\sum \rho_i$ 很小（可能只有 1~2 个非零项）
- 此时任何新加入的轨迹都会剧烈改变估计值
- 尤其是第一个有效轨迹的权重较大时，会引起跳变

结果：估计值震荡 → MSE 波动甚至上升

对比：普通重要性采样为何稳定下降?

普通 IS 使用的是：

$$V = \frac{1}{n} \sum \rho_i G_i$$

- 虽然方差大，但它是无偏的
- 初始估计接近 0（未见有效轨迹时默认为 0）
- 随着有效轨迹加入，逐步向真值移动
- 所以 MSE 更可能是单调下降趋势（尽管最终更高）

> 加权重要性采样初期 MSE 上升的原因是：
>
> - 它是有偏估计，在样本少时受高权重轨迹主导；
> - 初始估计不稳定，易因少数样本产生跳跃；
> - 分母累计不足导致数值敏感；
>
> 而随着更多数据到来，归一化机制发挥作用，方差降低，MSE 最终快速下降并优于普通 IS。

这体现了“**短期偏差 vs 长期稳定性**”的权衡。

3.**在图 5.4 和例 5.5 的结果中采用了首次访问型 MC 方法。假设在同样的问题中采用了每次访问型 MC 方法。估计器的方差仍会是无穷吗？为什么?**

- MDP 只有一个非终止状态 s
- 目标策略 π：总是选择“左”
- 行为策略 b：左右各 0.5
- 轨迹可循环多次，形成很长的访问序列

- 在首次访问型 MC 中，每幕只考虑第一次进入 s 的回报（但这里 s 是唯一状态，所以等价于起点）
- 普通重要性采样估计为：

$$V = \frac{1}{n} \sum_{i=1}^{n} \rho_i G_i$$

已证明其方差无穷

现在问题是：如果改用**每次访问型 MC**，是否还能避免无穷方差？

在每次访问型 MC 中，**状态 s 每出现一次就被视为一个独立的数据点。**

但由于这个 MDP 中：

- s 是唯一非终止状态
- 每次"左"动作返回都再次访问 s
- 所以一条长度为 $k+1$ 的轨迹会在 s 中被"访问" $k+1$ 次

例如：

- 轨迹：左 → 左 → ... → 左（共 $k+1$ 次）→ 终止
- 状态 s 被访问了 $k+1$ 次
- 每次访问后都有后续路径和对应的部分回报（但从那次访问起的回报仍是 1）

注意：从任意一次访问 s 开始到最后的回报 $G_t = 1$（因为最终获得 +1）

并且每次访问的重要性采样比是从该步到终止的乘积。

设某次访问发生在第 j 步（共 $k+1$ 次访问），则：

- 剩余步数：$k+1-j$
- 重要性权重：$\rho_j = 2^{k+1-j}$
- 回报：$G_j = 1$

所以在每次访问型 MC 下，这条轨迹会贡献 $k+1$ 个样本项：$(\rho_j, G_j = 1)$

但是方差仍不会是有限的。理由如下：

即使采用每次访问型方法，我们仍然在估计：

$$\mathbb{E}_b[\rho G]$$

只不过现在样本数量大大增加。

但关键是：**这些新增的样本并不是独立或低方差的。**

更严重的是：

- 长轨迹不仅自身概率低，而且会产生多个高权重项
- 例如一条经历 10 次循环的轨迹会产生 10 个 ρ 值分别为 $2^1, 2^2, \ldots, 2^{10}$ 的项
- 每一项都对平均值有显著影响
- 并且它们来自同一条稀有但极高权重的轨迹

因此，平方期望依然发散：

$$\mathbb{E}_b\left[(\rho G)^2\right] = \infty \quad \text{（推导同前）}$$

所以无论使用**首次访问**还是**每次访问**型 MC，只要使用的是**普通重要性采样**，估计器的方差仍然是无穷的。

增量实现

同策略增量实现

蒙特卡洛预测方法可以以**增量方式**实现，即基于片段逐个进行更新，而非等待所有数据收集完毕再进行批量计算。在多臂老虎机问题中我们平均的是***奖励***，而在蒙特卡洛方法中我们平均的是**回报**。在其余方面，**同策略蒙特卡洛方法**可以使用多臂老虎机的方法实现。

> [!NOTE]
>
> 多臂老虎机（Multi-armed Bandit）假设
>
> - 有 k 个"臂"（即 k 个可选动作）
> - 每个臂 a 有一个未知的奖励分布，其期望值为 $q_*(a)$
> - 每次选择一个臂拉动，会获得一个随机奖励 R
> - 目标是通过尝试不同的臂，**最大化累积奖励**
>
> **关键特点**
>
> 1. **无状态概念**：每次决策都是独立的，没有环境状态变化
> 2. **即时反馈**：每个动作只产生**单步即时奖励**
> 3. **探索-利用权衡**：需要平衡尝试新动作（探索）和选择已知最佳动作（利用）
>
> **价值估计方法**
>
> 在多臂老虎机中，我们通过**样本平均**来估计每个动作的价值：
>
> $$Q_n(a) = \frac{1}{n}\sum_{i=1}^{n} R_i(a)$$
>
> 其中：
>
> - $R_i(a)$ 是第 i 次选择动作 a 时获得的**奖励**
> - n 是选择动作 a 的总次数
>
> 上述平均可以增量实现为：
>
> $$Q_{n+1}(a) = Q_n(a) + \frac{1}{n}[R_{n+1}(a) - Q_n(a)]$$
>
> 这个公式的关键点：
>
> - 我们平均的是**单步即时奖励** R
> - 每次更新只使用当前获得的**单个奖励值**
> - 价值估计 $Q(a)$ 逐渐收敛到期望奖励 $q_*(a)$
>
> 多臂老虎机问题有多种解决策略：
>
> 1. ε-贪婪策略
>
> - **描述**：以概率 $1-\varepsilon$ 选择当前估计价值最高的动作，以概率 ε 随机选择一个动作
> - **特点**：
>   - 简单直观
>   - 探索率固定
>   - 随机探索，不考虑动作的不确定性
>
> 2. UCB（Upper Confidence Bound）
>
> - **描述**：选择具有最高"上置信界"的动作：

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- **特点**：
  - 不是简单的概率选择，而是确定性地选择UCB最高的动作
  - 探索程度随访问次数自动调整
  - 不确定性高的动作有更高探索优先级

3. Softmax（Boltzmann探索）

- **描述**：按Gibbs分布选择动作：

$$P(A_t = a) = \frac{e^{Q_t(a)/\tau}}{\sum_b e^{Q_t(b)/\tau}}$$

- **特点**：
  - 高价值动作有更高概率被选中
  - 温度参数 τ 控制探索程度
  - 不是简单的"选择最大+随机探索"，而是基于价值的软性选择

4. Thompson采样

- **描述**：基于贝叶斯后验分布采样，选择采样值最高的动作
- **特点**：
  - 从后验分布中采样可能的真实价值
  - 选择采样值最高的动作
  - 探索是"智能"的，基于当前知识的不确定性

[!IMPORTANT]

多臂老虎机中平均的是**奖励**

- **奖励**（Reward）：单步即时反馈
  - 在时间步 t 执行动作 A_t 后立即获得的标量值 R_{t+1}
  - 只反映当前动作的直接结果
  - 例如：在老虎机问题中，拉动某个臂后立即获得的硬币数量
- **特点**：
  - 与后续状态和动作无关
  - 仅依赖于当前动作
  - 是马尔可夫决策过程中的基本反馈单元

蒙特卡洛方法中平均的是**回报**

- **回报**（Return）：多步累积反馈
  - 定义为从时间步 t 开始的未来奖励的折扣和：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  - 反映了从当前状态开始的**长期结果**
  - 例如：在二十一点游戏中，从当前状态开始直到游戏结束的总收益
- **特点**：
  - 考虑了当前决策对未来奖励的影响

- 包含了多步奖励的累积效应
- 依赖于整个后续轨迹，而不仅仅是单步

同策略蒙特卡洛方法中，行为策略和目标策略是同一个策略 π。对于状态价值函数的估计，我们可以使用增量实现。

算法步骤

初始化，对于所有 $s \in \mathcal{S}$：

- $V(s) \leftarrow$ 任意实数（如 0）
- $N(s) \leftarrow 0$（访问计数器）

循环（对于每个回合）：

1. 按照策略 π 生成完整回合：$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
2. $G \leftarrow 0$
3. **对于** $t = T-1, T-2, \ldots, 0$：
   - $G \leftarrow \gamma G + R_{t+1}$
   - **如果** S_t 是该回合中首次访问（首次访问方法）：
     - $N(S_t) \leftarrow N(S_t) + 1$
     - $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}[G - V(S_t)]$

增量更新公式

对于状态 s，当它被首次访问时：

$$N(s) \leftarrow N(s) + 1$$
$$V(s) \leftarrow V(s) + \frac{1}{N(s)}[G - V(s)]$$

这与多臂老虎机的增量更新公式形式相同：

$$Q_{n+1}(a) = Q_n(a) + \frac{1}{n}[R - Q_n(a)]$$

关键区别：

- 多臂老虎机：R 是**单步即时奖励**
- 蒙特卡洛方法：G 是**多步回报**（从状态 s 开始的完整轨迹的折扣奖励和）

离策略增量实现

离策略蒙特卡洛需要特殊处理，需要分别考虑使用**普通重要性采样**和使用**加权重要性采样**的方法。

普通重要性采样

普通重要性采样的增量实现通过将回报通过重要性采样比率 $\rho_{t:T(t)-1}$ 进行缩放，然后进行**简单平均**；增量实现与同策略方法类似：

$$V_{n+1} = V_n + \frac{1}{n}(\rho_n G_n - V_n)$$

最后用缩放后的回报 $\rho_n G_n$ 代替多臂老虎机中的奖励。

加权重要性采样

考虑一个回报序列 $G_1, G_2, \cdots, G_{n-1}$，它们都从相同的状态开始，每个回报对应一个随机权重 W_i（例如，$W_i = \rho_{t:T(t)-1}$，即重要性采样比率）。

目标是计算加权平均值：

$$V_n \doteq \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}, \qquad n \geqslant 2,$$

并在获得额外回报 G_n 时能够增量更新该估计值。

为实现增量更新，需要为每个状态维护权重累加和 C_n，其中：

$$C_n = \sum_{k=1}^{n} W_k$$

初始条件：$C_0 = 0$（V_1 是任意的，无需特别指定）。

当获得新的回报 G_n 及其对应权重 W_n 时，更新规则为：

$$V_{n+1} \doteq V_n + \frac{W_n}{C_n}\Big[G_n - V_n\Big], \qquad n \geqslant 1,$$

$$C_{n+1} \doteq C_n + W_{n+1}$$

输入: 任意的目标策略 π
初始化, 对所有 $s \in \mathcal{S}, a \in \mathcal{A}(s)$
　　任意初始化 $Q(s,a) \in \mathbb{R}$
　　$C(s,a) \leftarrow 0$
循环 (对每一幕):
　　选择一个行为策略 b，使得对所有 s, a 满足 $\pi(a|s) > 0 \implies b(a|s) > 0$（即 b 覆盖 π）
　　根据行为策略 b 生成一幕序列: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
　　$G \leftarrow 0$ （初始化累积回报）
　　$W \leftarrow 1$ （初始化重要性采样比率）
　　对幕中的每一步循环, $t = T-1, T-2, \ldots, 0$:
　　　　$G \leftarrow \gamma G + R_{t+1}$
　　　　$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
　　　　$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
　　　　$W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
　　　　如果 $W = 0$, 则退出内层循环（因为后续的重要性采样比率将保持为0）

说明:

- 该算法使用**加权重要度采样**（weighted importance sampling）进行离轨策略学习
- W 表示累积重要性采样比率：$W = \rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$
- $C(s,a)$ 维护的是权重 W 的累加和，用于计算加权平均
- 更新公式 $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$ 实现了增量式加权平均
- 当 $\pi = b$（同轨策略）时，$W = 1$ 恒成立，算法退化为标准的蒙特卡洛方法
- 在无限幕数下，$Q(s,a)$ 收敛到 $q_\pi(s,a)$（对所有被行为策略 b 访问的 (s,a) 对）

这是一个完整的用于蒙特卡洛策略评估的逐幕增量算法。这个算法表面上针对于离轨策略的情况,采用加权重要度采样,但是对于**同轨策略的情况同样适用**,只需要选择同样的目标策略与行动策略即可 (这种情况下 $\pi = b$, W 始终为 1)。近似值 Q 收敛于 q_π（对于所有遇到的 "状态-动作" 二元组), 而动作则根据 b 进行选择, 这个 b 可能是一个不同的策略。

练习

> 对使用首次访问型蒙特卡洛的策略评估（5.1节）进行修改，要求采用 2.4 节中描述的对样本平均的增量式实现。

- **第 5.1 节** 描述了 **首次访问型蒙特卡洛（First-Visit Monte Carlo）策略评估**：
  - 目标是估计某个策略 π 下的状态价值函数 $V^\pi(s)$。
  - 方法：通过多次运行环境得到多个 episode，记录每个状态第一次出现后的累积回报，然后取平均。
- **第 2.4 节** 讲的是 **增量式平均（incremental implementation of sample averages）**：
  - 平均值可以不用保存所有数据，而是逐次更新：

$$Q_{k+1} = Q_k + \frac{1}{k}(R_k - Q_k)$$

  - 这样可以节省内存和计算资源

要实现的就是将传统的"存储所有回报然后求平均"的首次访问型 MC 策略评估，改为使用 **增量式平均法更新状态价值。**

对于每个状态 s，我们维护：

- $V(s)$：当前对该状态价值的估计
- $N(s)$：该状态作为"首次访问"出现在 episode 中的次数

当我们在某 episode 中第一次遇到状态 s，并得到了其后续的累积回报 G，则更新：

$$N(s) \leftarrow N(s) + 1$$
$$V(s) \leftarrow V(s) + \frac{1}{N(s)}[G - V(s)]$$

这就是 **增量式样本平均** 的应用。

> 从式 (5.7) 推导出加权平均的更新规则（式 5.8），推导时遵循非加权规则（式 2.3）的推导思路。

- **式 (2.3)** 是普通（等权重）样本平均的增量更新公式：

$$Q_{k+1} = Q_k + \frac{1}{k}[R_k - Q_k]$$

- **重要性采样（importance sampling）** 用来处理策略评估中的离轨学习。此时每个回报 G_t 都带有一个权重 W_t（即重要性采样比率），因此不能再用简单平均，而要用 **加权平均。**

从加权平均的定义出发：

$$V_n = \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} = \frac{S_n}{C_n}, \quad \text{其中 } S_n = \sum_{k=1}^{n} W_k G_k,\ C_n = \sum_{k=1}^{n} W_k$$

考虑如何从 V_{n-1} 更新到 V_n：

$$V_n = \frac{S_{n-1} + W_n G_n}{C_{n-1} + W_n} = \frac{C_{n-1} V_{n-1} + W_n G_n}{C_{n-1} + W_n}$$

将其变形：

$$V_n = V_{n-1} + \frac{W_n}{C_{n-1} + W_n}(G_n - V_{n-1}) = V_{n-1} + \frac{W_n}{C_n}(G_n - V_{n-1})$$

这与非加权情况下的式 (2.3) $Q_k = Q_{k-1} + \frac{1}{k}(R_k - Q_{k-1})$ 形式相似，只不过这里的步长是 $\frac{W_n}{C_n}$ 而非 $\frac{1}{k}$。

若将 V_{n-1} 记为 V_n，V_n 记为 V_{n+1}，并将第 n 个样本记为 (G_n, W_n)，即可得：

$$V_{n+1} = V_n + \frac{W_n}{\sum_{k=1}^{n} W_k}(G_n - V_n) \tag{5.8}$$

离轨策略蒙特卡洛控制

同轨策略方法在使用某个策略进行控制的同时也对那个策略的价值进行评估；而在离轨策略方法中，这两个工作是分开的。用于生成行动数据的策略称为**行动策略**，它可能与实际上被评估和改善的策略无关；被评估和改善的策略称为**目标策略**。分离的好处是当行动策略能对所有可能的动作继续进行采样时，目标策略可以是确定的（例如贪心策略）。

离轨策略蒙特卡洛控制方法要求行动策略对目标策略可能做出的所有动作都有非零的概率被选择。为了试探所有可能性，行动策略必须是**软性的**（即在所有状态下选择所有动作的概率都非零）。行动策略可以是任何软性策略，且可能在幕间甚至幕内发生变化；但为保证目标策略收敛到最优策略，对每一个“状态-动作”二元组都需要取得无穷多次回报。

离轨策略MC控制算法

用于估计 $\pi \approx \pi_*$ 和 q_*，基于GPI（广义策略迭代）和重要度采样。目标策略 π 是对应Q的贪心策略（Q是对 q_π 的估计），行动策略 b 为任意软性策略。

算法步骤：

初始化：

对所有 $s \in S$，$a \in A(s)$：

$Q(s,a) \in \mathbb{R}$（任意值）

$C(s,a) \leftarrow 0$

$\pi(s) \leftarrow \arg\max_a Q(s,a)$（出现平分时选取方法应保持一致）

无限循环（对每幕）：

$b \leftarrow$ 任意软性策略

根据 b 生成一幕数据：$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

$W \leftarrow 1$

对幕中的每一时刻循环（$t = T-1, T-2, \ldots, 0$）：

$G \leftarrow \gamma G + R_{t+1}$

$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$（出现平分时选取方法应保持一致）

如果 $A_t \neq \pi(S_t)$，则退出内层循环（处理下一幕数据）

$W \leftarrow W \cdot \frac{1}{b(A_t|S_t)}$

一个潜在的问题是：当幕中某时刻后剩下的所有动作都是贪心的时候，这种方法只会从幕的尾部进行学习。如果非贪心的行为较为普遍，则学习速度会很慢，尤其是对于那些在很长的幕中较早出现的状态。这可能会极大地降低学习速度；目前，对于这个问题在离轨策略蒙特卡洛方法中的严重程度尚无足够的研究和讨论。