贝尔曼方程（Bellman Equation）

🔍 一、基础原理

1. **最优性原理（Principle of Optimality）**
   最优策略的子策略必然是最优的，即：无论初始状态和决策如何，剩余决策对剩余问题必构成最优策略1,6,8。

   - 数学表述：若策略 $\pi^$ *最优，则 $ V^{\pi^*}(s) = \max_{a} Q^{\pi^*}(s, a) $。

2. **马尔可夫性质（Markov Property）**
   未来状态仅依赖当前状态和动作，与历史无关1,4,9。

   - 状态转移概率：$ P(s' \mid s, a) $ 表示从状态 $ s $ 执行动作 $ a $ 转移到 $ s' $ 的概率。

📐 二、核心方程形式

1. **标准贝尔曼方程**

- **状态价值函数**:
  $$ V^\pi(s) = \sum_{a} \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \left[r(s, a, s') + \gamma V^\pi(s') \right] $$

  > 描述策略 π 下状态 s 的期望累积折扣奖励1,4,7。

- **动作价值函数**:
  $$ Q^\pi(s, a) = \sum_{s'} P(s' \mid s, a) \left[r(s, a, s') + \gamma \sum_{a'} \pi(a' \mid s') Q^\pi(s', a') \right] $$

  > 关联状态-动作对的长期价值4,7。

2. **贝尔曼最优方程**

- **最优状态价值**:
  $$ V^*(s) = \max_{a} \sum_{s'} P(s' \mid s, a) \left[r(s, a, s') + \gamma V^*(s') \right] $$
- **最优动作价值**:
  $$ Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] $$

  > 通过最大化动作选择实现全局最优策略2,4,7。

3. **向量形式（解析解）**

$$ \mathcal{V} = \mathcal{R} + \gamma \mathcal{P} \mathcal{V} \implies \mathcal{V} = (I - \gamma \mathcal{P})^{-1} \mathcal{R} $$

> 需已知状态转移矩阵 $\mathcal{P}$ 和奖励向量 $\mathcal{R}$，计算复杂度 $O(n^3)$，仅适用于小规模问题5,8。

⚙️ 三、求解方法

方法	原理	适用场景
值迭代	迭代更新 $V(s)$：$V_{k+1}(s) = \max_a \sum_{s'} P(s' \mid s,a) [r + \gamma V_k(s')]$	模型已知（转移概率已知）[2,8]
策略迭代	交替进行策略评估（解 V^π）和策略改进（更新 π）[3]	策略空间明确
Q-learning	时序差分更新：$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$	模型未知（无转移概率）[2,7]

🌐 四、应用领域

1. **强化学习**
    - 求解马尔可夫决策过程（MDP）最优策略（如AlphaGo）[7]。
2. **最优控制**
    - 机器人路径规划、资源调度[2,8]。
3. **经济学**
    - 投资组合优化、库存管理[6,9]。
4. **序列决策问题**
    - NLP词性标注、DNA序列分析[1,4]。

🧩 五、挑战与扩展

1. **维度灾难（Curse of Dimensionality）**
   状态空间随变量数指数增长，需结合函数逼近（如神经网络）[7,8]。
2. **部分可观测问题（POMDP）**
   状态不完全可见时需引入置信状态，NP-Hard问题[7,8]。
3. **连续时间问题**
   需用 **哈密顿-雅可比-贝尔曼方程（HJB方程）** 求解[6,9]：$$\frac{\partial V}{\partial t} + \min_u \left\{ \nabla_x V \cdot f(x,u) + g(x,u) \right\} = 0$$

📚 六、学习资源

1. **经典教材**
    - *Dynamic Programming and Optimal Control*（Bertsekas）[8]
    - *Reinforcement Learning: An Introduction*（Sutton & Barto）[1,4]
2. **代码实践**

```python
# 值迭代伪代码[8](@ref)
def value_iteration(P, R, gamma, max_iter):
    V = np.zeros(len(states))
    for _ in range(max_iter):
        new_V = np.zeros_like(V)
        for s in states:
            new_V[s] = max([sum(P[s][a][s_prime] * (R[s][a][s_prime] + gamma * V[s_prime])
                            for s_prime in states]) for a in actions)
```

```
        V = new_V
    return V
```