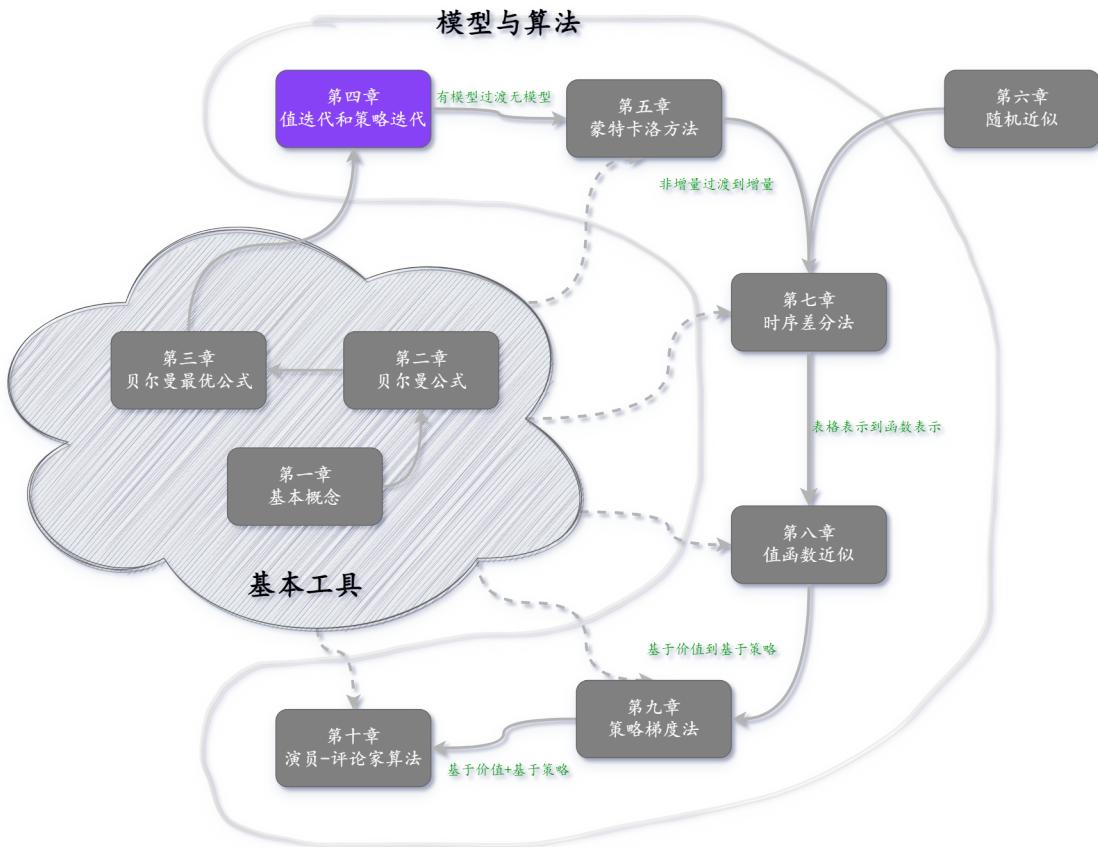


# 第四章 值迭代和策略迭代



通过前几章的准备，我们现在准备好展示第一个可以找到最优策略的算法。本章介绍了三种相互关联的算法。第一种是**值迭代算法（Value Iteration Algorithm）**，这正是上一章讨论的由**收缩映射定理**提出的求解贝尔曼最优方程的算法。在本章中，我们更多地关注该算法的实现细节。第二种是**策略迭代算法（Policy Iteration Algorithm）**，其思想被广泛应用于强化学习算法中。第三种是**截断策略迭代算法（Truncated Policy Iteration Algorithm）**，它是一个统一的算法，包括值迭代算法和策略迭代算法。

本章介绍的算法称为**动态规划算法（Dynamic Programming Algorithm）**[1 2](#)，它需要系统模型。这些算法是后续章节介绍的无模型强化学习算法的重要基础。例如，第五章中介绍的蒙特卡罗算法可以通过扩展本章中介绍的策略迭代算法而立即得到。

## 4.1 值迭代

这一部分介绍值迭代算法。正是上一章（定理 3.3）中介绍的解贝尔曼最优方程的收缩映射定理所提出的算法。具体地说，算法是：

$$v_{k+1} = \max_{\pi \in \Pi} (r_\pi + \gamma P_\pi v_k), \quad k = 0, 1, 2, \dots$$

定理 3.3 保证了当  $k \rightarrow \infty$  时， $v_k$  和  $\pi_k$  分别收敛于最优状态值和最优策略。

该算法是迭代的，每次迭代都有两个步骤：

- 每次迭代的第一步是策略更新（Policy Update）。从数学上来说，它的目标是找到一个可以解决以下优化问题的策略：

$$\pi_{k+1} = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_k),$$

其中  $v_k$  是在上一次迭代中获得的。

- 第二步称为值更新（Value Update）。从数学上讲，它计算出一个新值  $v_{k+1}$ ：

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k, \quad (4.1)$$

其中  $v_{k+1}$  将在下一次迭代中使用。

上面介绍的值迭代算法是矩阵 - 向量的形式。为了实现这个算法，我们需要进一步检查它的元素形式。虽然矩阵 - 向量形式对于理解算法的核心思想很有用，但元素形式对于解释实现细节是必要的。

### 4.1.1 元素形式和实现

考虑时间戳  $k$  和状态  $s$ ：

- 首先，策略更新步骤  $\pi_{k+1} = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_k)$  的元素形式为：

$$\pi_{k+1}(s) = \operatorname{argmax}_{\pi} \sum_a \pi(a|s) \underbrace{\left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s') \right)}_{q_k(s, a)}, \quad s \in \mathcal{S}.$$

我们在 3.3.1 节中表明，可以解决上述优化问题的最优策略是：

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s), \\ 0 & a \neq a_k^*(s), \end{cases} \quad (4.2)$$

其中  $a_k^*(s) = \operatorname{argmax}_a q_k(s, a)$ 。如果  $a_k^*(s) = \operatorname{argmax}_a q_k(s, a)$  有多个解，我们可以选择其中任何一个而不影响算法的收敛性。由于新策略  $\pi_{k+1}$  选择具有最大  $q_k(s, a)$  的动作，因此这样的策略被称为贪婪的。

- 其次，值更新步骤  $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$  的元素形式为：

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a|s) \underbrace{\left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s') \right)}_{q_k(s, a)}, \quad s \in \mathcal{S}.$$

将 (4.2) 代入上式可得：

$$v_{k+1}(s) = \max_a q_k(s, a).$$

综上所述，上述步骤可以表示为：

$$v_k(s) \rightarrow q_k(s, a) \rightarrow \text{new greedy policy } \pi_{k+1}(s) \rightarrow \text{new value } v_{k+1}(s) = \max_a q_k(s, a)$$

算法 4.1 总结了实现细节。

## ❤ 算法 4.1：值迭代算法

**初始化：**所有  $(s, a)$  的概率模型  $p(r|s, a)$  和  $p(s'|s, a)$  已知，初步猜测  $v_0$ 。

**目标：**搜索最优状态值和求解贝尔曼最优方程的最优策略。

**While**  $v_k$  未收敛，因为  $\|v_k - v_{k-1}\|$  大于阈值，对于第  $k$  次迭代，**do**

**For** 每个  $s \in \mathcal{S}$ , **do**

**For** 每个  $a \in \mathcal{A}(s)$ , **do**

$$Q\text{-值: } q_k(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_k(s')$$

$$\text{最大化动作值: } a_k^*(s) = \operatorname{argmax}_a q_k(s, a)$$

$$\text{策略更新: If } a = a_k^*, \pi_{k+1}(a|s) = 1; \text{ Else } \pi_{k+1}(a|s) = 0$$

$$\text{值更新: } v_{k+1}(s) = \max_a q_k(s, a)$$

一个可能令人困惑的问题是 (4.1) 中的  $v_k$  是否是一个状态值。答案是不。虽然  $v_k$  最终收敛到最优状态值，但并不能保证满足任何策略的贝尔曼方程。例如，一般不满足  $v_k = r_{\pi_k} + \gamma P_{\pi_{k+1}} v_k$  或  $v_k = r_{\pi_k} + \gamma P_{\pi_k} v_k$ 。它只是算法生成的中间值。另外，由于  $v_k$  不是状态值，所以  $q_k$  也不是动作值。

### 4.1.2 说明性示例

接下来我们通过一个例子来说明值迭代算法的逐步实现。此示例是一个带有 1 个禁区的  $2 \times 2$  网格（图 4.2）。目标区域是  $s_4$ 。奖励设置为  $r_{\text{boundary}} = r_{\text{forbidden}} = -1$  和  $r_{\text{target}} = 1$ 。折扣因子为  $\gamma = 0.9$ 。

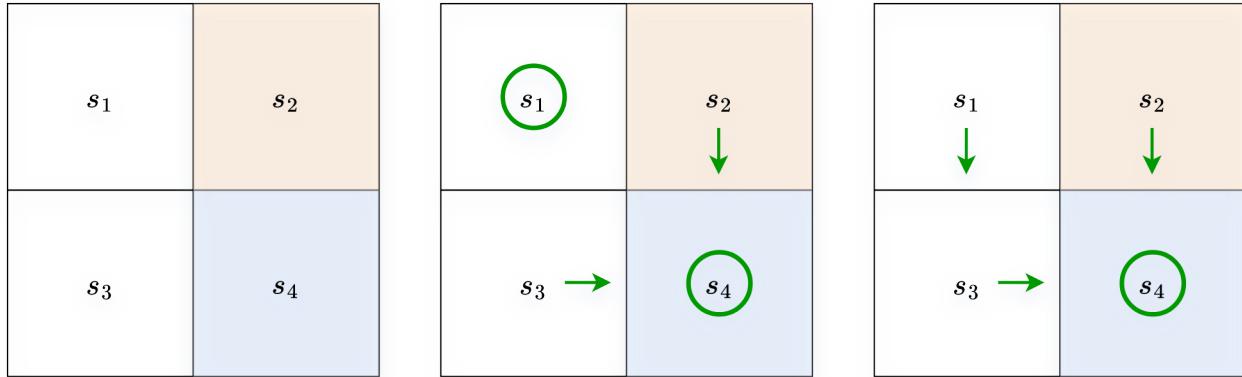


图 4.2

Q-Table	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	$-1 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$	$0 + \gamma v(s_3)$	$-1 + \gamma v(s_1)$	$0 + \gamma v(s_1)$
$s_2$	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_4)$	$0 + \gamma v(s_1)$	$-1 + \gamma v(s_2)$
$s_3$	$0 + \gamma v(s_1)$	$1 + \gamma v(s_4)$	$-1 + \gamma v(s_3)$	$-1 + \gamma v(s_3)$	$0 + \gamma v(s_3)$
$s_4$	$-1 + \gamma v(s_2)$	$-1 + \gamma v(s_4)$	$-1 + \gamma v(s_4)$	$0 + \gamma v(s_3)$	$1 + \gamma v(s_4)$

表 4.1

每个状态-动作对的  $q$  值表达式如表 4.1 所示。

- $k = 0$ :

不失一般性，选择初始值为  $v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$ 。

**$q$  值计算：**将  $v_0(s_i)$  代入表 4.1，得到表 4.2 所示的  $q$  值。

Q-Table	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	-1	-1	0	-1	0
$s_2$	-1	-1	1	0	-1
$s_3$	0	1	-1	-1	0
$s_4$	-1	-1	-1	0	1

表 4.2

**策略更新：** $\pi_1$  是通过为每个状态选择具有最大  $q$  值的动作来获得的：

$$\pi_1(a_5|s_1) = 1, \quad \pi_1(a_3|s_2) = 1, \quad \pi_1(a_2|s_3) = 1, \quad \pi_1(a_5|s_4) = 1.$$

该策略如图 4.2（中间子图）所示。很明显，这个策略不是最优的，因为它选择保持在  $s_1$  不变。值得注意的是， $(s_1, a_5)$  和  $(s_1, a_3)$  的  $q$  值实际上是相同的，我们可以随机选择任一动作。

**值更新：** $v_1$  是通过将  $v$  值更新为每个状态的最大  $q$  值来获得的：

$$v_1(s_1) = 0, \quad v_1(s_2) = 1, \quad v_1(s_3) = 1, \quad v_1(s_4) = 1.$$

- $k = 1$ :

**$q$  值计算：**将  $v_1(s_i)$  代入表 4.1，得到表 4.3 所示的  $q$  值。

Q-Table	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$
$s_1$	$-1 + \gamma 0$	$-1 + \gamma 1$	$0 + \gamma 1$	$-1 + \gamma 0$	$0 + \gamma 0$
$s_2$	$-1 + \gamma 1$	$-1 + \gamma 1$	$1 + \gamma 1$	$0 + \gamma 0$	$-1 + \gamma 1$
$s_3$	$0 + \gamma 0$	$1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$
$s_4$	$-1 + \gamma 1$	$-1 + \gamma 1$	$-1 + \gamma 1$	$0 + \gamma 1$	$1 + \gamma 1$

表 4.3

**策略更新：** $\pi_2$  是通过选择最大  $q$  值获得的：

$$\pi_2(a_3|s_1) = 1, \quad \pi_2(a_3|s_2) = 1, \quad \pi_2(a_2|s_3) = 1, \quad \pi_2(a_5|s_4) = 1.$$

该策略如图 4.2（右子图）所示。

**值更新：** $v_2$  是通过将  $v$  值更新为每个状态的最大  $q$  值来获得的：

$$v_2(s_1) = \gamma 1, \quad v_2(s_2) = 1 + \gamma 1, \quad v_2(s_3) = 1 + \gamma 1, \quad v_2(s_4) = 1 + \gamma 1.$$

- $k = 2, 3, 4 \dots$

值得注意的是，如图 4.2(c) 所示，策略  $\pi_2$  已经是最优的。因此，在这个简单的例子中，我们只需要运行两次迭代即可获得最优策略。对于更复杂的例子，我们需要运行更多迭代，直到  $v_k$  的值收敛（例如，直到  $\|v_{k+1} - v_k\|$  小于预先指定的阈值）。

## 4.2 策略迭代

本节介绍另一个重要的算法：**策略迭代 (Policy Iteration)**。与值迭代不同，策略迭代不是直接求解贝尔曼最优方程。然而，它与值迭代有着密切的关系，如下所示。此外，策略迭代的思想非常重要，因为它广泛应用于强化学习算法中。

### 4.2.1 算法分析

策略迭代是一种迭代算法。每次迭代都有两个步骤：

- 第一个是**策略评估 (Policy Evaluation)** 步骤。顾名思义，此步骤通过计算相应状态值来评估给定的策略。即求解如下贝尔曼方程：

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}, \quad (4.3)$$

其中  $\pi_k$  是上一次迭代得到的策略， $v_{\pi_k}$  是要计算的状态值。 $r_{\pi_k}$  和  $P_{\pi_k}$  的值可以从系统模型中获得。

- 第二个是**策略提升 (Policy Improvement)**。顾名思义，此步骤用于改进策略。特别是，一旦第一步计算出  $v_{\pi_k}$ ，就可以得到新的策略  $\pi_{k+1}$ ：

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}}(r_{\pi} + \gamma P_{\pi} v_{\pi_k}).$$

上面对算法的描述自然伴随着三个问题：

- ✓ 在策略评估步骤中，如何求解状态值  $v_{\pi_k}$ ？
- ✓ 在策略提升步骤中，为什么新策略  $\pi_{k+1}$  比  $\pi_k$  更好？
- ✓ 为什么这个算法最终能够收敛到最优策略呢？

接下来我们一一解答这些问题：

#### 1. 在策略评估步骤中，如何求解状态值 $v_{\pi_k}$ ？

我们在第 2 章中介绍了两种求解 (4.3) 中贝尔曼方程的方法。接下来我们简要回顾一下这两种方法。第一种方法是解析解： $v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$ 。这种封闭式解对于理论分析很有用，但实现起来效率低下，因为它需要其他数值算法来计算矩阵逆。第二种方法是迭代算法，可以很容易实现：

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \dots \quad (4.4)$$

其中  $v_{\pi_k}^{(j)}$  表示  $v_{\pi_k}$  的第  $j$  个估计。从任何初始猜测  $v_{\pi_k}^{(0)}$  开始，可以确保当  $j \rightarrow \infty$  时， $v_{\pi_k}^{(j)} \rightarrow v_{\pi_k}$  成立。详细信息请参见第 2.7 节。

有趣的是，策略迭代是一种迭代算法，策略评估步骤中嵌入了另一个迭代算法(4.4)。理论上，这种嵌入式迭代算法需要无限数量的步骤（即  $j \rightarrow \infty$ ）才能收敛到真实的状态值  $v_{\pi_k}$ 。然而，这是不可能实现的。实际上，当满足某个标准时，迭代过程终止。例如，终止标准可以是  $\|v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)}\|$  小于预先指定的阈值或者  $j$  超过预先指定的值。如果我们不运行无限次迭代，我们只能获得一个不精确的  $v_{\pi_k}$  值，该值将用于后续的策略提升步骤。这会引起问题吗？答案是不会。当我们稍后在 4.3 节中介绍截断策略迭代算法时，原因就会变得清晰。

## 2. 在策略提升步骤中，为什么新策略 $\pi_{k+1}$ 比 $\pi_k$ 更好？

策略改进步骤可以改进给定的策略，如下所示。

### 引理 4.1 (策略提升)

如果  $\pi_{k+1} = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_k})$ ，则  $v_{\pi_{k+1}} \geq v_{\pi_k}$ 。

这里， $v_{\pi_{k+1}} \geq v_{\pi_k}$  意味着对于所有  $s$ ， $v_{\pi_{k+1}}(s) \leq v_{\pi_k}(s)$ 。该引理的证明在框 4.1 中给出。

#### □ 4.1 引理 4.1 的证明：

由于  $v_{\pi_{k+1}}$  和  $v_{\pi_k}$  是状态值，因此它们满足贝尔曼方程：

$$\begin{aligned} v_{\pi_{k+1}} &= r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}, \\ v_{\pi_k} &= r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}. \end{aligned}$$

由于  $\pi_{k+1} = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_k})$ ，我们知道：

$$r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k} \geq r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

那么接下来：

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} &= (r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}) - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}). \end{aligned}$$

因此：

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} &\leq \gamma^2 P_{\pi_{k+1}}^2 (v_{\pi_k} - v_{\pi_{k+1}}) \leq \dots \leq \gamma^n P_{\pi_{k+1}}^n (v_{\pi_k} - v_{\pi_{k+1}}) \\ &\leq \lim_{n \rightarrow \infty} \gamma^n P_{\pi_{k+1}}^n (v_{\pi_k} - v_{\pi_{k+1}}) = 0. \end{aligned}$$

该极限是由于当  $n \rightarrow \infty$ ， $\gamma^n \rightarrow 0$  且  $P_{\pi_{k+1}}^n$  对于任何  $n$  都是非负随机矩阵。这里，随机矩阵是指所有行的行和都等于 1 的非负矩阵。

## 3. 为什么这个算法最终能够收敛到最优策略呢？

策略迭代算法生成两个序列。第一个是策略序列： $\{\pi_0, \pi_1, \dots, \pi_k, \dots\}$ 。第二个是状态值序列： $\{v_{\pi_0}, v_{\pi_1}, \dots, v_{\pi_k}, \dots\}$ 。假设  $v^*$  是最优状态值。然后，对于所有  $k$ ， $v_{\pi_k} \leq v^*$ 。由于根据引理 4.1 不断改进策略，我们知道：

$$v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \dots \leq v_{\pi_k} \leq \dots \leq v^*.$$

由于  $v_{\pi_k}$  是非递减的，并且始终以  $v^*$  为界，因此根据单调收敛定理<sup>3</sup>，当  $k \rightarrow \infty$  时， $v_{\pi_k}$  收敛到一个常数值，表示为  $v_\infty$ 。下面的分析表明  $v_\infty = v^*$ 。

### 定理 4.1 (策略迭代的收敛性)

策略迭代算法生成的状态值序列  $\{v_{\pi_k}\}_{k=0}^\infty$  收敛到最优状态值  $v^*$ 。结果，策略序列  $\{\pi_k\}_{k=0}^\infty$  收敛到最优策略。

该定理的证明在框 4.2 中给出。证明不仅展示了策略迭代算法的收敛性，而且揭示了策略迭代和值迭代算法之间的关系。宽松地说，如果两种算法从相同的初始猜测开始，由于策略评估步骤中嵌入了额外的迭代，策略迭代会比值迭代收敛得更快。当我们在 4.3 节中介绍截断策略迭代算法时，这一点会变得更加清晰。

### □ 4.2 定理 4.2 的证明：

证明的想法是证明策略迭代算法会比值迭代算法收敛得更快。特别地，为了证明  $\{v_{\pi_k}\}_{k=0}^\infty$  的收敛性，我们引入另一个序列  $\{v_k\}_{k=0}^\infty$  生成：

$$v_{k+1} = f(v_k) = \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k).$$

这种迭代算法就是值迭代算法。我们已经知道，当给定任何初始值  $v_0$  时， $v_k$  收敛于  $v^*$ 。

对于  $k = 0$ ，我们总能找到一个  $v_0$ ，使得对于任何  $\pi_0$ ， $v_{\pi_0} \geq v_0$ 。

接下来我们通过归纳法证明对于所有  $k$ ， $v_k \leq v_{\pi_k} \leq v^*$ 。

对于  $k \geq 0$ ，假设  $v_{\pi_k} \geq v_k$ 。

对于  $k + 1$ ，我们有：

$$\begin{aligned} v_{\pi_{k+1}} - v_{k+1} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k) \\ &\geq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - \max_{\pi} (r_{\pi} + \gamma P_{\pi} v_k) \\ &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\ &\geq (r_{\pi'_k} + \gamma P_{\pi'_k} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\ &= \gamma P_{\pi'_k} (v_{\pi_k} - v_k). \end{aligned}$$

由于  $v_{\pi_k} - v_k \geq 0$  并且  $P_{\pi'_k}$  是非负的，我们有  $P_{\pi'_k} (v_{\pi_k} - v_k) \geq 0$ ，因此  $v_{\pi_{k+1}} - v_{k+1} \geq 0$ 。

因此，我们可以通过归纳法证明，对于任意  $k \geq 0$ ， $v_k \leq v_{\pi_k} \leq v^*$ 。由于  $v_k$  收敛于  $v^*$ ，所以  $v_{\pi_k}$  也收敛于  $v^*$ 。

## 4.2.2 元素形式和实现

为了实现策略迭代算法，我们需要研究其元素形式。

- 首先，策略评估步骤使用 (4.4) 中的迭代算法从  $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$  求解  $v_{\pi_k}$ 。该算法的元素形式是：

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}^{(j)}(s') \right), \quad s \in \mathcal{S},$$

其中  $j = 0, 1, 2, \dots$ 。

- 其次，策略提升步骤求解  $\pi_{k+1} = \text{argmax}_{\pi}(r_{\pi} + \gamma P_{\pi} v_{\pi_k})$ 。该方程的元素形式为：

$$\pi_{k+1}(s) = \text{argmax}_{\pi} \sum_a \pi(a|s) \underbrace{\left( \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s') \right)}_{q_{\pi_k}(s, a)}, \quad s \in \mathcal{S},$$

其中  $q_{\pi_k}(s, a)$  是策略  $\pi_k$  下的动作值。设  $a_k^*(s) = \text{argmax}_a q_{\pi_k}(s, a)$ 。那么，贪心最优策略为：

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s), \\ 0 & a \neq a_k^*(s), \end{cases}$$

算法 4.2 总结了实现细节：

### 算法 4.2：策略迭代算法

**初始化：**所有  $(s, a)$  的系统模型  $p(r|s, a)$  和  $p(s'|s, a)$  已知。初始猜测  $\pi_0$ 。

**目标：**寻找最优状态值和最优策略。

**While**  $v_{\pi_k}$  尚未收敛，对于第  $k$  次迭代，**do**

策略评估：

初始化：任意初始猜测  $v_{\pi}(0)$

**While**  $v_{\pi_k}^{(j)}$  尚未收敛时，对于第  $j$  次迭代，**do**

**For** 每个状态  $s \in \mathcal{S}$ ，**do**

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[ \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}^{(j)}(s') \right]$$

策略提升：

**For** 每个状态  $s \in \mathcal{S}$ ，**do**

**For** 每个动作  $a \in \mathcal{A}$ ，**do**

$$q_{\pi_k}(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s')$$

$$a_k^*(s) = \text{argmax}_a q_{\pi_k}(s, a)$$

**If**  $a = a_k^*$ ，则  $\pi_{k+1}(a|s) = 1$ ，**Else**  $\pi_{k+1}(a|s) = 0$

### 4.2.3 说明性示例

- 一个简单的示例

考虑图 4.3 所示的一个简单示例。有两种状态具有三种可能的动作： $\mathcal{A} = \{a_\ell, a_0, a_r\}$ 。这三个动作分别代表左移、不变、右移。奖励设置为  $r_{\text{boundary}} = -1$  和  $r_{\text{target}} = 1$ 。折扣因子为  $\gamma = 0.9$ 。

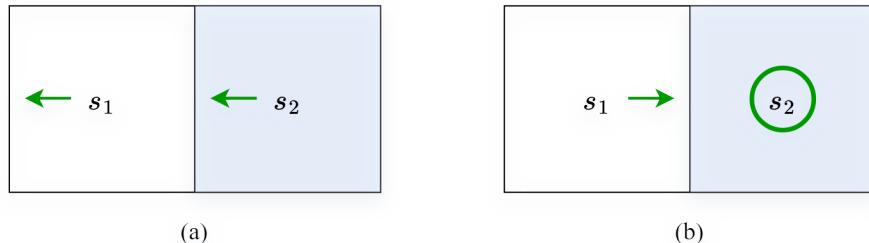


图 4.3

接下来我们将逐步介绍策略迭代算法的实现。当  $k = 0$  时，我们从图 4.3(a) 所示的初始策略开始。这个策略不好，因为它没有走向目标区域。接下来我们将展示如何应用策略迭代算法来获得最优策略。

- 首先，在策略评估步骤中，我们需要求解贝尔曼方程：

$$\begin{aligned} v_{\pi_0}(s_1) &= -1 + \gamma v_{\pi_0}(s_1), \\ v_{\pi_0}(s_2) &= 0 + \gamma v_{\pi_0}(s_1). \end{aligned}$$

由于方程很简单，可以手动求解：

$$v_{\pi_0}(s_1) = -10, \quad v_{\pi_0}(s_2) = -9.$$

实际中，方程可以通过 (4.4) 中的迭代算法求解。例如，选择初始状态值为  $v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0$ 。由 (4.3) 得出

$$\begin{aligned} \begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases} \\ \begin{cases} v_{\pi_0}^{(2)}(s_1) = -1 + \gamma v_{\pi_0}^{(1)}(s_1) = -1.9, \\ v_{\pi_0}^{(2)}(s_2) = 0 + \gamma v_{\pi_0}^{(1)}(s_1) = 0.9, \end{cases} \\ \begin{cases} v_{\pi_0}^{(3)}(s_1) = -1 + \gamma v_{\pi_0}^{(2)}(s_1) = -2.71, \\ v_{\pi_0}^{(3)}(s_2) = 0 + \gamma v_{\pi_0}^{(2)}(s_1) = -1.71, \end{cases} \\ \vdots \end{aligned}$$

随着迭代次数的增加，我们可以看到随着  $j$  的增加， $v_{\pi_0}^{(j)}(s_1) \rightarrow v_{\pi_0}(s_1) = -10$  和  $v_{\pi_0}^{(j)}(s_2) \rightarrow v_{\pi_0}(s_2) = -9$  的趋势。

- 其次，在策略提升步骤中，关键是计算每个状态 - 动作对的  $q_{\pi_0}(s, a)$ 。下面的  $q$  表可以用来演示这样的过程：

$q_{\pi_k}(s, a)$	$a_\ell$	$a_0$	$a_r$
$s_1$	$-1 + \gamma v_{\pi_k}(s_1)$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$
$s_2$	$0 + \gamma v_{\pi_k}(s_1)$	$1 + \gamma v_{\pi_k}(s_2)$	$-1 + \gamma v_{\pi_k}(s_2)$

表 4.4

将之前的策略评估步骤中获得的  $v_{\pi_0}(s_1) = -10$ 、 $v_{\pi_0}(s_2) = -9$  代入表 4.4，得到表 4.5。

$q_{\pi_0}(s, a)$	$a_\ell$	$a_0$	$a_r$
$s_1$	-10	-9	-7.1
$s_2$	-9	-7.1	-9.1

表 4.5

通过求  $q_{\pi_0}$  的最大值，可以得到改进策略  $\pi_1$  为：

$$\pi_1(a_r|s_1) = 1, \quad \pi_1(a_0|s_2) = 1.$$

该策略如图 4.3(b) 所示。显然，这个政策是最优的。

上述过程表明，在这个简单的例子中，单次迭代足以找到最优策略。更复杂的示例需要更多的迭代。

### • 一个更复杂的例子

	1	2	3	4	5
1	○	○	○	○	○
2	○	○	○	○	○
3	○	○	○	○	○
4	○	○	○	○	○
5	○	○	○	○	○

(a)  $\pi_0$  and  $v_{\pi_0}$ 

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	-100.0	-100.0	0.0	0.0
3	0.0	0.0	-100.0	0.0	0.0
4	0.0	-100.0	10.0	-100.0	0.0
5	0.0	-100.0	0.0	0.0	0.0

(b)  $\pi_1$  and  $v_{\pi_1}$ 

	1	2	3	4	5
1	○	○	→	○	○
2	○	↑	↑	↓	←
3	↑	←	↓	→	←
4	↑	→	○	←	○
5	↑	→	↑	←	↑

(c)  $\pi_2$  and  $v_{\pi_2}$ 

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	0.0
5	0.0	9.0	10.0	9.0	0.0

(d)  $\pi_3$  and  $v_{\pi_3}$ 

	1	2	3	4	5
1	○	←	←	○	↓
2	○	↓	↑	↓	←
3	○	←	↓	○	↓
4	↑	→	○	←	↓
5	↑	→	↑	←	←

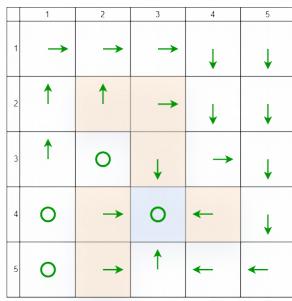
(e)  $\pi_4$  and  $v_{\pi_4}$ 

	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	6.6
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

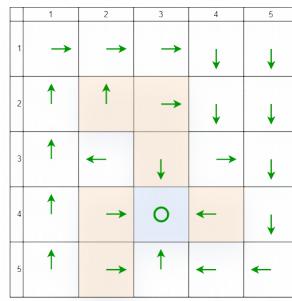
(f)  $\pi_5$  and  $v_{\pi_5}$ 

⋮

⋮

(g)  $\pi_9$  and  $v_{\pi_9}$ 

1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	0.0	10.0	5.9	6.6
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

(g)  $\pi_9$  and  $v_{\pi_9}$ (h)  $\pi_{10}$  and  $v_{\pi_{10}}$ 

接下来我们使用图 4.4 所示的更复杂的示例来演示策略迭代算法。奖励设置为  $r_{\text{boundary}} = -1$ 、 $r_{\text{forbidden}} = -10$ 、 $r_{\text{target}} = 1$ 。折扣因子为  $\gamma = 0.9$ 。当从随机初始策略（图 4.4(a)）开始时，策略迭代算法可以收敛到最优策略（图 4.4(h)）。

在迭代过程中观察到两个有趣的现象：

- 首先，如果我们观察策略如何演变，一个有趣的模式是，靠近目标区域的状态比远离目标区域的状态更早找到最优策略。只有较近的状态能够先找到到达目标的轨迹，较远的状态才能找到经过较近的状态到达目标的轨迹。
- 其次，状态值的空间分布呈现出一个有趣的模式：距离目标越近的状态具有越大的状态值。这种模式的原因是，从更远的状态开始的智能体必须走很多步才能获得正奖励。此类奖励将大幅折扣，因此相对较小。

## 4.3 截断策略迭代

接下来我们介绍一种更通用的算法，称为**截断策略迭代 (Truncated Policy Iteration)**。我们将看到，值迭代和策略迭代算法是截断策略迭代算法的两种特殊情况。

### 4.3.1 值迭代和策略迭代的比较

首先，我们通过如下列出步骤来比较值迭代和策略迭代算法。

- **策略迭代：**选择任意初始策略  $\pi_0$ 。在第  $k$  次迭代中，执行以下两个步骤：

- 步骤 1：策略评估 (PE)。给定  $\pi_k$ ，求解  $v_{\pi_k}$ ：

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

- 步骤 2：政策改进 (PI)。给定  $v_{\pi_k}$ ，求解  $\pi_{k+1}$ ：

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}}(r_{\pi} + \gamma P_{\pi} v_{\pi_k}).$$

- **值迭代：**选择任意初始值  $v_0$ 。在第  $k$  次迭代中，执行以下两个步骤：

- 步骤 1：策略更新 (PU)。给定  $v_k$ ，求解  $\pi_{k+1}$ ：

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}}(r_{\pi} + \gamma P_{\pi} v_k).$$

- 步骤 2：值更新 (VU)。给定  $\pi_{k+1}$ ，求解  $v_{k+1}$ ：

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k.$$

上述两种算法的步骤可以表示为：

$$\begin{array}{ll} \text{Policy Iteration: } & \pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots \\ \text{Value Iteration: } & v_0 \xrightarrow{PU} \pi'_1 \xrightarrow{VU} v_1 \xrightarrow{PU} \pi'_2 \xrightarrow{VU} v_2 \xrightarrow{PU} \dots \end{array}$$

可以看出，两种算法的流程非常相似。

	策略迭代算法	值迭代算法	注释
策略	$\pi_0$	$N/A$	
值	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 \doteq v_{\pi_0}$	
策略	$\pi_1 = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_0})$	$\pi_1 = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_0})$	两个策略一样
值	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$ since $v_{\pi_1} \geq v_{\pi_0}$
策略	$\pi_2 = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_1})$	$\pi'_2 = \operatorname{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_1)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$

表 4.6

我们更仔细地检查它们的值步长，以了解两种算法之间的差异。特别是，让两种算法从相同的初始条件开始： $v_0 = v_{\pi_0}$ 。两种算法的流程如表 4.6 所示。在前三个步骤中，由于  $v_0 = v_{\pi_0}$ ，这两种算法生成相同的结果。它们在第四步变得不同。在第四步中，值迭代算法执行  $v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$ ，这是一步计算，而策略迭代算法求解  $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ ，需要无限次迭代。如果我们明确地写出第四步求解  $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$  的迭代过程，一切就都清楚了。令  $v_{\pi_1}^{(0)} = v_0$ ，我们有

$$\begin{aligned} v_{\pi_1}^{(0)} &= v_0 \\ \text{value iteration} \leftarrow v_1 \leftarrow v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\ v_{\pi_1}^{(1)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\ &\vdots \\ \text{truncated policy iteration} \leftarrow \bar{v}_1 \leftarrow v_{\pi_1}^{(j)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} \\ &\vdots \\ \text{policy iteration} \leftarrow v_{\pi_1} \leftarrow v_{\pi_1}^{(\infty)} &= r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)} \end{aligned}$$

从上述过程中可以得到以下观察结果：

- 如果迭代仅运行一次，则  $v_{\pi_1}^{(1)}$  实际上是  $v_1$ ，如值迭代算法中计算的那样。
- 如果迭代运行无限次，则  $v_{\pi_1}^{\infty}$  实际上是  $v_{\pi_1}$ ，如策略迭代算法中计算的那样。
- 如果迭代运行有限次（表示为  $j_{\text{truncate}}$ ），则此类算法称为截断策略迭代。之所以称为截断，是因为从  $j_{\text{truncate}}$  到  $\infty$  的剩余迭代被截断。

因此，值迭代和策略迭代算法可以被视为截断策略迭代算法的两个极端情况：值迭代在  $j_{\text{truncate}} = 1$  处终止，策略迭代在  $j_{\text{truncate}} = \infty$  处终止。需要说明的是，上述比较虽然是示例性的，但其前提是  $v_{\pi_1}^{(0)} = v_0 = v_{\pi_0}$ 。如果没有这个条件，就无法直接比较两种算法。

### ❤ 算法 4.3：截断策略迭代算法

**初始化：**所有  $(s, a)$  的概率模型  $p(r|s, a)$  和  $p(s'|s, a)$  已知。初始猜测  $\pi_0$ 。

**目标：**寻找最优状态值和最优策略。

**While**  $v_k$  尚未收敛时，对于第  $k$  次迭代，**do**

策略评估：

初始化：选择初始猜测为  $v_k^{(0)} = v_{k-1}$ 。最大迭代次数设置为  $j_{\text{truncate}}$ 。

**While**  $j < j_{\text{truncate}}$  时，**do**

**For** 每个状态  $s \in \mathcal{S}$  **do**

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[ \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}^{(j)}(s') \right]$$

设置  $v_k = v_k^{(j_{\text{truncate}})}$

策略提升：

**For** 每个状态  $s \in \mathcal{S}$ ，**do**

**For** 每个动作  $a \in \mathcal{A}$ ，**do**

$$q_{\pi_k}(s, a) = \sum_r p(r|s, a)r + \gamma \sum_{s'} p(s'|s, a)v_{\pi_k}(s')$$

$$a_k^*(s) = \operatorname{argmax}_a q_{\pi_k}(s, a)$$

**If**  $a = a_k^*$ ，则  $\pi_{k+1}(a|s) = 1$ ，**Else**  $\pi_{k+1}(a|s) = 0$

## 4.3.2 截断策略迭代算法

简而言之，截断策略迭代算法与策略迭代算法相同，只是它仅在策略评估步骤中运行有限次数的迭代。算法 4.3 总结了其实现细节。值得注意的是，算法中的  $v_k$  和  $v_k^{(j)}$  不是状态值。相反，它们是真实状态值的近似值，因为在策略评估步骤中仅执行有限次数的迭代。

如果  $v_k$  不等于  $v_{\pi_k}$ ，算法还能找到最优策略吗？答案是肯定的。直观上，截断策略迭代介于值迭代和策略迭代之间。一方面，它比值迭代算法收敛得更快，因为它在策略评估步骤中计算多次迭代。另一方面，它的收敛速度比策略迭代算法慢，因为它只计算有限次数的迭代。这种直觉如图 4.5 所示。这种直觉也得到了以下分析的支持。

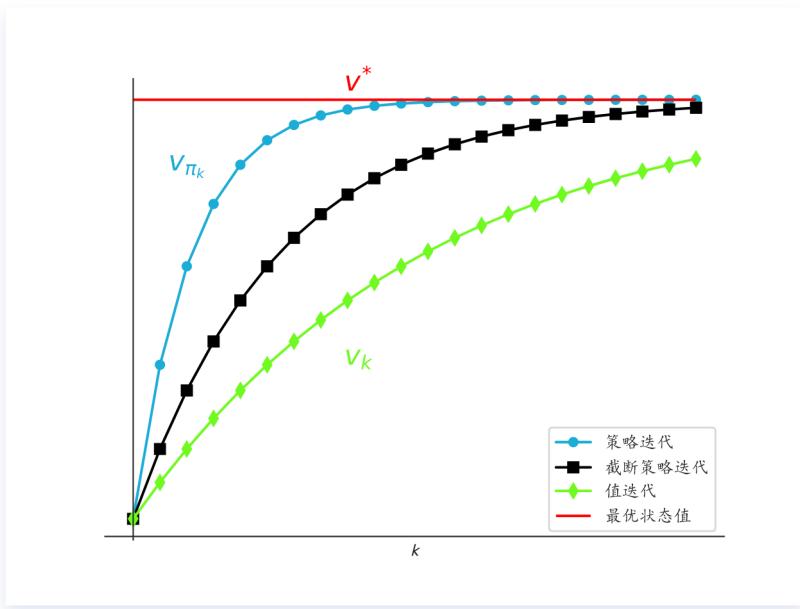


图 4.5

### 命题 4.1 价值改进

考虑策略评估步骤中的迭代算法：

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \dots$$

如果初始猜测选择为  $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$ , 则对于  $j = 0, 1, 2, \dots$  则成立：

$$v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$$

### □ 4.3 命题 4.1 的证明：

首先, 由于  $v_{\pi_k}^{(j)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j-1)}$  且  $v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}$ , 我们有

$$v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)} = \gamma P_{\pi_k} \left( v_{\pi_k}^{(j)} - v_{\pi_k}^{(j-1)} \right) = \dots = \gamma^j P_{\pi_k}^j \left( v_{\pi_k}^{(1)} - v_{\pi_k}^{(0)} \right). \quad (4.5)$$

其次, 由于  $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$ , 我们有：

$$v_{\pi_k}^{(1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(0)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_{k-1}} \geq r_{\pi_{k-1}} + \gamma P_{\pi_{k-1}} v_{\pi_{k-1}} = v_{\pi_{k-1}} = v_{\pi_k}^{(0)},$$

其中不等式是由于  $\pi_k = \text{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_{k-1}})$  引起的。将  $v_{\pi_k}^{(1)} \geq v_{\pi_k}^{(0)}$  代入 (4.5) 得到  $v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$ 。

值得注意的是, 命题 4.1 需要假设  $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$ 。但实际中  $v_{\pi_{k-1}}$  是不可用的, 只有  $v_{k-1}$  可用。尽管如此, 命题 4.1 仍然阐明了截断策略迭代算法的收敛性。

到目前为止, 截断策略迭代的优势是显而易见的。与策略迭代算法相比, 截断算法在策略评估步骤中只需要有限次数的迭代, 因此计算效率更高。与值迭代相比, 截断策略迭代算法可以通过在策略评估步骤中运行更多迭代来加快收敛速度。

1. R. E. Bellman, Dynamic programming. Princeton University Press, 2010. [←](#)
2. R. E. Bellman and S. E. Dreyfus, Applied dynamic programming. Princeton University Press, 2015. [←](#)
3. J. Bibby, “Axiomatisations of the average and a further generalisation of monotonic sequences,” Glasgow Mathematical Journal, vol. 15, no. 1, pp. 63–65, 1974. [←](#)