1. value iteration：值迭代

由上一章所述，$f(v)$为压缩映射，故其满足：

$$v_{k+1} = f(v_k) = \max_{\pi}(r_\pi + \gamma P_\pi v_k)$$
$$k \to \infty, v_k \to v^*$$

该算法即称为值迭代算法，求解步骤分为两步：

（1） policy update：策略更新，当v_k已知时，求π_{k+1}：

$$\pi_{k+1} = arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$

（2） value update：值更新，根据上一步求得的π_{k+1}求出v_{k+1}：

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

在这里v_k并不一定是一个 state value，因为其不一定满足贝尔曼公式$v_\pi = r_\pi + \gamma P_\pi v_\pi$，它可以只是一个普通的向量。

■ policy update：策略更新

$$\pi_{k+1}(s) = arg\max_{\pi}\sum_a \pi(a|s)\left(\sum_r p(r|s,a)r + \gamma\sum_{s'} p(s'|s,a)v_k(s')\right), s \in S$$

由于等式中的量均已知，对每个$s \in S$，我们都可以得到这样一个式子，即可对策略进行更新，更新后的策略为：

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

它是一个贪婪的策略

■ value update：值更新

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

写成元素形式为:

$$v_{k+1}=\sum_{a}\pi_{k+1}(a|s)\left(\sum_{r}p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_k(s')\right), s\in S$$

由于此时$\pi_{k+1}(a|s)$已知，代入后对每个状态 s 均可求出它的最新的 state value

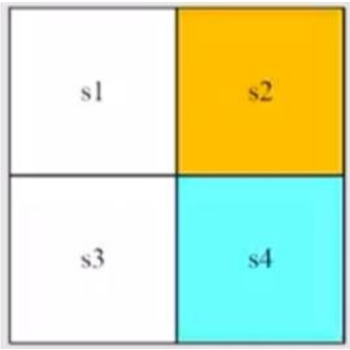
由于$\pi_{k+1}(a|s)$是贪婪的，故最后结果为：$v_{k+1}=\max_{a} q_k(a,s)$

其求解过程伪代码为:

For every state $s\in\mathcal{S}$, do
    For every action $a\in\mathcal{A}(s)$, do
        q-value: $q_k(s,a)=\sum_r p(r|s,a)r+\gamma\sum_{s'}p(s'|s,a)v_k(s')$
    Maximum action value: $a_k^*(s)=\arg\max_a q_k(a,s)$
    Policy update: $\pi_{k+1}(a|s)=1$ if $a=a_k^*$, and $\pi_{k+1}(a|s)=0$ otherwise
    Value update: $v_{k+1}(s)=\max_a q_k(a,s)$

例子:

有初始状态如下图，$r_{boundary}=r_{forbidden}=-1, r_{target}=1, \gamma=0.9$



则每个 state 的每个 action 的 action value 如下表所示:

q-value	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	$-1+\gamma v(s_2)$	$-1+\gamma v(s_4)$	$-1+\gamma v(s_4)$	$0+\gamma v(s_3)$	$1+\gamma v(s_4)$

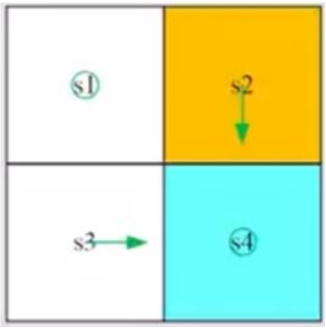
第一轮迭代，k=0 时：令$v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$，代入后可以得到如下所示的结果：

q-value	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

第一步，策略更新：选择各 state 的 action value 最大的 action 作为此次最优策略（贪心），得到结果为：

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

该策略如图所示：



第二步，值更新：根据当前策略，计算各 state 的 state value，得到结果为：

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

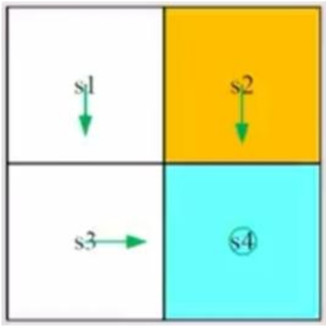
第二轮迭代，k=1：更新 q-value 表，得到结果为：

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	$-1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$	$1+\gamma 1$

第一步，策略更新：选择各 state 的 action value 最大的 action 作为此次最优策略（贪心），得到结果为：

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

该策略如图所示：



第二步，值更新：根据当前策略，计算各 state 的 state value，得到结果为：

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

求解完毕

2．policy iteration：策略迭代

与 value iteration 不同的是，value iteration 在初始时会给定一个随机的 state value 序列，而在 policy iteration 中，在初始时会给定一个随机的 policy π_0

求解步骤分为两步：

（1） policy evaluation(PE)：策略评估，根据已有的策略，计算出在当前策略下的 state value

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

（2） policy improvement(PI)：策略改进，根据求得的 state value，求得各个状态的 action value，并选取最大的作为该 state 的新的 policy（贪心）

$$\pi_{k+1} = arg \max_{\pi} \left(r_\pi + \gamma P_\pi v_{\pi_k} \right)$$

■ 在步骤一 policy evaluation 中，如何求解 state value？

见第二章，分两种方法：直接法和迭代法

■ 直接法：通过公式$v_\pi = r_\pi + \gamma P_\pi v_\pi$，得到$v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$

■ 迭代法（存疑）：$v_{k+1} = r_\pi + \gamma P_\pi v_k$

给定初始的v_0，得到$v_1, v_2, \ldots, v_k$，当$k \to \infty$时，$v_k \to v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$。通过这种方式我们可以得到一个 state value 序列$\{v_0, v_1, v_2, \ldots\}$

■ 在步骤二 policy improvement 中，为什么得到的π_{k+1}一定优于π_k？

直接给出结论：若$\pi_{k+1} = arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_k}\right)$，则$v_{\pi_{k+1}} \geq v_{\pi_k}$

■ 为什么这样的迭代算法可以最终得到最优策略？

在经过多轮的迭代后，根据上述问题的结论，则有：

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

由此可知，v_{π_k}会持续增长，并当$k \to \infty$时，收敛于v^*

- policy evaluation：采用迭代法进行求解：

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

当$j \to \infty$或$\left\| v_{\pi_{k+1}}^{(j+1)} - v_{\pi_k}^{(j)} \right\|$足够小时，求解完成，得到$v_{\pi_{k+1}}^{(j+1)}$

- policy improvement：根据上一步求得的$v_{\pi_k}^{(j+1)}$，求出各个 state 的 action value，选择 action value 最大的值作为当前策略（贪心）：

$$\pi_{k+1} = arg\max_\pi \left(r_\pi + \gamma P_\pi v_{\pi_k}\right)$$

$$\pi_{k+1}(s) = arg\max_\pi \sum_a \pi(a|s) \left(\sum_r p(r|s,a) r + \gamma \sum_{s'} p(s'|s,a) v_k(s') \right), s \in S$$

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

$$a_k^*(s) = arg\max_a q_k(a,s)$$

策略迭代算法求解的伪代码为:

Policy evaluation:

Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$

While $v_{\pi_k}^{(j)}$ has not converged, for the jth iteration, do

  For every state $s \in \mathcal{S}$, do

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

Policy improvement:

For every state $s \in \mathcal{S}$, do

  For every action $a \in \mathcal{A}(s)$, do

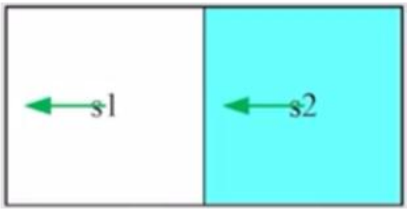
  $$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma\sum_{s'} p(s'|s,a)v_{\pi_k}(s')$$

  $a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$

  $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

例子:

有初始策略如下图，$r_{boundary} =-1, r_{target} = 1, \gamma = 0.9$，这里只有三个工作，$a_l$表示向左走，$a_r$表示向右走，$a_0$表示原地不动



第一轮迭代，k=0 时:

第一步，policy evaluation：根据初始条件计算出当前各个状态的 state value，结果如下所示:

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

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

迭代法求解 state value，假设$v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0$，则有以下结果:

$$\begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases}$$

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

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

$$\ldots$$

最后逼近我们采用直接法求得的结果：$v_{\pi_0}(s_1) =- 10, v_{\pi_0}(s_2) =- 9$

第二步，policy improvement：根据求得的 state value，计算出每个 state 的 action value，贪心地对策略进行更新：

$q_{\pi_k}(s,a)$	a_ℓ	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	$0+\gamma v_{\pi_k}(s_1)$	$1+\gamma v_{\pi_k}(s_2)$	$-1+\gamma v_{\pi_k}(s_2)$

将$v_{\pi_0}^{(0)}(s_1) =- 10, v_{\pi_0}^{(0)}(s_2) =- 9$ 代入后，得到以下结果：

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

根据结果对策略进行更新，得到：

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

此时策略已达到最优，求解完毕

3. truncated policy iteration：截断策略迭代

首先让我们回顾一下前两种迭代算法：policy iteration 和 value iteration

Policy iteration: start from π_0

- Policy evaluation (PE):

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

- Policy improvement (PI):

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})$$

Value iteration: start from v_0

- Policy update (PU):

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)$$

- Value update (VU):

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

接下来对这两种算法的过程进行比较：

$$\text{Policy iteration: } \pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \ldots$$

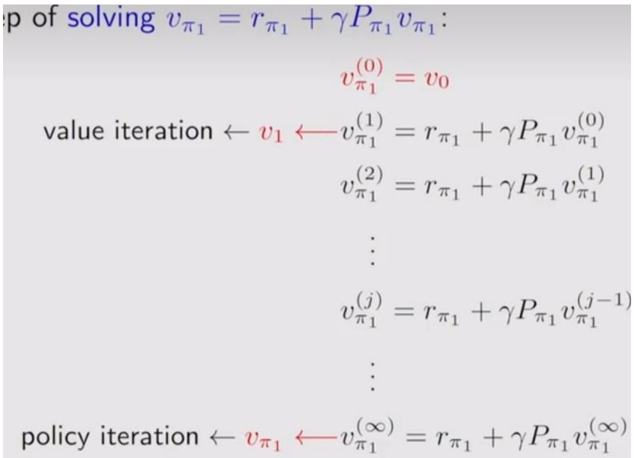
$$\text{Value iteration: } u_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} u_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} u_2 \xrightarrow{PU} \ldots$$

可以看到，两种策略的过程形式是比较类似的，接下来对这两种算法进行整体比较：

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 := v_{\pi_0}$	
3) Policy:	$\pi_1 = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_0})$	$\pi_1 = \arg\max_\pi(r_\pi + \gamma P_\pi v_0)$	The two policies are the same
4) Value:	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$ since $v_{\pi_1} \geq v_{\pi_0}$
5) Policy:	$\pi_2 = \arg\max_\pi(r_\pi + \gamma P_\pi v_{\pi_1})$	$\pi_2' = \arg\max_\pi(r_\pi + \gamma P_\pi v_1)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$

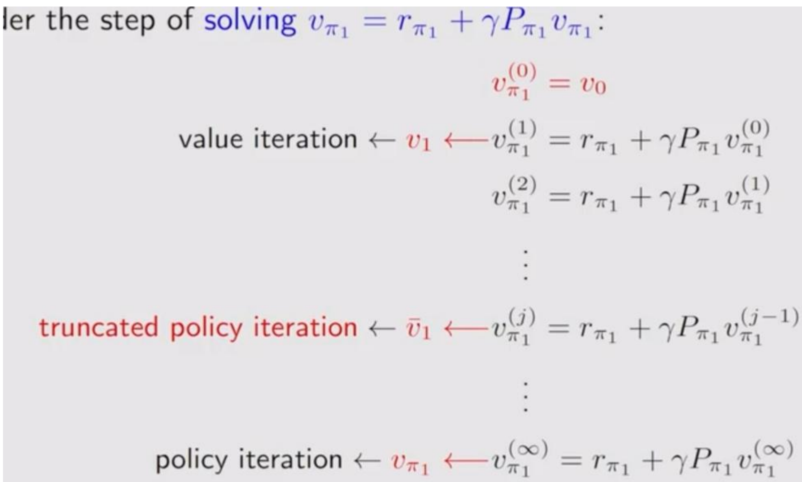
在 value iteration 中，初始状态下是没有策略的，且我们可以任意指定一个 state value 序列，这里为了方便比较，我们将 policy iteration 中根据初始策略计算出来

的 state value 赋值给v_0。可以看到，在第三步的策略更新中，两个算法是一致的，但是在第四步根据当前新策略计算 state value 步骤中，两种算法有所不同，具体如下所示：



在 policy iteration 中，我们采用迭代法来求解新的 state value；而在 value iteration 中，我们直接用指定的v_0和更新的策略求解出v_1，实际上存在关系$v_{\pi_1}^{(1)} = v_0$；而随着迭代的进行，$v_{\pi_1}^{(j)}$一定会持续增长，所以在该步骤中 policy iteration 求得的值v_{π_1}一定大于 value iteration 求得的值v_1。

但是在实际应用中，policy iteration 是不可实现的，因为我们无法做到迭代无穷次。truncated policy iteration（截断策略迭代）则是在这个过程中进行了指定次数$j_{truncated}$的迭代，它是介于 value iteration 和 policy iteration 之间的迭代算法，如下图所示：



该算法的伪代码如下：

Policy evaluation:

Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum iteration is set to be j_{truncate}.

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

  For every state $s \in \mathcal{S}$, do

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

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

Policy improvement:

For every state $s \in \mathcal{S}$, do

  For every action $a \in \mathcal{A}(s)$, do

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

  $a_k^*(s) = \arg\max_a q_k(s,a)$

  $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

对三种迭代算法进行比较，让它们均从相同的 state value 开始，得到结果如下：