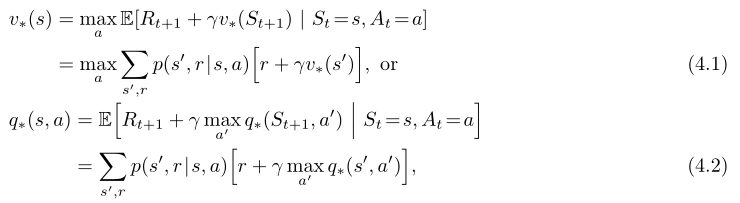
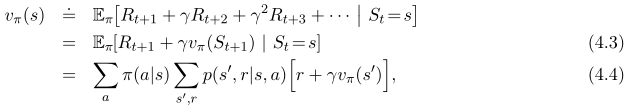
# 第四章 动态规划

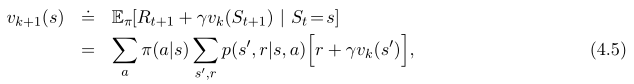
**引言：**当完全了解一个具有MDP属性的模型时，DP用于求解最优策略。实际上DP受制于两个方面：对环境的完全了解、巨大的算力。所以DP在RL应用不多，但是理论仍旧重要。

DP甚至RL的核心思想就是用value function来搜索好的policy。在本章，我们回去学习如何用DP来计算value function。一旦我们有了V\*或者Q\*，那么就很容易获得最优的策略。DP可以理解为：给bellman等式添加更新机制。

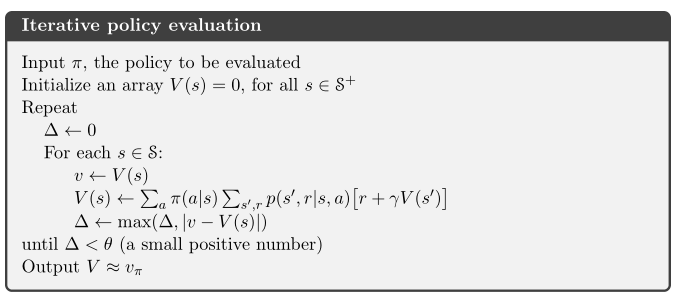
****

## 4.1 Policy Evaluation-求value function



第一项可以看做行为层，第二层为基于action求value值。结合V\*可以看出，V\*就是上式action层的最大值。直接解上面的方程会比较难，可以迭代求取。

上面的公式就是迭代的公式。完整的算法：



## 4.2 Policy Improvement

假如我们的state-value function迭代完毕，那么我么需要怎么改进之前的策略呢?

4.2.1 策略提升定理

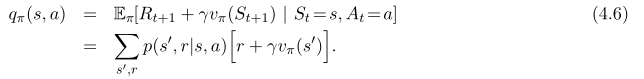
假如对所有的状态满足：

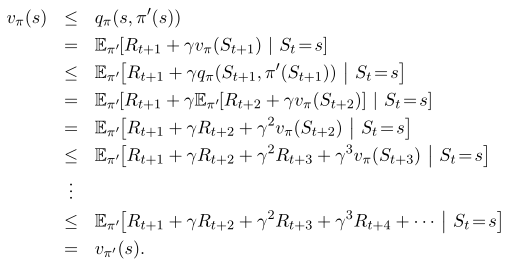


那么就可以得到：



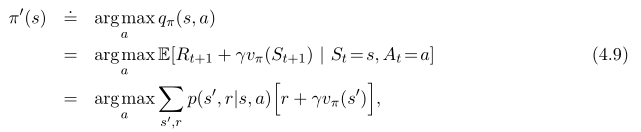
基于下式，可以证明上定理：





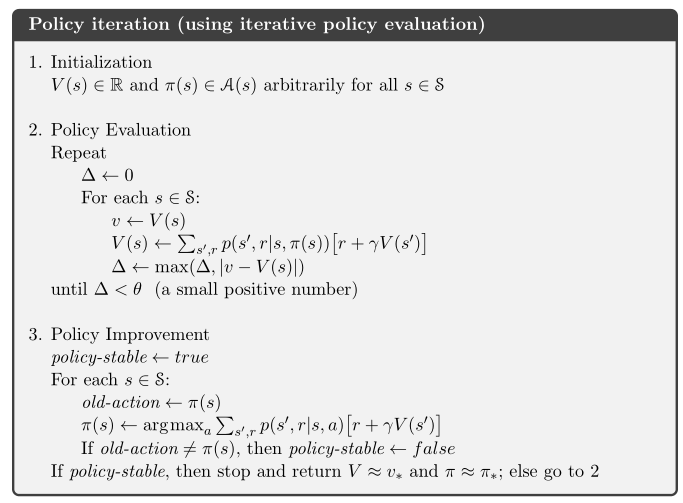
注：期望用π'注释表示action使用π'选择，实际上就是局部使用了新的policy。

很自然的，我们或扩展到所有的状态，并且获得一个贪心的策略：



贪心提升策略虽然只是向前看了一步，但是可以收敛到全局最优。

## 4.3 Policy Iteration



## 4.4 Value Iteration-值迭代算法

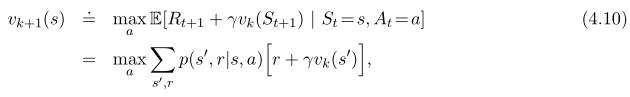
首先小结下前面列出的算法：

policy evaluation：对state value function 迭代收敛

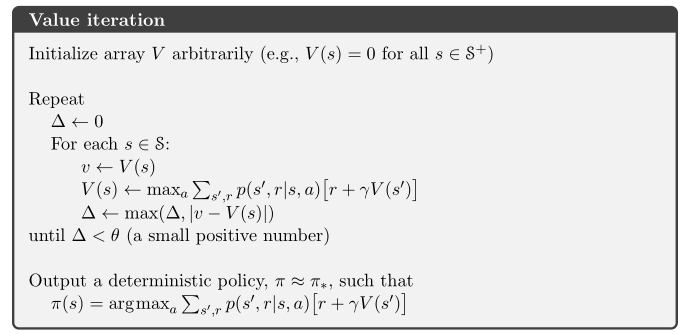
policy improvement：基于evaluation的结果对policy优化

policy Iteration：将前面两个算法构成闭环。

但是，假如我们需要的只是一个policy的话，policy evaluation是否有必要迭代到收敛呢？显然不需要，我们实际上只需要policy收敛就行了。基于这个想法，有以下的value iteration计算公式：



完整算法如下：可以理解为一步evaluation直接策略提升。



policy evaluation与value iteration的区别：

policy evaluation：用期望值更新当前值

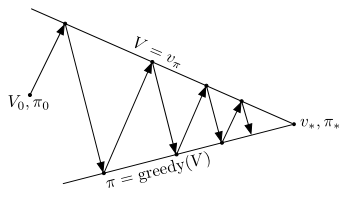
value iteration：用最大的期望值更新当前值(最大则内涵了improvement)

补充：从前面的网格世界我们可以看出：策略迭代到第三步就可以stable。而value 需要迭代很多次才会stable，我们自然会想到一种优化的方法：evaluation若干步就去policy improvement，一旦policy stable 就退出策略提升的过程。(加速)

## 4.5 异步动规

更加灵活的更新方式：原地操作、改变顺序等等

## 4.6 Generalized Policy Iteration-通用策略迭代



小结：一般的RL过程中都会涉及value和policy，我们的目标当然是追求最优+稳定。本章提出DP的方法来实现这两个要求。概括的讲就是:GPI。

首先值迭代到稳定(上箭头)、然后策略提升(下箭头)直到最优。当然，向上的箭头可能没有达到收敛就直接policy improvment，例如 value iteration。