




强化学习

Python编程实验二

动态规划求解小型方格世界最优策略





■ 实验理论回顾

- 策略评估 (Policy Evaluation)
- 策略迭代 (Policy Iteration)
- 价值迭代 (Value Iteration)



■ 动态规划 (DP)

规划：在已知环境动力学的基础上进行**评估**和**控制**。

动态规划算法：把求解**复杂**问题分解为求解**子问题**，通过求解子问题进而得到整个问题的解。在解决子问题的时候，其结果通常需要存储起来被用来解决后续复杂问题。

动态规划求解问题的性质：

- 1、一个**复杂**问题的最优解由数个**小问题**的**最优解**构成，可以通过寻找子问题的最优解来得到复杂问题的最优解。
- 2、子问题在复杂问题内重复出现，使得子问题的解可以被存储起来**重复利用**。



■ 预测和控制

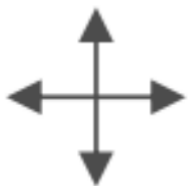
预测和**控制**是规划的两个重要内容。预测是对给定策略的评估过程，控制是寻找一个最优策略的过程。对预测和控制的数学描述是这样：

预测 (prediction): 已知一个马尔科夫决策过程 $\text{MDP} \langle S, A, P, R, \gamma \rangle$ 和一个策略 π ，或者是给定一个马尔科夫奖励过程 $\text{MRP} \langle S, P\pi, R\pi, \gamma \rangle$ ，求解基于该策略的**价值函数** v_π 。

控制 (control): 已知一个马尔科夫决策过程 $\text{MDP} \langle S, A, P, R, \gamma \rangle$ ，求解**最优价值函数** v^* 和 **最优策略** π^* 。



■ 实验示例



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

4 × 4 小型方格世界

要求：通过移动自己来到达两个灰色格子中的任意一个来完成任务

说明：状态0、状态15为终止状态 $R = 0$ ； $\gamma = 1$ ；其余状态奖励 $R = -1$



■ 策略评估(Policy Evaluation)

策略评估 (policy evaluation) 指计算给定策略下状态价值函数的过程。

从任意一个状态价值函数开始，依据给定的策略，结合贝尔曼期望方程、状态转移概率和奖励同步迭代更新状态价值函数，直至其收敛，得到该策略下最终的状态价值函数。

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$



价值函数

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$$-1.7 \approx (-1 + 1 \times (\frac{1}{4} \times 0 + \frac{1}{4} \times (-1) + \frac{1}{4} \times (-1) + \frac{1}{4} \times (-1)))$$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0

$$-2.4 \approx (-1 + 1 \times (\frac{1}{4} \times 0 + \frac{1}{4} \times (-2) + \frac{1}{4} \times (-1.7) + \frac{1}{4} \times (-2)))$$

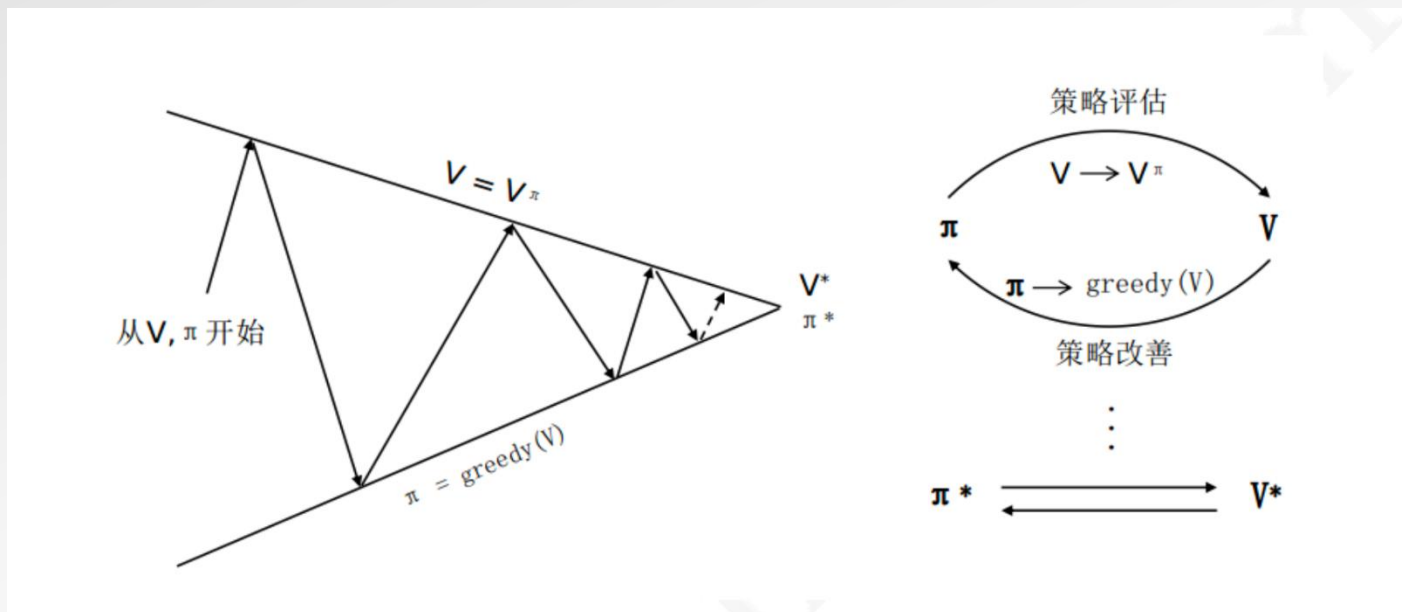
小型方格世界迭代中的价值函数



■ 策略迭代(Policy Iteration)

当给定一个策略 π 时，可以得到基于该策略的价值函数 v_π ，基于产生的价值函数可以得到一个贪婪策略 $\pi' = \text{greedy}(v_\pi)$ 。

依据新的策略 π' 会得到一个新的价值函数，并产生新的贪婪策略，如此重复循环迭代将最终得到最优价值函数 v^* 和最优策略 π^* 。策略在循环迭代中得到更新改善的过程称为策略迭代 (policy iteration)。



策略迭代过程示意图

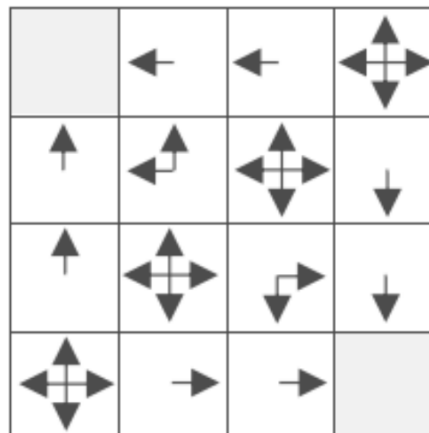


■ 策略迭代(Policy Iteration)

完成对一个策略的评估，将得到基于该策略下每一个状态的价值。

贪婪策略：个体在某个状态下选择的行为是其能够到达后续所有可能的状态中**价值最大**的那个状态。

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



小型方格世界策略的改善 (k=2)



■ 价值迭代 (Value Iteration)

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

(e) $k=10$

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0

	←	←	↙
↑	↖	↖	↓
↑	↖	↘	↓
↙	→	→	

小型方格世界 $k = \infty$ 时的贪婪策略

问题:

是否可以提前设置一个迭代终点来减少迭代次数而不影响得到最优策略呢?

是否可以每迭代一次就进行一次策略评估呢?



■ 价值迭代 (Value Iteration)

任何一个**最优策略**可以分为两个阶段，首先该策略要能产生当前状态下的**最优行为**，其次对于该最优行为到达的后续状态时该策略仍然是一个**最优策略**。

最优化原则：一个策略能够获得**某状态 s** 的最优价值当且仅当该策略也同时获得状态 s 所有可能的**后续状态 s'** 的最优价值。

一个状态的**最优价值**可以由其后续状态的最优价值通过前一章所述的贝尔曼最优方程来计算：

$$v_*(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right)$$



■ 价值迭代 (Value Iteration)

0			

V_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

小型方格世界最短路径问题



■ Python代码实现

■ 小型方格世界 MDP 建模

■ 策略评估

■ 策略迭代

■ 价值迭代