马尔可夫决策过程 MDP

马尔可夫决策过程是强化学习最基本的数学工具，如果需要用强化学习去解决一个实际问题，第一步就是将要做的事情把实际问题抽象为一个马尔可夫决策过程，也就是将强化学习解决实际问题的过程要素抽象为马尔可夫决策过程的各个组成要素。

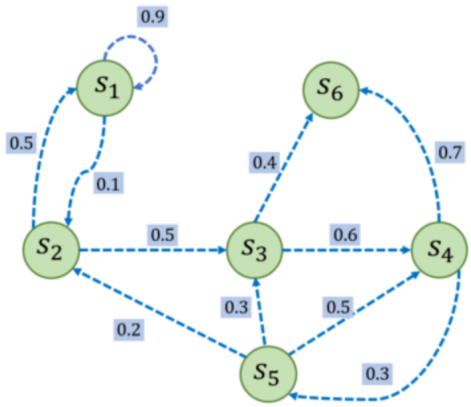
在理解马尔可夫决策过程前，先从简单的马尔可夫过程开始理解，然后加入奖励要素理解马尔可夫奖励过程，在马尔可夫奖励过程的此基础上加入动作要素就是我们需要学习的**马尔可夫决策过程**

马尔可夫过程：

马尔可夫过程指具有马尔可夫性质的随机过程，也被称为**马尔可夫链**

马尔可夫过（MP）程可以用一个二元组来表示<**S**，**P**>，**S**为这个过程种存在的所有状态集合，它是一个有限数量状态的集合；**P**是状态转移的概率的矩阵，矩阵的元素$P_{i,j}$表示着一个状态 S_i 转移到 S_j 的概率

下面两张图就很鲜活的概括**S**和**P**：



上图为具有六个状态的马尔可夫过程，那么对应该过程转移由蓝色标注的状态转换概率，我们可得**P**状态转移矩阵

$$\mathcal{P}=\begin{bmatrix}0.9 & 0.1 & 0 & 0 & 0 & 0\\ 0.5 & 0 & 0.5 & 0 & 0 & 0\\ 0 & 0 & 0 & 0.6 & 0 & 0.4\\ 0 & 0 & 0 & 0 & 0.3 & 0.7\\ 0 & 0.2 & 0.3 & 0.5 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$$

给定一个马尔可夫过程，我们就可以从某个状态出发，根据它的状态转移矩阵生成一个状态**序列**（episode），这个步骤也被叫做**采样**（sampling）。

马尔可夫性质：

我们在状态转移时，假设初始时为s_1，t = 1，即在t = 1时刻时状态为s_1，t每增加1就会增加一个状态

假设在t = 4时，有$s_1 \to s_2 \to s_3 \to s_6$如上图所示

在t = 4这个时刻，状态为s_6，而转移到s_6这个状态的的概率只取决于它上一个时刻的状态s_3，即概率为$P(s_6|s_3)$，而不会受到它过去状态转移的影响，用公式表示为$\mathbf{P(s_6|s_3) = P(s_6|s_1 \to s_2 \to s_3)}$

即当前仅当某时刻的状态只取决于上一时刻的状态时，该随机过程被称为具有马尔可夫性质。

因为虽然t时刻的状态只与t-1时刻状态有关，但t-1时刻状态其实已经包含了t-2时刻状态的信息，通过这种链式的关系，历史的信息被传递到现在。所以**马尔可夫性可以大大简化运算，因为只要当前状态可知，所有的历史信息都不再需要了，利用当前状态信息就可以决定未来。**

马尔可夫奖励过程：

在马尔可夫过程的基础上加入奖励函数 *r* 和折扣因子 γ，就可以得到**马尔可夫奖励过程**（MRP）。一个马尔可夫奖励过程可以用四元组概括<**S**，**P**，**r**，**γ**>，各个组成元素的含义如下所示。

- S有限状态集合
- P状态转移矩阵
- r奖励函数，在某个状态s下的奖励r(s)，表示转移到该状态下可获得的奖励的期望
- γ折扣因子，取值范围[0，1），在累积获得奖励时，我们希望尽可能快的获得奖励，引入折扣因子在计算累积奖励期望时对长时间获得的奖励打一些折扣

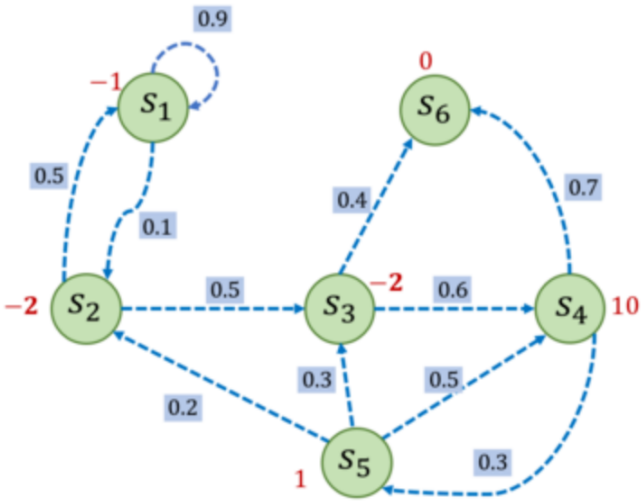
接近 1 的γ更关注长期的累计奖励，接近 0 的γ更考虑短期奖励。

累积回报：

在一个马尔可夫奖励过程中，从第t时刻状态$\mathbf{S_t}$开始，直到终止状态时，所有奖励的衰减之和称为**回报** G_t（Return），公式如下：

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

将刚刚那个只有6个状态的马尔可夫过程修改以下，在每个状态加上对应的奖励值（状态旁红色的数值）



以刚刚的$s_1 \to s_2 \to s_3 \to s_6$为例，设立衰减因子（折扣因子γ = 1/2）为0.5，从t = 1时刻开始，初始状态为s_1，即可通过回报公式计算s_1的回报G_1

这就是我们所关心的智能体从初始状态出发到"游戏"结束时所获得的奖励的累积回报G_1，奖励的累积回报从计算的公式我们可以看出它就是状态在不断变化为新的状态的奖励序列之和，只不过奖励前乘了折扣因子，这是因为我们希望这个累积回报值越大越好的同时这个值变大的越早越好（即奖励越早拿到越好），为了保证奖励越早拿越好，所以才增加了衰减因子，衰减因子因为是小于1的数，所以会随着时间的推移会越来越小，在公式种可得长期得到的奖励占比会越来越少

代入回报公式得：

$$G_1 = -1 + 0.5 \times (-2) + (0.5)^2 \times (-2) + (0.5)^3 \times 0 = -2.5$$

用代码实现，回报计算过程。

虽然他没有像李沐老师那样单独去讲解代码的实现，但是他的注释实在是太简介了，我哭死🎃

```python
import numpy as np
np.random.seed(0)
# 定义状态转移概率矩阵P
P = [
    [0.9, 0.1, 0.0, 0.0, 0.0, 0.0],
    [0.5, 0.0, 0.5, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.6, 0.0, 0.4],
    [0.0, 0.0, 0.0, 0.0, 0.3, 0.7],
    [0.0, 0.2, 0.3, 0.5, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 1.0],
]
P = np.array(P)

rewards = [-1, -2, -2, 10, 1, 0]  # 定义奖励函数
gamma = 0.5  # 定义折扣因子


# 给定一条序列,计算从某个索引（起始状态）开始到序列最后（终止状态）得到的回报
def compute_return(start_index, chain, gamma):
    G = 0
    for i in reversed(range(start_index, len(chain))):
        G = gamma * G + rewards[chain[i] - 1]
    return G


# 一个状态序列,s1-s2-s3-s6
chain = [1, 2, 3, 6]
start_index = 0
G = compute_return(start_index, chain, gamma)
print("根据本序列计算得到回报为：%s。" % G)

根据本序列计算得到回报为：-2.5。
```

价值函数：

基于状态的累积回报（简称回报）在上一节做了定义了，每个状态因为变换的过程不同，所以累积的回报也是不同的。这意味着一个状态有很多很多种累积回报，那么如何衡量这个状态s的**价值**呢？一般来说，一个状态s的累积回报越大，说明这个状态s的价值越高。但是正如前文所说的，状态s的累积回报是多种多样的，不能单看一种累积回报的大小来定义这个状态s的价值，所以还有一个方法——

对状态s的累积回报求均值，即状态s累积回报的期望作为该状态的价值，具体计算方法就是观察很多很多从状态s开始的变化序列，然后计算这些轨迹的累积回报是多少，计算之后求平均值，即累积回报的期望作为状态s的价值

上图所示，这个马尔可夫过程有6个状态，我们就需要去衡量每个状态的期望累积回报值，即每个状态的奖励

所以我们对一个过程的状态价值可以定义为一个**价值函数**，价值函数的输入为某个状态，输出为这个状态的价值。

$$V(\mathbf{s}) = E[G_t|\mathbf{S}_t = \mathbf{s}]$$

对该公式展开化简可得：

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t|S_t = s] \\
&= \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s] \\
&= \mathbb{E}[R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \ldots) | S_t = s] \\
&= \mathbb{E}[R_t + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_t + \gamma V(S_{t+1}) | S_t = s]
\end{aligned}
$$

在上式最后的等式可得，状态s的价值为当前状态下的奖励加衰减因子乘以下个状态价值的期望，这也就是说当该状态没有状态变化时，即时奖励的期望正是奖励函数的输出

$$V(\mathbf{s}) = E[R_t|\mathbf{S}_t = \mathbf{s}]$$

另一方面，等式中的剩余部分**E[γV(S_{t+1}) | S_t = s]**这个期望可以根据以s为初始状态出发的转移概率得到

$$V(s) = r(s) + \gamma \sum_{s' \in S} p(s'|s)V(s')$$

❓❓❓❓这里不是很理解为什么E[γV(S_{t+1}) | S_t = s]这部分对应初始状态s后续所转移的状态的转移概率与其价值乘积的和，应该是初始状态s之后转移的每个状态独立分布，所以可以将期望展开这样写，应该是这样的。概率论有点忘了😹 （哈哈哈，我这话说的好拗口啊

总之，上面的公式是马尔可夫奖励过程中非常有名的**贝尔曼方程**（Bellman equation），对每一个状态都成立。

所以对于每个状态s的奖励函数r(s)和状态转移矩阵P，根据贝尔曼方程通过解方程的方法，就能得到解析解，即个状态价值函数向量

$$
\begin{aligned}
\mathcal{V} &= \mathcal{R} + \gamma \mathcal{P}\mathcal{V} \\
(I - \gamma\mathcal{P})\mathcal{V} &= \mathcal{R} \\
\mathcal{V} &= (I - \gamma\mathcal{P})^{-1}\mathcal{R}
\end{aligned}
$$

如果用解析解求价值函数，时间复杂度为O(n^3)，很明显效率很低，并且随着矩阵的规模变大这不是一个好的方法，只适合规模很小的马尔可夫奖励过程

所以在求解大规模的马尔可夫奖励过程中的价值函数时，可以使用**动态规划算法、蒙特卡洛算法、时序差分算法**，这些算法以后会介绍

那么现在动手实现一下，利用贝尔曼方程的解析解来计算所有状态的价值，这里状态的所有价值是一个列向量，向量的每个元素对应着每个状态的价值

```
def compute(P, rewards, gamma, states_num):
    ''' 利用贝尔曼方程的矩阵形式计算解析解,states_num是MRP的状态数 '''
    rewards = np.array(rewards).reshape((-1, 1))  # 将rewards写成列向量形式
    value = np.dot(np.linalg.inv(np.eye(states_num, states_num) - gamma *
P),
                   rewards) # 代入解析解的公式,计算价值向量
    return value

#测试计算上图马尔可夫过程中的每个状态价值
V = compute(P, rewards, gamma, 6)
#状态转移概率矩阵P,奖励函数rewards(这里奖励是具体的值可以理解为常函数),折扣因子γ,状态数
print("MRP中每个状态价值分别为\n", V)

```

```
MRP中每个状态价值分别为
 [[-2.01950168]
 [-2.21451846]
 [ 1.16142785]
 [10.53809283]
 [ 3.58728554]
 [ 0.        ]]
```

好啦，理解了马尔可夫的奖励过程，终于迎来了本章的主题**马尔可夫决策过程**

马尔可夫决策过程：

一只小船随波逐流飘荡，可以理解为一个马尔可夫的奖励过程，凭借运气达到目的地意味着获得大量的奖励。但是如果船上有一名水手控制着小船往哪个方向前进，就可以主动的选择去往目的地。

所以在马尔可夫奖励的过程的基础上又外界的“刺激”来改变随机过程，就有了**马尔可夫决策过程**

我们将这个来自外界的刺激称为智能体的动作，在马尔可夫奖励过程的基础上加入动作，就得到了**马尔可夫决策过程（MDP）**。马尔可夫决策过程由<**S**，**A**，**P**，**r**，**γ**>元组构成，其中：

- S状态集合
- A是动作的集合
- P(s' | s , a)状态转移函数，表示在状态s执行动作a之后到达状态s'的概率。
- r(s , a)奖励函数，此时的奖励可以同时取决于状态s和动作a，在奖励函数只取决于状态s时退化成r(s)
- γ折扣因子

与MRP的区别：

MDP（马尔可夫决策过程）与MRP（马尔可夫奖励过程）很像，但还是有些区别

- 多了动作的集合，我们可以进行决策，使智能体决定做怎样的动作发展到我们想要智能体所处的状态
- 状态集合可以不再是有限的，在连续状态的MDP环境，状态集合是无法列举的，现在学习是关注的比较简单的离散的MDP环境，所以此时是有限的
- MDP的状态转移函数更为复杂，它是一个概率模型函数，而不单单是一个状态转移矩阵。这是因为状态转移函数更具有一般意义，当状态不再是有限的时候就无法再用矩阵数组进行表示
- 状态转移函数和奖励函数相比MRP多了作为自变量的动作a

策略：

马尔可夫决策过程相比MRP多了动作，那么智能体在一个状态下如何选择动作呢，这就是**策略**

为了将策略形式化，我们将策略记为一个函数，该函数的作用就是为智能体选择动作，通常用**π**表示

它的输出为输入状态为s的情况下，采取动作a的概率

$$\pi(\mathbf{a}|\mathbf{s}) = P(\mathbf{A}_t = \mathbf{a}|\mathbf{S}_t = \mathbf{s})$$

策略分为两种情况：

- 确定性策略：在该状态下只输出一个确定的动作，即只有该动作概率为1，其他动作的概率为0
- 随机性策略：每个状态的输出是关于动作的概率分布，根据分布采样就可以得到一个动作

有马尔可夫性质可知，策略只需要与当前的状态有关，不需要考虑历史状态，历史的信息被传递到了现在。

价值函数：

在MRP中定义过价值函数，MDP同样可以定义类似的价值函数，但此时的价值函数与策略有关。

这意味着对两个不同的策略在同一个状态下所取得的价值是不同的，所以由于动作的存在，可以除状态价值函数外额外定义一个动作价值函数

状态价值函数：

$$V^{\pi}(\mathbf{s}) = E_{\pi}[G_t|\mathbf{S}_t = \mathbf{s}]$$

表示从状态s出发遵循策略π能获得的期望回报

动作价值函数：

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = E_{\pi}[G_t|\mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}]$$

遵循策略π时，对当前状态s执行动作a得到的期望回报

二者关系：

在使用策略π时，状态s的价值等于在该状态下基于策略π采取所有动作的概率与相应价值相乘在求和的结果

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) Q^{\pi}(s, a)$$

使用策略π时，状态s下采取动作a的价值等于即时奖励加上经过衰减后的所有可能的下一个状态的状态转移概率与相应的价值的乘积：

$$Q^{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi}(s')$$

这里虽然没有推导，但还是很好理解的。在策略下的状态的价值函数肯定与动作有关，动作的价值函数也肯定变化后的状态价值函数有关，这两个函数的价值都是关于状态s和动作a的累积回报的期望

那么如何求得价值函数呢？在MRP那一节提到了将价值函数中的累积回报公式展开化简得**贝尔曼方法**，然后解方程得解析解的方式来求得价值函数

同样的我们也可以对MDP的状态价值函数和动作价值函数进行展开化简，得到类似的方程 😺

推导过程我可能也不会，直接上结果吧：

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[R_t + \gamma V^{\pi}(S_{t+1})|S_t = s] \\ &= \sum_{a \in A} \pi(a|s) \left(r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^{\pi}(s') \right) \end{aligned}$$

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}_{\pi}[R_t + \gamma Q^{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\ &= r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s', a') \end{aligned}$$

这样就得到了两个价值函数的**贝尔曼期望方程**

形式就是这个形式，模样也是抄的书上的

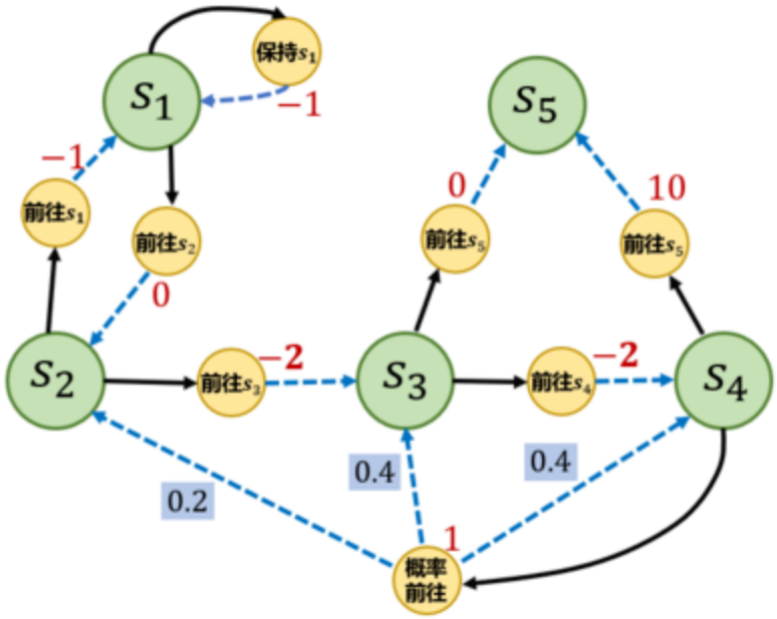
刚开始接触新的知识，因为什么都不懂，感觉什么都重要，有很多篇幅都是直接从书上进行摘录了，尤其是这里的价值函数和贝尔曼期望方程....

因为书上都明确的阐述了`价值函数和贝尔曼方程是强化学习非常重要的组成部分，之后的一些强化学习算法都是据此推导出来的，读者需要明确掌握！`所以这一块是很重要的

MDP实现：

上面阐述了一大堆，如果动手实现一下马尔可夫决策过程的一个例子应该通透不少。

还是以MRP那个图为例，只是在此基础上增加了动作，如下所示：



每个状态下的黄色圆形对应着某个状态下的动作，红色的数字表示某个状态执行动作可得到的奖励，而蓝色矩形表示某个状态执行某个动作的概率，没有蓝色矩形的动作默认在该状态下执行该动作的概率为1

所以根据这个过程，我们自然希望是采取什么样的策略能够得到最大的奖励，不过现在不是真正的强化学习，而是通过实现这个简单的例子帮助理解计算价值函数。

所以在对马尔可夫决策过程进行解决问题时，我们需要通过这张图抽象出MDP的各个组成要素，然后数字化进行编程。当然书上已经做好这个工作了，只需结合代码对照入座就是了🐷

要素抽象：

```
S = ["s1", "s2", "s3", "s4", "s5"]  # 状态集合
A = ["保持s1", "前往s1", "前往s2", "前往s3", "前往s4", "前往s5", "概率前往"]  # 动作集合

# 状态转移函数,说是函数,但其实并不是函数...因为状态比较少,同时也是离散的,直接进行了枚举
P = {
    "s1-保持s1-s1": 1.0, # 确定性动作
    "s1-前往s2-s2": 1.0,
    "s2-前往s1-s1": 1.0,
    "s2-前往s3-s3": 1.0,
    "s3-前往s4-s4": 1.0,
    "s3-前往s5-s5": 1.0,
    "s4-前往s5-s5": 1.0,
    "s4-概率前往-s2": 0.2, # 概率动作
    "s4-概率前往-s3": 0.4,
```

```
    "s4-概率前往-s4": 0.4,
}

# 奖励函数,直接将上图的每个动作的奖励枚举出来
R = {
    "s1-保持s1": -1,
    "s1-前往s2": 0,
    "s2-前往s1": -1,
    "s2-前往s3": -2,
    "s3-前往s4": -2,
    "s3-前往s5": 0,
    "s4-前往s5": 10,
    "s4-概率前往": 1,
}
gamma = 0.5  # 折扣因子
MDP = (S, A, P, R, gamma) # MDP五元组

# 策略1,随机策略
Pi_1 = {
    "s1-保持s1": 0.5,
    "s1-前往s2": 0.5,
    "s2-前往s1": 0.5,
    "s2-前往s3": 0.5,
    "s3-前往s4": 0.5,
    "s3-前往s5": 0.5,
    "s4-前往s5": 0.5,
    "s4-概率前往": 0.5,
}
# 策略2
Pi_2 = {
    "s1-保持s1": 0.6,
    "s1-前往s2": 0.4,
    "s2-前往s1": 0.3,
    "s2-前往s3": 0.7,
    "s3-前往s4": 0.5,
    "s3-前往s5": 0.5,
    "s4-前往s5": 0.1,
    "s4-概率前往": 0.9,
}


# 把输入的两个字符串通过“-”连接,便于使用上述定义的P、R变量
def join(str1, str2):
    return str1 + '-' + str2
```

边缘化：

以上就是将一个MDP的各个要素抽象的全过程，状态都是有限的，并且奖励函数和状态转移函数都是离散的，都是常函数，从图上直接看出来的。实际问题肯定会复杂很多，一般是根据状态的函数

然后我们自定义了两个策略，策略1、策略2。这两个策略也是随便自定义，现在实现MDP并不为对这个图进行强化学习出最好的策略，而是完整的走一遍，假设现在我们有了策略，那么下一步就是在求该策略下的状态价值函数，毕竟只有求出状态价值函数才能判断我们策略的好坏

MDP的状态价值函数虽然有了贝尔曼期望方程，但是好像并不好套现，不过在MRP是我们利用了贝尔曼方程求得了解析解的方法计算出了状态价值函数。那么能不能再确定的策略下将MDP转换为→MRP，然后进行求解？

答案是肯定的。我们可以将策略的动作选择进行**边缘化**（marginalization），就可以得到没有动作的MRP

对于某一个状态，我们根据策略所有动作的概率进行加权，得到的奖励和就可以认为是一个 MRP 在该状态下的奖励，即：

$$r'(s) = \sum_{a \in \mathcal{A}} \pi(a|s) r(s,a)$$

以策略1下的s_4奖励的边缘化为例：

$$r'(s_4) = 0.5 \times 10 + 0.5 \times 1 = 5.5$$

同理，我们计算采取动作的概率与使s转移s'到的概率的乘积，再将这些乘积相加，其和就是一个 MRP 的状态从s转移至s'的概率：

$$P'(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a)$$

以策略1下的s_4转移至s_2边缘化的状态转移概率为例：

$$P'(s_2|s_4) = 0.5 \times 0.2 = 0.1$$

边缘化在策略1下将MDP重构成了MRP

由价值函数定义知，转化前的 MDP 的状态价值函数和转化后的 MRP 的价值函数是一样的

故用 MRP 中计算价值函数的解析解来计算这个 MDP 中该策略的状态价值函数

这是一个很好的办法呀，将MDP转换为MRP，就可以不用考虑动作的因素，从而解决特定策略下状态价值函数的求解问题，然后分别求出各种策略下的状态价值函数进行求解，然后对各策略状态价值函数进行比较

但这个 MRP 解析解的方法在状态动作集合比较大的时候不是很适用，那有没有其他的方法呢？第 4 章将介绍用动态规划算法来计算得到价值函数。3.5 节将介绍用蒙特卡洛方法来近似估计这个价值函数

用蒙特卡洛方法的好处在于我们不需要知道 MDP 的状态转移函数和奖励函数，它可以得到一个近似值，并且采样数越多越准确。

策略1的状态价值：

编写代码来实现该方法，计算用**策略1**时的状态价值函数

边缘化后的处理，书上的代码已经计算好了，我简单的验证了几个例子是对的

```
gamma = 0.5
# 转化后的MRP的状态转移矩阵
P_from_mdp_to_mrp = [
    [0.5, 0.5, 0.0, 0.0, 0.0],
    [0.5, 0.0, 0.5, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.5, 0.5],
    [0.0, 0.1, 0.2, 0.2, 0.5],
    [0.0, 0.0, 0.0, 0.0, 1.0],
```

```
]
P_from_mdp_to_mrp = np.array(P_from_mdp_to_mrp)
R_from_mdp_to_mrp = [-0.5, -1.5, -1.0, 5.5, 0]

V = compute(P_from_mdp_to_mrp, R_from_mdp_to_mrp, gamma, 5)
print("MDP中每个状态价值分别为\n", V)

MDP中每个状态价值分别为
 [[-1.22555411]
 [-1.67666232]
 [ 0.51890482]
 [ 6.0756193 ]
 [ 0.        ]]
```

蒙特卡洛方法

蒙特卡洛方法是基于概率统计的数值计算方法，状态的价值是它的期望回报。所以很直观的想法就是根据选择的策略，在这个策略下采样很多条序列，求出每个序列在该状态下的累积回报，然后在求平均值作为该状态下的价值

$$V^{\pi}(\mathbf{s}) = \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

所以利用蒙特卡洛方法，我们不需要知道MDP的状态转移函数和奖励函数，是利用数据进行估计的近似值，并且采样数据越多，估计的越准确

我们在估计某个状态的价值，在一条序列中，可能没有出现这个状态，可能只出现过一次这个状态，也可能出现很多次这个状态。蒙特卡洛估计方法会在该状态每一次出现都计算它的回报，并记录该状态出现的次数和回报，并进行累加求出总次数和总汇报，然后求平均值作为该状态的价值

所以需要为每一个状态维护一个**计数器**和**总回报**

（1）根据使用的策略π进行采样，得到若干条状态转移动作序列

$$s_0^{(i)} \xrightarrow{a_0^{(i)}} r_0^{(i)}, s_1^{(i)} \xrightarrow{a_1^{(i)}} r_1^{(i)}, s_2^{(i)} \xrightarrow{a_2^{(i)}} \cdots \xrightarrow{a_{T-1}^{(i)}} r_{T-1}^{(i)}, s_T^{(i)}$$

（2）对每一条序列中的每一次出现的状态s进行计数器和总汇报的维护

- 更新状态**s**的计数器**N(s)** ← **N(s)+1**；
- 更新状态**s**的总回报**M(s)** ← **M(s)+G**；

（3）每一个状态的价值被估计为回报的期望**V(s) = M(s) / N(s)**

计算回报的期望时，除了可以把所有的回报加起来除以次数，还有一种增量更新的方法。对于每个状态s和对应回报G，进行如下计算：

- **N(s)** ← **N(s)+1**
- **V(s)** ← **V(s) + 1/N(s)(G - V(s))**

这种增量式的更新在第二章第8页估计期望奖励中有推导和使用过，通过迭代的方式更新价值，应该是为后面的动规dp做铺垫

占用度量

$$v^{\pi}(\mathbf{s})$$

用上述符号来表示在策略π下的状态访问分布，即$v^{\pi}(s_1)$表示在策略π下，访问状态s_1的概率，概率越大访问该状态的占比越高

同一个MDP的不同策略访问到的状态的概率分布是不同的，比如有一个策略根本不采取前往s_5这一动作，那么 $v^{\pi}(s_5) = 0$，所以如果一个策略始终不去执行前往s_5这一动作，那么该动作对应的数据就用永远无法被观测到，因此，智能体的策略不同，与环境交互产生的数据分布就不同。

而导致不同状态分布的产生与决策的动作有关，在策略π下的状态概率分布可以延申细化为策略π下状态和所要执行的动作的概率分布，即**状态动作对(s,a)的概率**，这是不同策略最后影响状态价值函数的关键，比如某个策略能得到更多奖励的状态动作对(s,a)访问概率大，说明该状态动作对(s,a)取得多，那么该策略的价值函数就更大

我们用策略的**占用度量**来定义动作状态对(s,a)被访问到的概率，$\rho^{\pi}(s,a)$

它与状态访问分布$v^{\pi}(s)$的关系为：

$$\rho^{\pi}(s,a) = \nu^{\pi}(s)\pi(a|s)$$

占用度量有两个性质：

- 当且仅当这两个占用度量相同时，这两个策略相同，也就是说，如果一个智能体的策略有所改变，那么它和环境交互得到的占用度量也会相应改变。

$$\rho^{\pi_1} = \rho^{\pi_2} \iff \pi_1 = \pi_2$$

- 给定一合法占用度量ρ，可生成该占用度量的唯一策略，该策略为

$$\pi_{\rho} = \frac{\rho(s,a)}{\sum_{a'}\rho(s,a')}$$

⚠️：以上提到的“合法”占用度量是指存在一个策略使智能体与 MDP 交互产生的状态动作对被访问到的概率。

所以通过以上性质可得，强化学习的策略在训练中会不断地更新，其对应的数据分布-占用度量也会相应地改变。因此，强化学习的一大难点就在于占用度量是随着智能体的学习而不断发生改变的。

由于奖励建立在状态动作对之上，一个策略对应的价值其实就是一个占用度量下对应的奖励和期望，因此寻找最优策略对应着寻找最优占用度量

最优策略

我们知道策略是决定状态s所要执行某个动作a的概率，那么什么要的决策是最好的呢？我们希望策略能达到的效果是什么样的呢？

强化学习的目标通常是找到一个策略，使得智能体从初始状态出发能获得最多的期望回报。

当且仅当对于任意的状态s都有$V^{\pi}(s) >= V^{\pi'}(s)$，记**π >= π'**（这里的$V^{\pi}(s)$是策略π下的状态s的价值函数，和小v的状态分布区分开

于是在有限状态和动作集合的 MDP 中，至少存在一个策略比其他所有策略都好或者至少存在一个策略不差于其他所有策略，这个策略就是**最优策略**（optimal policy）。最优策略可能有很多个，我们都将其表示为：

$$\pi^*(\mathbf{s})$$

最优策略都有相同的状态价值函数，我们称之为**最优状态价值函数**，表示为：

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in \mathcal{S}$$

同理，我们定义**最优动作价值函数**：

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

最优状态价值是选择此时使最优动作价值最大的那一个动作时的状态价值：

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$

根据贝尔曼期望方程，将方程中的状态价值函数和动作价值函数替换为最优状态价值函数和最优动作价值函数就可得**贝尔曼最优方程**：

$$V^*(s) = \max_{a \in \mathcal{A}} \{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a')$$

第 4 章将介绍如何用动态规划算法得到最优策略。