马尔可夫决策过程

简介

马尔可夫决策过程 (Markov Reward Process) 是强化学习的基础。要学习好强化学习我们首先得明确掌握马尔可夫决策过程的基础知识。我们通常说的强化学习中的环境一般就是一个马尔可夫决策过程。与多臂老虎机不同，马尔可夫决策过程包含了状态信息以及状态之间的转移机制。如果我们要用强化学习去解决一个实际问题，我们第一步要做的事情就是能够把这个实际问题抽象为一个马尔可夫决策过程，也就是明确马尔可夫决策过程的各个组成要素。在本节内容中，我们将从马尔可夫过程出发，一步一步进行介绍，最后引出马尔可夫决策过程。

马尔可夫过程（Markov Process）

随机过程（Stochastic Process）

随机过程是概率论的“动力学”部分。概率论的研究对象是静态的随机现象，而随机过程的研究对象是随时间演变的随机现象。例如，天气随时间的变化，城市交通随时间的变化。随机过程中，随机现象在某时刻t的取值被称为状态S_t，所有可能的状态组成状态集合$\mathcal{S}$。于是，随机现象研究的便是状态的变化过程。在某时刻t的状态S_t通常取决于t时刻之前的状态。我们将已知历史信息（$S_1, \ldots, S_t$）时下一个时刻状态为S_{t+1}的概率表示成$P(S_{t+1}|S_1, \ldots, S_t)$。

马尔可夫性质（Markov Property）

一个随机过程被称为具有马尔可夫性质，当且仅当某时刻的状态只却决于上一时刻的状态，用公式表示为$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \ldots, S_t)$。即下一个状态只取决于当前状态，而不会受到过去状态的影响。需要明确的是，马尔可夫性并不代表这个随机过程就和历史完全没有关系。因为虽然$t+1$时刻的状态只与t时刻的状态有关，但是t时刻的状态其实包含了$t-1$时刻的状态的信息，通过这种链式的关系，历史的信息被传递到了现在。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。

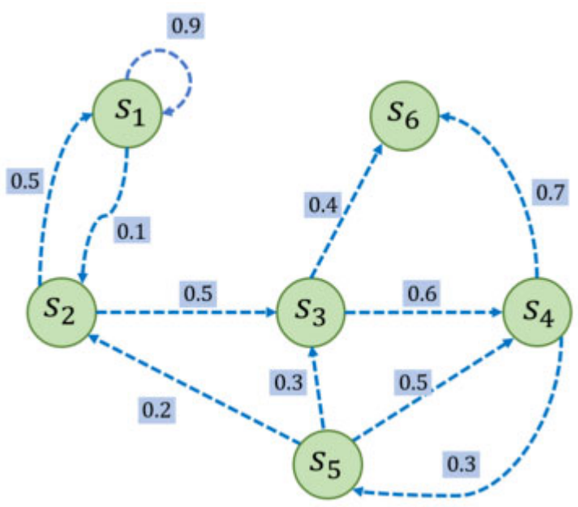
马尔可夫过程（Markov Process）

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（Markov Chain）。我们通常用元组$< \mathcal{S}, \mathcal{P} >$描述一个马尔可夫过程，其中$\mathcal{S}$是有限数量的状态集合，$\mathcal{P}$是状态转移矩阵（State Transition Matrix）。我们假设一共有n个状态，此时$\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$。状态转移矩阵$\mathcal{P}$定义了所有状态对之间的转移概率，即

$$\mathcal{P} = \begin{bmatrix} p(s_1|s_1) & \ldots & p(s_n|s_1) \\ \ldots & & \\ p(s_1|s_n) & \ldots & p(s_n|s_n) \end{bmatrix}$$

矩阵$\mathcal{P}$中第i行第j列元素$p(s_j|s_i) = P(S_{t+1} = s_j|S_t = s_i)$表示从状态$s_i$转移到状态$s_j$的概率，我们称$p(s'|s)$为状态转移函数。从某个状态出发，到达其他状态的概率和必须为1，即状态转移矩阵$\mathcal{P}$的每一行的和为1。

下图是一个具有6个状态的马尔可夫过程的简单例子。其中每个绿色圆圈表示一个状态，每个状态都有一定概率（包括概率为零）转移到其他状态，其中s_6通常被称为终止状态（terminal state），因为它不会再转移到其他状态，可以理解为它永远以概率1转移到自己。状态之间的虚线箭头表示状态的转移，箭头旁的数字表示该状态转移发生的概率。从每个状态出发转移到其他状态的概率总和为1。比如说，s_1有90%概率保持不变，有10%概率转移到s_2，而在s_2又有50%概率回到s_1，有50%概率转移到s_3。



我们可以写出这个马尔可夫过程的状态转移矩阵$\mathcal{P} = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0.2 & 0.3 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

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（sample）**。例如：从s_1出发，可以生成序列$s_1 - s_2 - s_3 - s_6$ 或序列 $s_1 - s_1 - s_2 - s_3 - s_4 - s_5 - s_3 - s_6$等等。生成这些序列的概率和状态转移矩阵有关。

马尔可夫奖励过程（Markov Reward Process）

在马尔可夫过程的基础上加入奖励函数 r和折扣因子 γ，我们就可以得到马尔可夫奖励过程。于是，一个马尔可夫奖励过程由$< \mathcal{S}, \mathcal{P}, r, \gamma >$构成，其中

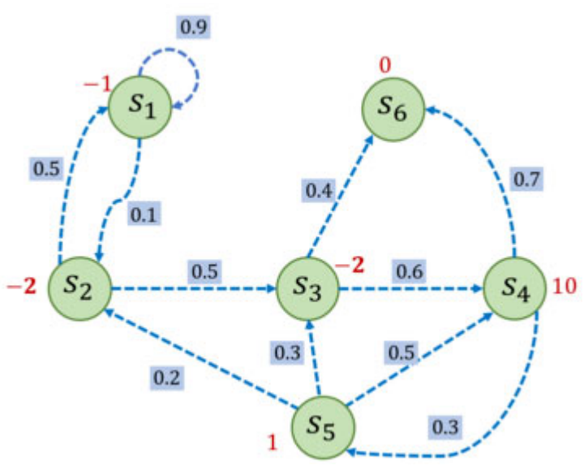
- $\mathcal{S}$是有限状态的集合
- $\mathcal{P}$是状态转移矩阵
- r是奖励函数，某个状态s的奖励 $r(s)$指转移到该状态时可以获得的奖励
- γ是折扣因子（Discount Factor），γ的取值范围为$[0, 1)$。引入折扣因子的理由为远期利益具有一定不确定性，有时我们更希望能够尽快获得一些奖励，所以我们需要对远期利益打一些折扣。接近1的γ更关注长期的累计奖励，接近0的γ更考虑短期奖励。

回报（Return）

在一个马尔可夫奖励过程中，从某一状态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，公式如下：

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

其中，R_t表示在时刻t获得的奖励。下图中，我们继续沿用之前马尔可夫链的例子，然后在此基础上添加了奖励函数，构建成一个马尔可夫奖励过程。比如，进入状态s_2可以得到奖励-2，表明我们不希望进入s_2，进入s_4可以获得最高的奖励10，但是进入s_6之后奖励为零，并且此时序列也终止了。



如果我们选取s_1为起始状态，设置γ=0.5，采样到一条状态序列为$s_1 - s_2 - s_3 - s_6$，就可以计算s_1的回报G_1，得到$G_1 = -1 + 0.5 \times (-2) + 0.5^2 \times (-2) = -2.5$

接下来我们将图中马尔可夫奖励过程用代码表示，并且定义计算回报的函数。

In [1]:

```
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，计算从某个索引开始到序列最后得到的回报
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
```

In [2]:

```
# 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（Value Function）

在马尔可夫奖励过程中，一个状态的期望回报被称为这个状态的价值（value）。所有状态的价值就组成了价值函数，价值函数的输入为某个状态，输出为这个状态的价值。我们将价值函数写成$V(s)=\mathbb{E}[G_t|S_t=s]$。我们将其展开为

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

上式的最后一个等号中，一方面，即时奖励的期望就等于即时奖励，即$\mathbb{E}[R_t|S_t=s]=r(s)$，另一方面，式子中剩余部分$\mathbb{E}[\gamma V(S_{t+1})|S_t=s]$可以根据从状态$s$出发的转移概率得到，即我们可以得到

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

上式就是马尔可夫奖励过程中非常有名的**贝尔曼方程（Bellman Equation）**。

上式的贝尔曼方程对每一个状态都成立。若一个马尔可夫奖励过程一共有n个状态，即$\mathcal{S}=\{s_1,s_2,\ldots,s_n\}$，我们将所有状态的价值表示成一个列向量$\mathcal{V}=[V(s_1),V(s_2),\ldots,V(s_n)]^T$，同理奖励函数写成一个列向量$\mathcal{R}=[r(s_1),r(s_2),\ldots,r(s_n)]^T$。于是我们可以将贝尔曼方程写成矩阵的形式

$$\mathcal{V}=\mathcal{R}+\gamma\mathcal{P}\mathcal{V}$$

$$\begin{bmatrix}V(s_1)\\V(s_2)\\\cdots\\V(s_n)\end{bmatrix}=\begin{bmatrix}r(s_1)\\r(s_2)\\\cdots\\r(s_n)\end{bmatrix}+\gamma\begin{bmatrix}p(s_1|s_1)&p(s_2|s_1)&\cdots&p(s_n|s_1)\\p(s_1|s_2)&p(s_2|s_2)&\cdots&p(s_n|s_2)\\\cdots&&&\\p(s_1|s_n)&p(s_2|s_n)&\cdots&p(s_n|s_n)\end{bmatrix}\begin{bmatrix}V(s_1)\\V(s_2)\\\cdots\\V(s_n)\end{bmatrix}$$

于是我们可以直接根据矩阵运算求解得到以下解析解：

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

但实际上，解析解的计算复杂度是$O(n^3)$，n是状态个数，所以这种方法只适用很小的马尔可夫奖励过程。求解较大规模的马尔可夫奖励过程中的价值函数，我们可以使用动态规划（Dynamic Programming），蒙特卡洛法（Monte-Carlo method），时序差分法（Temporal Difference），这些方法我们将会在之后学到。

我们接下来将求解价值函数的解析解方法用代码写出来，并据此计算上图MRP中所有状态的价值。

In [3]:

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

V = compute(P, rewards, gamma, 6)
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.        ]]
```

根据以上代码我们求解得到各个状态的价值函数$V(s)$，具体如下

$$\begin{bmatrix}V(s_1)\\V(s_2)\\V(s_3)\\V(s_4)\\V(s_5)\\V(s_6)\end{bmatrix}=\begin{bmatrix}-2.02\\-2.21\\1.16\\10.54\\3.59\\0\end{bmatrix}$$

我们现在用贝尔曼方程来进行简单验证。对于状态s_4来说，当$\gamma=0.5$时，我们有

$$V(s_4) = r(s_4) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_4)V(s')$$
$$10.54 = 10 + 0.5 \times (0.7 \times 0 + 0.3 \times 3.59)$$

我们发现左右两边是相等的，说明我们求解得到的价值函数是满足状态为s_4时的贝尔曼方程。同学们可以自行验证其他状态时候的贝尔曼方程是否也成立。若对于所有状态都成立，就可以说明我们求解得到的价值函数是正确的。马尔可夫奖励过程中的价值函数也可以通过蒙特卡洛的方法估计得到，我们将在本节课程接下来的马尔可夫决策过程中进行介绍该方法。

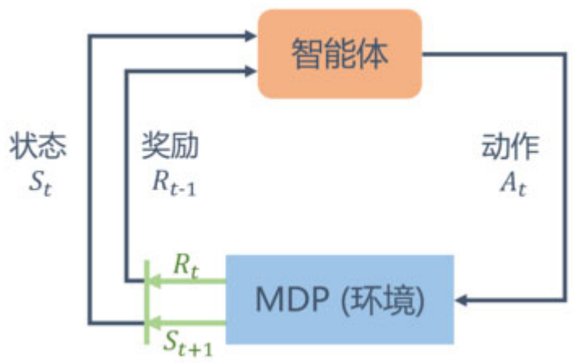
马尔可夫决策过程（Markov Decision Process）

在马尔可夫奖励过程MRP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元组$\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$构成，其中：

- $\mathcal{S}$是状态的集合
- $\mathcal{A}$是动作的集合
- γ是折扣因子
- $r(s, a)$是奖励函数，此时奖励同时取决于状态s和动作a
- $p(s'|s, a)$是状态转移函数，表示在状态s执行动作a之后到达状态s'的概率

我们发现MDP与MRP非常相像，主要区别为MDP中的状态转移函数和奖励函数都比MRP多了动作a作为自变量。注意在上面MDP的定义中，我们不再使用类似MRP定义中的状态转移矩阵方式，而是直接表示成了状态转移函数。一是此时状态转移与动作也有关，变成了一个三维数组，而不再是一个矩阵（二维数组）；二是状态转移函数更具有一般意义，例如当状态集合不是有限的时候，就无法用数组表示，但我们仍可以用状态转移函数表示。我们在之后的课程学习中会遇到连续状态的MDP环境，那时状态集合都不是有限的。现在我们主要关注于离散状态的MDP环境，此时状态集合是有限的。

不同于马尔可夫奖励过程，在马尔可夫决策过程中，通常存在一个智能体（agent）来执行动作。举个例子，如果一艘小船在大海中随着水流自由飘荡的过程就是一个马尔可夫奖励过程，它如果凭借运气漂到了一个目的地就能获得比较大的奖励；如果有个水手在控制着这条船往哪个方向前进，就可以主动选择前往目的地获得比较大的奖励。因为这是一个与时间相关的不断进行的过程，在智能体和环境MDP之间存在一个不断交互的过程。一般而言，它们之间的交互是如图循环过程：智能体根据当前状态S_t选择动作A_t；对于状态S_t和动作A_t，MDP根据奖励函数和状态转移函数得到S_{t+1}和R_t并反馈给智能体。智能体的目标是最大化得到的累计奖励。智能体根据当前状态从动作的集合$\mathcal{A}$中选择一个动作的函数，被称为策略。



策略（Policy）

智能体的策略通常用字母π表示。策略$\pi(a|s) = p(A_t = a|S_t = s)$是一个函数，表示在输入状态$s$情况下采取动作$a$的概率。当一个策略是**确定性策略**时，它在每个状态时只输出一个确定性的动作，即只有该动作的概率为1，其他动作的概率为0；当一个策略是**随机性**策略时，它在每个状态时输出的是关于动作的分布，然后根据该分布进行采样就可以得到一个动作。在MDP中，由于马尔可夫性质的存在，我们的策略只需要与当前状态有关，不需要考虑历史状态。回顾一下在MRP中的价值函数，在MDP中也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，这意为着对于两个不同的策略来说，它们在同一个状态上的价值也是不同的。这很好理解，因为不同的策略会采取不同的动作，从而之后会遇到不同的状态以及获得不同的奖励，于是它们的累计奖励的期望也就不同，即状态价值不同。

状态价值函数（State-value Function）

我们用$V^{\pi}(s)$表示在MDP中基于策略π的状态价值函数，定义为从状态s出发遵循策略π能获得的期望回报。它的数学表达为：

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|S_t = s]$$

动作价值函数（Action-value Function）

不同于MRP，在MDP中，由于动作的存在，我们额外定义一个动作价值函数。我们用$Q^{\pi}(s, a)$表示在MDP遵循策略π时，对当前状态s执行动作a得到的期望回报：

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t|S_t = s, A_t = a]$$

状态价值函数和动作价值函数之间的关系：

在使用策略π中，状态s的价值等于在该状态下基于策略π采取所有动作的价值与相应的概率相乘再求和的结果

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

在使用策略π中，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价值与相应的状态转移概率的乘积

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

贝尔曼期望方程（Bellman Expectation Equation）

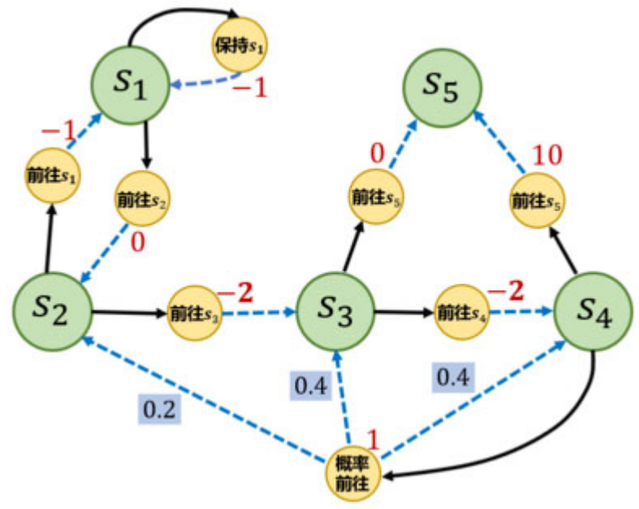
在贝尔曼方程中加上期望二字是为了与接下来的贝尔曼最优方程进行区分。我们通过简单推导就可以分别得到两个价值函数的贝尔曼期望方程：

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

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

价值函数和贝尔曼方程是强化学习中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。大家需要明确掌握！

下图是一个MDP的简单例子，其中每个绿色的圆圈代表一个状态，一共有从s_1到s_5这5个状态。黑色实线箭头代表可以采取的动作，黄色的小圆圈代表动作，需要注意，并非在每个状态都能采取每个动作，例如在状态s_1，智能体只能采取“保持s_1”和“前往s_2”这两个动作，无法采取其他动作。
每个红色的数字代表在某个状态时采取某个动作能获得的奖励。蓝色虚线箭头代表采取动作后可能转移到的状态，箭头边上的数字是转移概率，如果没有数字表示转移概率为1。比如，在s_2下， 如果采取动作“前往s_3”，就能得到奖励-2，并且转移到s_3，如果采取“前往s_1”，就能得到奖励-1，并且转移到s_1。在s_4下，如果采取“概率前往”，就能得到奖励1，并且会以概率0.2, 0.4, 0.4转移到s_2，s_3或s_4。



接下来我们将图中马尔可夫决策过程用代码表示，并且定义两个策略。第一个策略是一个随机策略，即在每个状态的时候，会以同样的概率选取它可能采取的动作，例如在s_1会以0.5和0.5概率选取动作“保持s_1”和“前往s_2”。第二个策略是一个我们提前设定的一个策略。

In [4]:

```
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
P = {
    "S1-保持S1-S1":1.0, "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,
    "S4-概率前往-S3":0.4, "S4-概率前往-S4":0.4,
}
# 奖励函数
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)

# 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，便于使用我们上述定义的P,R变量
def join(str1, str2):
    return str1 + '-' + str2
```

接下来我们想要计算该MDP的价值函数。我们现在有的工具是MRP的解析解方法。于是，一个很自然的想法是：给定一个MDP和一个策略π，我们是否可以将其转化为一个MRP？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了。具体的，对于某一个状态，我们计算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

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

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

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

于是，我们构建得到了一个MRP:$< \mathcal{S}, p', r', \gamma >$。根据价值函数的定义，我们可以发现转化前的MDP的状态价值函数和转化后的MRP的价值函数是一样的。于是我们可以用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。

我们现在就用代码实现来用该方法计算用随机策略（也就是代码中的Pi_1）时的状态价值函数。为了简单起见，我们直接给出转化后的MRP的状态转移矩阵和奖励函数，同学们可以自行验证。

In [5]:

```
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.        ]]
```

在知道了状态价值函数V后，我们可以计算动作价值函数Q。例如（s_4，概率前往）的动作价值为2.152，根据以下式子可以计算得到：

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

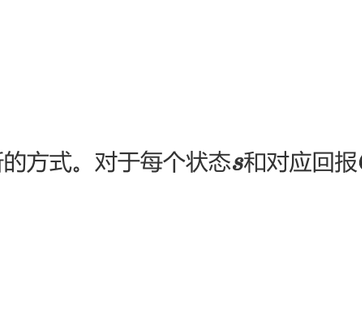
$$2.152 = 1 + 0.5 \times [0.2 \times (-1.68) + 0.4 \times 0.52 + 0.4 \times 6.08]$$

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？在下一节课程中我们将介绍用动态规划的方法来计算得到价值函数。本节课程接下来部分我们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，但是它只能得到一个近似值，并且采样数越多越准确。

蒙特卡洛方法（Monte-Carlo methods）

蒙特卡洛方法也被称为统计模拟方法，是一种基于概率统计的数值计算方法。运用蒙特卡洛方法时，我们通常使用重复随机抽样，然后运用概率统计方法来从抽样结果中归纳出我们想求的目标的数值估计。一个简单的例子是用蒙特卡洛方法来计算圆的面积。例如在这个正方形内部，随机产生若干个点，细数落在圆中点的个数，图中圆的面积与正方形面积之比就等于圆中点的个数与正方形中点的个数之比。如果我们随机产生的点的个数越多，计算得到圆的面积就越接近于真实的圆的面积。

$$\frac{\text{圆的面积}}{\text{矩形的面积}} = \frac{\text{圆中点的个数}}{\text{矩形中点的个数}}$$



我们现在介绍如何用蒙特卡洛方法来估计一个策略在一个马尔可夫决策过程中的状态价值函数。回忆一下一个状态的价值是它的期望回报，一个很直观的想法就是用策略在MDP上采样很多条序列，然后计算从这个状态出发的回报再求个平均值就可以了。也就是根据如下式子

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|S_t = s] \approx \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

在一条序列中可能没出现这个状态，也可能只出现过一次这个状态，也可能出现过很多次这个状态。我们介绍的蒙特卡洛价值估计算法中，会对每一次该状态的出现都计算它的回报。还有一种选择是一条序列只计算一次回报，也就是这条序列第一次出现该状态时候计算后面的累计奖励，而后面再次出现该状态就被忽略了。假设我们现在用策略π从状态s开始采样序列，据此来计算状态价值。我们为每一个状态维护一个计数器和总回报。具体过程为

1. 使用策略π采样

$$S_0^{(i)} \xrightarrow{A_0^{(i)}} R_0^{(i)}, S_1^{(i)} \xrightarrow{A_1^{(i)}} R_1^{(i)}, S_2^{(i)} \dots R_{T-1}^{(i)}, S_T^{(i)}$$

2. 对每一条序列中的每一时间步t的状态s进行以下操作

   - 更新状态s的计数器 $N(s) \leftarrow N(s) + 1$
   - 更新状态s的总回报 $M(s) \leftarrow M(s) + G_t$

3. 每一个状态的价值被估计为回报的平均值 $V(s) = M(s)/N(s)$

根据大数定律，当$N(s) \to \infty$，我们有$V(s) \to V^{\pi}(s)$

计算回报的平均值，除了把所有的回报加起来处以次数，还有一种增量更新的方式。对于每个状态s和对应回报G，我们进行

- $N(s) \leftarrow N(s) + 1$
- $V(s) \leftarrow V(s) + \frac{1}{N(s)}(G - V(S))$

这种增量式更新平均值的方法我们已经在多臂老虎机章节中看到过。

接下来我们用代码定义一个采样函数，需要遵守状态转移矩阵和相应的策略，我们每次将（s,a,r,s_next）元组放入序列中，直到到达终止序列。然后我们通过该函数实现用随机策略在图中MDP实际采样几条序列。

In [6]:

```
def sample(MDP, Pi, timestep_max, number):
    ''' 采样函数，策略Pi，限制最长时间步timestep_max，总共采样序列数number '''
    S, A, P, R, gamma = MDP
    episodes = []
    for _ in range(number):
        episode = []
        timestep = 0
        s = S[np.random.randint(4)] # 随机选择一个除了s5以外的状态s作为起点
        while s != "S5" and timestep <= timestep_max: # 当前状态为终止状态或者时间步太长时，一次采样结束
            timestep += 1
            rand, temp = np.random.rand(), 0
            # 在状态s根据策略选择动作
            for a_opt in A:
                temp += Pi.get(join(s, a_opt), 0)
                if temp > rand:
                    a = a_opt
                    r = R.get(join(s, a), 0)
                    break
            rand, temp = np.random.rand(), 0
            # 根据状态转移概率得到下一个状态s_next
            for s_opt in S:
                temp += P.get(join(join(s, a), s_opt), 0)
                if temp > rand:
                    s_next = s_opt
                    break
            episode.append((s, a, r, s_next)) # 把（s,a,r,s_next）元组放入序列中
            s = s_next # s_next变成当前状态，开始接下来的循环
        episodes.append(episode)
    return episodes

# 采样5次，每个序列最长不超过1000步
episodes = sample(MDP, Pi_1, 20, 5)
print('第一条序列\n', episodes[0])
print('第二条序列\n', episodes[1])
print('第五条序列\n', episodes[4])
```

```
第一条序列
 [('S1', '前往S2', 0, 'S2'), ('S2', '前往S3', -2, 'S3'), ('S3', '前往S5', 0, 'S5')]
第二条序列
 [('S4', '概率前往', 1, 'S4'), ('S4', '前往S5', 10, 'S5')]
第五条序列
 [('S2', '前往S3', -2, 'S3'), ('S3', '前往S4', -2, 'S4'), ('S4', '前往S5', 10, 'S5')]
```

In [7]:

```
# 对所有采样序列计算所有状态的价值
def MC(episodes, V, N, gamma):
    for episode in episodes:
        G = 0
        for i in range(len(episode)-1, -1, -1):##一个序列从后往前计算
            (s, a, r, s_next) = episode[i]
            G = r + gamma * G
            N[s] = N[s] + 1
            V[s] = V[s] + (G - V[s]) / N[s]

timestep_max = 20
# 采样1000次，可以自行修改
episodes = sample(MDP, Pi_1, timestep_max, 1000)
gamma = 0.5
V = {"S1":0, "S2":0, "S3":0, "S4":0, "S5":0}
N = {"S1":0, "S2":0, "S3":0, "S4":0, "S5":0}
MC(episodes, V, N, gamma)
print("使用蒙特卡洛法计算MDP状态价值为\n", V)
```

```
使用蒙特卡洛法计算MDP状态价值为
 {'S1': -1.228923788722258, 'S2': -1.6955696284402704, 'S3': 0.4823809701532294, 'S4': 5.967514743019431, 'S5': 0}
```

可以看到用蒙特卡洛方法估计得到的状态价值和我们用MRP解析解得到的状态价值是很接近的。这当然得益于我们采样了比较多的序列，同学们可以尝试一下修改采样次数然后观察蒙特卡洛方法的结果。

占用度量（Occupancy Measure）

我们在之前提到，不同策略的价值函数是不一样的。但这是为什么呢？原因就在于对于同一个MDP，不同策略会访问到的状态分布是不同的。想象一下，在如图的MDP中，我们现在有一个策略，它的动作执行会使得智能体尽快到达终止状态s_5，于是当它处于状态s_3时，它不会采取“前往s_4”的动作，而只会以1的概率采取“前往s_5”的动作，所以它不会获得在s_4状态下采取“前往s_5”可以得到的很大的奖励10。可想而知，根据贝尔曼方程，这个策略在状态s_3的状态会比较小，究其原因是因为它没法到达状态s_4。所以我们需要理解不同策略会访问到不同分布的状态这件事实，它会影响到策略的价值函数。

首先我们定义MDP的初始状态分布为$\nu_0(s)$，在有些材料中，初始状态分布会被定义进MDP的组成元素中。我们用$P(S_t = s)$表示在t时刻状态为s的概率，所以我们有$P(S_0 = s) = \nu_0(s)$，然后我们就可以定义一个策略的**状态访问分布（State Visitation Distribution）** 为

$$\nu^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_t^\pi(s)$$

其中$1-\gamma$是用来使得概率加和为1的归一化因子。状态访问概率表示一个策略和MDP交互它会访问到的状态的分布，需要注意理论上计算该分布的公式中需要交互到无穷步之后，但实际情况中智能体和MDP的交互在一个序列中是有限的，不过我们仍然可以用以上公式来表达状态访问概率的思想。状态访问概率有如下性质

$$\nu^\pi(s') = (1-\gamma)\nu_0(s') + \gamma\int p(s'|s,a)\pi(a|s)\nu^\pi(s)dsda$$

此外，我们还可以定义**占用度量（Occupancy Measure）** 为

$$\rho^\pi(s,a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_t^\pi(s)\pi(a|s)$$

它表示（动作，状态）对被访问到的概率。两者之间存在如下关系

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

我们有如下定理

定理1：和同一个MDP交互的两个策略π_1和π_2得到的占用度量ρ^{π_1}和ρ^{π_2}满足：$\rho^{\pi_1} = \rho^{\pi_2} \iff \pi_1 = \pi_2$

定理2：给定一占用度量ρ，可生成该占用度量的唯一策略是

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

接下来我们用代码来近似估计占用度量函数。这里我们采用近似估计，设置一个较大的episode length最大值，然后采样很多次，用(s,a)组出现的频率估计实际概率

In [8]:

```
def occupancy(episodes, s, a, timestep_max, gamma):
    ''' 计算状态-动作对（s，a）出现的频率，以此来估算策略的占用度量 '''
    rho = 0
    total_times = np.zeros(timestep_max) # 记录每个时间步t各被经历过几次
    occur_times = np.zeros(timestep_max) # 记录(s_t,a_t)=(s,a)的次数
    for episode in episodes:
        for i in range(len(episode)):
            (s_opt, a_opt, r, s_next) = episode[i]
            total_times[i] += 1
            if s == s_opt and a == a_opt:
                occur_times[i] += 1
    for i in reversed(range(timestep_max)):
        if total_times[i]:
            rho += gamma**i * occur_times[i]/total_times[i]
    return (1 - gamma) * rho
```

In [9]:

```
gamma = 0.5
timestep_max = 1000

episodes_1 = sample(MDP, Pi_1, timestep_max, 1000)
episodes_2 = sample(MDP, Pi_2, timestep_max, 1000)
rho_1 = occupancy(episodes_1, "S4", "概率前往", timestep_max, gamma)
rho_2 = occupancy(episodes_2, "S4", "概率前往", timestep_max, gamma)
print(rho_1, rho_2)
```

```
0.112567796310472 0.23199480615618912
```

我们可以发现不同策略对于同一个状态动作对的占用度量是不一样的。

最优策略（Optimal Policy）

强化学习的目标通常是找到一个策略使得它从初始状态出发能获得最多的期望回报。我们首先定义策略之间的偏序关系为$\pi > \pi'$ 当且仅当 对于任意的状态s都有$V^\pi(s) \geq V^{\pi'}(s)$。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一样好，这个策略就是**最优策略**。尽管可能最优策略有很多个，我们都将其表示为

$$\pi^*(s)$$

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**：

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

为了使得$Q^{\pi}(s,a)$最大，我们需要在当前的(s,a)之后都执行最优策略。于是我们得到了两者之间的关系：

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

这和普通策略的状价值函数和动作价值函数之间的关系是一样的。另一方面，最优价值函数是选择此时最优动作状态最大的那一个动作时的价值：

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

贝尔曼最优方程(Bellman Optimality Equation)

根据两者关系，我们可以得到**贝尔曼最优方程(Bellman Optimality Equation)**：

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

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

我们将在下一节课程中介绍如何用动态规划的方法得到最优策略。

总结

在本章内容中，我们从零开始学习了马尔可夫决策过程的一些基础概念知识，以及如何用求解贝尔曼方程得到状态价值的解析解和用蒙特卡洛的方法估计各个状态的价值。马尔可夫决策过程是强化学习的基础，之后学习的强化学习方法通常都是在求解马尔可夫决策过程中的最优策略。即强化学习中的环境就是一个马尔可夫决策过程。