强化学习基础篇（十四）价值迭代在FrozenLake中的实现

本节将主要基于gym环境中的FrozenLake-v0进行价值迭代的实现。

1. 价值迭代算法的伪代码

价值迭代算法，用于估计$\pi = \pi_*$

算法参数：小阈值$\theta > 0$，用于确定估计量的精度。
对于任意$s \in S^+$，任意初始化$V(s)$，其中$V(\text{终止状态}) = 0$
循环：
$\Delta \leftarrow 0$
对每一个$s \in S$循环：
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|$
直到$\Delta < \theta$
输出一个确定的$\pi \approx \pi_*$，使得
$\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

2.源代码

```python
import numpy as np

def calc_action_value(state, V, discount_factor=1.0):
        """
        Calculate the expected value of each action in a given state.
        对于给定的状态 s 计算其动作 a 的期望值
        """
        A = np.zeros(env.nA)
        for a in range(env.nA):
            for prob, next_state, reward, done in env.P[state][a]:
                A[a] += prob * (reward + discount_factor * V[next_state])
        return A

def value_iteration(env, theta=0.1, discount_factor=1.0):
    """
    Value Iteration Algorithm. 值迭代算法
    """
    # 初始化状态值
    V = np.zeros(env.nS)

    # 迭代计算找到最优的状态值函数 optimal value function
    for _ in range(50):
        delta = 0 # 停止标志位

        # 计算每个状态的状态值
        for s in range(env.nS):
            A = calc_action_value(s, V) # 执行一次找到当前状态的动作期望
            best_action_value = np.max(A) # 选择最好的动作期望作为新的状态值
```

```
            # 计算停止标志位
            delta = max(delta, np.abs(best_action_value - V[s]))

            # 更新状态值函数
            V[s] = best_action_value

        if delta < theta:
            break


    # 输出最优策略：通过最优状态值函数找到决定性策略
    policy = np.zeros([env.nS, env.nA]) # 初始化策略

    for s in range(env.nS):
        # 执行一次找到当前状态的最优状态值的动作期望 A
        A = calc_action_value(s, V)

        # 选出状态值最大的作为最优动作
        best_action = np.argmax(A)
        policy[s, best_action] = 1.0

    return policy, V

env = gym.make("FrozenLake-v0")
policy, v = value_iteration(env)

print("Reshaped Policy (0=up, 1=right, 2=down, 3=left):")
print(np.reshape(np.argmax(policy, axis=1), [4,4]))
print("")

print("Reshaped Value Function:")
print(v.reshape([4,4]))
print("")
```

运行后的最终策略和值函数如下：

```
Reshaped Policy (0=up, 1=right, 2=down, 3=left):
[[0 1 2 3]
 [0 0 0 0]
 [1 1 0 0]
 [0 2 1 0]]

Reshaped Value Function:
[[0.         0.         0.01234568 0.00411523]
 [0.         0.         0.06995885 0.        ]
 [0.02469136 0.14814815 0.26200274 0.        ]
 [0.         0.3127572  0.62688615 0.        ]]
```

3.代码解析

对于给定的状态 s 计算其动作 a 的期望值

主要依据公式$\sum_{s',r} p(s',r|s,a)[r+\gamma V(s')]$，通过遍历当前状态下的所有动作，获得当前状态动作的期望。

```python
def calc_action_value(state, V, discount_factor=1.0):
        """
        Calculate the expected value of each action in a given state.
        对于给定的状态 s 计算其动作 a 的期望值
        """
        A = np.zeros(env.nA)
        for a in range(env.nA):
            for prob, next_state, reward, done in env.P[state][a]:
                A[a] += prob * (reward + discount_factor * V[next_state])
        return A
```

在有了calc_action_value()函数后，就可以实现值迭代算法。在正式进行值迭代之前，采用np.zeros()函数将状态值向量都初始化为0。

由于折扣因子为1，可能会导致状态值的更新无法根据阀值停止的情况，因此需要使用截断的方式来控制遍历次数。在FrozenLake游戏环境中，状态的次数较少，因此迭代次数无需太多。根据经验，迭代次数设置为状态值的3-4倍（如16×3~50）就可以满足实际的迭代需求。

在算法值送代过程中，需要将策略初始化$[env.nS \times env.nA]$大小的矩阵，然后遍历每一个状态，找到使得状态值最大的动作（即最优动作），最后在策略矩阵中把该动作的位置（best action）设为1。
在上述代码中，当经过50次值选代或者满足条件$\Delta < \theta$时，就认为值选代算法已经找到最优状态v^*，而足有策略就是选择使得状态值最大的动作。因此，最后要做的就是根据最优状态值获得最优策略π^*。