RL Assignment 1

> 陈麒麟 517030910155

作业要求：

- 实现GridWorld类
- 用策略迭代和策略评估优化随机策略

作业完成：

- 实现Sutton版RL提供的GirdWorld类
- 按要求用策略迭代和策略评估优化随机策略
- 实现值迭代并进行迭代次数与终态对比

代码实现

GridWorld类

> 代码见 *gridworld.py*

该类实现了一个可以实例化的格子世界，概率转矩阵P、reward=-1已在其中声明，通过以下方法实例化：

```
env = GridworldEnv([m,n])
```

(m,n为自定义格子世界尺寸，默认出口在左上和右下)

PolicyIteration.py

> 代码见 *PolicyIteration.py*

- 首先使用GridWorld类实例化6x6的gridworld，之后通过函数 *policy_iteration* 集成策略迭代和策略评估：

```python
def policy_iteration(env, theta=0.001, discount_factor=1.0):
    """
    Policy Iteration Algorithm.

    Args:
        env: gridWorld
```

- 通过 *one_step_lookahead(state, V)* 按0.25的等概率计算更新后的值函数：

```python
def one_step_lookahead(state, V):

  A = 0.0
  for a in range(env.nA):
  for prob, next_state, reward, done in env.P[state][a]:
  A += 0.25 * (reward + discount_factor * V[next_state])
  return A
```

- 在theta=0.001的更新阈值下不断迭代

```python
V = np.zeros(env.nS)
Vtmp = np.zeros(env.nS)
iteration_step = 0
while True:
  iteration_step += 1
  # Stopping condition
  delta = 0
  # Update each state...
  for s in range(env.nS):
    # Calculate the new value
    new_action_value = one_step_lookahead(s, V)
    # Calculate delta across all states seen so far
    delta = max(delta, np.abs(new_action_value - V[s]))
    # Update the value function
    Vtmp[s] = new_action_value
      # Check if we can stop
      if delta < theta:
        print("iterations:",iteration_step)
        break
      else:
        V = Vtmp
```

- 通过 *greedy_policy_choose(state, V)* 进行一步的greedy策略选择：

```python
def greedy_policy_choose(state, V):
  A = np.zeros(env.nA)
  for a in range(env.nA):
    for prob, next_state, reward, done in env.P[state][a]:
      A[a] = V[next_state]
  return np.argmax(A)
```

ValueIteration.py

> 代码见 *ValueIteration.py*

与策略迭代类似，但是在每一次值迭代时选取最优策略，详见代码。

测试

6x6 Policy Iteration 结果

（最优策略、状态矩阵）

> theta=0.001 下迭代次数为259

```
Reshaped Grid Policy (0=n, 1=e, 2=s, 3=w):
[[0 3 3 3 3 3]
 [0 0 3 3 3 2]
 [0 0 0 3 2 2]
 [0 0 0 1 2 2]
 [0 0 1 1 1 2]
 [0 1 1 1 1 0]]

Final state:
[[  0.         -33.98388247 -51.89823156 -61.58573288 -66.50634971
  -68.50535173]
 [-33.98388247 -46.05530222 -56.12751562 -62.35536082 -65.43085527
  -66.5072893 ]
 [-51.89823156 -56.12751562 -60.20288998 -62.2791254  -62.3562168
  -61.5874427 ]
 [-61.58573288 -62.35536082 -62.2791254  -60.20370572 -56.12903392
  -51.90034928]
 [-66.50634971 -65.43085527 -62.3562168  -56.12903392 -46.05714509
  -33.98568766]
 [-68.50535173 -66.5072893  -61.5874427  -51.90034928 -33.98568766
    0.        ]]
```

6x6 Value Iteration 结果

（最优策略、状态矩阵）

> theta=0.001 下迭代次数为6

```
Reshaped Grid Policy (0=n, 1=e, 2=s, 3=w):
[[0 3 3 3 3 2]
 [0 0 0 0 0 2]
 [0 0 0 0 1 2]
 [0 0 0 1 1 2]
 [0 0 1 1 1 2]
 [0 1 1 1 1 0]]

Final state:
[[ 0. -1. -2. -3. -4. -5.]
 [-1. -2. -3. -4. -5. -4.]
 [-2. -3. -4. -5. -4. -3.]
 [-3. -4. -5. -4. -3. -2.]
 [-4. -5. -4. -3. -2. -1.]
 [-5. -4. -3. -2. -1.  0.]]
```

4x4 Policy Iteration 结果（验证）

（最优策略、状态矩阵）

> theta=0.001 下迭代次数为89

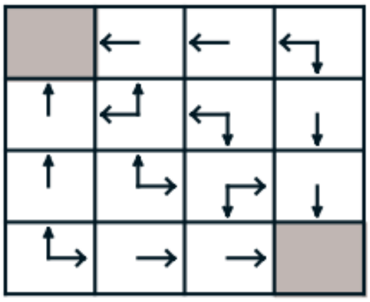
```
Reshaped Grid Policy (0=n, 1=e, 2=s, 3=w):
[[0 3 3 3]
 [0 3 3 2]
 [0 0 2 2]
 [0 1 1 0]]

Final state:
[[  0.         -13.99366052 -19.99088615 -21.98996852]
 [-13.99366052 -17.99222062 -19.99155339 -19.99164997]
 [-19.99088615 -19.99155339 -17.9928726  -13.9946786 ]
 [-21.98996852 -19.99164997 -13.9946786    0.        ]]
```

与textbook上结果一致，故验证通过：

$k = \infty$

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



结论

Policy迭代与评估方法实现正确，而且其迭代次数比value迭代要多。