1 Introdução

O objetivo desse laboratório foi implementar (em *Python*) algorítmos de programação dinâmica, seguindo o Processo Decisório de Markov (MDP). A tarefa recebida foi implementar num *grid* métodos de avaliação de política, iteração de política e iteração de valor. Para isso, foi pré-disponibilizado um código do *grid* para implementação.

2 Avaliação de política

2.1 Descrição do método

A avaliação de política realiza, iterativamente a solução da equação de um sistema linear. Após percorrer cada célula do *grid*, ele calcula o valor desta a partir do valor de suas células vizinhas válidas (método do *grid_world*). Por se tratar de um método iterativo, ele observa o critério de parada (número de iterações ou se todas as células satisfazem $|v_{k+1} - v_k| < \epsilon$).

2.2 Código

```
def policy_evaluation(grid_world, initial_value, policy, iteracoes=10000, epsilon=1.0e-5):
    cont=0
    faça:
    cont+=1
        s=(i,j) #percorre as tuplas do grid
        sum_a=0
        sum_s_line = 0
        for a in range(NUM_ACTIONS):
            sum_a+=policy[s][a]*grid_world.reward(s,a)

            for s_line in sucessores_validos(s):
                prob=grid_world.transition_probability(s,a,s_line)
                sum_s_line+=gamma*policy[s][a]*prob*value[s_line]
            value[s]=sum_a+sum_s_line
    enquanto (cont<iteracoes e max(value-vk)>epsilon)
    return value
```

2.3 Resultados e discussão

O resultado da avaliação de política foi testado para dois valores diferentes de p_c e γ, isto é, valor da probabilidade do agente executar uma ação corretamente e fator de desconto a cada iteração.

Primeiramente, para $p_c = \gamma = 1$, os resultados foram os seguintes:

```
Evaluating random policy, except for the goal state, where policy always executes stop:
Value function:
[  -384.09,  -382.73,  -381.19,    *    ,  -339.93,  -339.93]
[  -380.45,  -377.91,  -374.65,    *    ,  -334.93,  -334.93]
[  -374.35,  -368.82,  -359.85,  -344.88,  -324.92,  -324.93]
[  -368.76,  -358.18,  -346.03,    *    ,  -289.95,  -309.94]
[    *    ,  -344.12,  -315.06,  -250.02,  -229.99,    *    ]
[  -359.12,  -354.12,    *    ,  -200.01,  -145.00,     0.00]
Policy:
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[    *    ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,    *    ]
```

```
[  SURDL   ,  SURDL   ,    *     ,  SURDL   ,  SURDL   ,    S    ]
-----------------------------------------------------------------
```

E, posteriormente, para $p_c = 0,8$ e $\gamma = 0,98$:

```
Evaluating random policy, except for the goal state, where policy always executes stop:
Value function:
[   -47.19,   -47.11,   -47.01,    *    ,   -45.13,   -45.15]
[   -46.97,   -46.81,   -46.60,    *    ,   -44.58,   -44.65]
[   -46.58,   -46.21,   -45.62,   -44.79,   -43.40,   -43.63]
[   -46.20,   -45.41,   -44.42,    *    ,   -39.87,   -42.17]
[    *    ,   -44.31,   -41.64,   -35.28,   -32.96,    *    ]
[   -45.73,   -45.28,    *    ,   -29.68,   -21.88,     0.00]
Policy:
[  SURDL   ,  SURDL   ,  SURDL   ,    *    ,  SURDL   ,  SURDL  ]
[  SURDL   ,  SURDL   ,  SURDL   ,    *    ,  SURDL   ,  SURDL  ]
[  SURDL   ,  SURDL   ,  SURDL   ,  SURDL  ,  SURDL   ,  SURDL  ]
[  SURDL   ,  SURDL   ,  SURDL   ,    *    ,  SURDL   ,  SURDL  ]
[    *     ,  SURDL   ,  SURDL   ,  SURDL  ,  SURDL   ,    *    ]
[  SURDL   ,  SURDL   ,    *     ,  SURDL  ,  SURDL   ,    S    ]
-----------------------------------------------------------------
```

Tais resultados estão de acordo com o esperado uma vez que, fazendo p_c e γ serem diferentes de 1, deveria haver um desconto no módulo da função valor, o que significa que cada iteração vale um pouco menos que a iteração anterior.

3 Iteração de política

3.1 Descrição do método

Esse método avalia cada política chamando o método 2.2 a com *evaluations_per_policy* iterações para cada um deles. Após isso, ele muda a política de forma gulosa, utilizando a função *greedy_policy* pré-implementada. No final, ele utiliza os mesmos critérios de parada que em 2.2.

3.2 Código

```
def policy_iteration(grid_world, initial_value, initial_policy, evaluations_per_policy=3,
                     num_iterations=10000,epsilon=1.0e-5):
    value = np.copy(initial_value)
    policy = np.copy(initial_policy)
    i=0
    faça
        vk=np.copy(value)
        i+=1
        value=policy_evaluation(grid_world, value, policy,num_iterations=evaluations_per_policy)
        policy=greedy_policy(grid_world,value)

    enquanto (i<num_iterations e np.max(abs(vk-value))>epsilon):

    return value,policy
```

3.3 Resultados e discussão

Foram utilizados os mesmos valores apresentados em 2.3, obtendo os seguintes resultados: para $p_c = \gamma = 1$:

```
Policy iteration:
Value function:
[   -10.00,    -9.00,    -8.00,    *    ,    -6.00,    -7.00]
[    -9.00,    -8.00,    -7.00,    *    ,    -5.00,    -6.00]
[    -8.00,    -7.00,    -6.00,    -5.00,    -4.00,    -5.00]
[    -7.00,    -6.00,    -5.00,    *    ,    -3.00,    -4.00]
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
```

```
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
Policy:
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
------------------------------------------------------------------
```

E, para $p_c = 0,8$ e $\gamma = 0,98$:

```
Policy iteration:
Value function:
[   -11.65,   -10.78,    -9.86,    *    ,    -7.79,    -8.53]
[   -10.72,    -9.78,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.72,    -8.70,    -7.59,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.17,    -3.87,    -2.76,    *    ]
[    -8.63,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
Policy:
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    RD   ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
------------------------------------------------------------------
```

4 Iteração de valor

4.1 Descrição do método

A iteração de valor calcula qual ação gera um valor máximo de acordo com a equação:

$$v_*(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_\pi(s') \right), \tag{1}$$

calculando a expressão entre parenteses a cada iteração. O critério de parada é o mesmo que os anteriores (vide 2.2).

4.2 Código

```
def value_iteration(grid_world, initial_value, num_iterations=10000, epsilon=1.0e-5):
    dimensions = grid_world.dimensions
    gamma=grid_world.gamma
    value = np.copy(initial_value)
    cont=0
    faça:
        cont+=1
        vk=np.copy(value)
        s=(i,j) #percorre as células do grid
        max_s=-inf
        for a in range(NUM_ACTIONS):
            v_star = grid_world.reward(s, a)
            for s_line in grid_world.get_valid_sucessors(s):
                prob=grid_world.transition_probability(s,a,s_line)
                v_star+=gamma*prob*value[s_line]
            if v_star>max_s:
                max_s=v_star

                value[s]=max_s
```

```
    enquanto (i<num_iterations e np.max(abs(vk-value))>epsilon):
    return value
```

4.3 Resultados e discussão

Os parâmetros utilizados foram os mesmos que nos métodos anteriores (vide **??**. Para $p_c = \gamma = 1$:

```
Value iteration:
Value function:
[   -10.00,    -9.00,    -8.00,    *    ,    -6.00,    -7.00]
[    -9.00,    -8.00,    -7.00,    *    ,    -5.00,    -6.00]
[    -8.00,    -7.00,    -6.00,    -5.00,    -4.00,    -5.00]
[    -7.00,    -6.00,    -5.00,    *    ,    -3.00,    -4.00]
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
Policy:
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
-----------------------------------------------------------------
```

E, para $p_c = 0,8$ e $\gamma = 0,98$:

```
Value iteration:
Value function:
[   -11.65,   -10.78,    -9.86,    *    ,    -7.79,    -8.53]
[   -10.72,    -9.78,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.72,    -8.70,    -7.59,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.17,    -3.87,    -2.76,    *    ]
[    -8.63,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
Policy:
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    RD   ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
-----------------------------------------------------------------
```