# Infinite Horizon MDP

Realisé par:

**Mohamed EL YESSEFI**

**Yahya YOUNES**

**2A-$R$**

2022-2023

# Contents

**1 The optimality equation** **3**

**2 Solving the equation by value iteration** **4**

**3 Find the optimal policy by Policy Iteration** **5**

# 1 The optimality equation

L'équation d'optimalité : V*(s) $= max(R(s,a) + \gamma * \sum P(s'|s,a) *$ V*(s'))

$$V^*(0) = max(10 + \gamma * V^*(1), 1 + \gamma * V^*(0))$$

$$V^*(1) = max(0 + \gamma * V^*(1), -15 + \gamma * V^*(0))$$

pour $\gamma = 0$ :

$$V^*(0) = 10 + \gamma * V^*(1) \Rightarrow V^*(0) = 10$$

$$V^*(1) = 0 + \gamma * V^*(1) \Rightarrow V^*(1) = 0$$

pour $\gamma = 1$ :

$$V^*(0) = 1 + \gamma * V^*(0) \Rightarrow V^*(0) = \frac{1}{1-\gamma}$$

$$V^*(1) = -15 + \gamma * V^*(0) \Rightarrow V^*(1) = -15 + \frac{\gamma}{1-\gamma}$$

**Résolution analytique**

$$10 = (\frac{1}{1-\gamma}) \Rightarrow \gamma = 0.9$$

$$0 = -15 + (\frac{\gamma}{1-\gamma}) \Rightarrow \gamma = 0.9375$$

# 2 Solving the equation by value iteration

The python code that uses the value iteration :

+ Code + Texte

[35]

```python
import numpy as np

# This is the function that solves the optimality equation through value iteration.
def iterative_value_evaluation (P, R, discount_factor, epsilon=1e-4):
  # Initialize value function V
  V = np. zeros (n_states)

  # Value iteration
  while True:
    V_prev = V. copy ()
    for s in range(n_states):
        V[s] = max([R[s,a] + discount_factor * np. sum(P[s, :] * V_prev) for a in range(n_states)])
    if np.abs(V - V_prev).max() < epsilon :
        break

  # Calculate optimal policy
  policy = np.zeros (n_states, dtype=int)

  for s in range(n_states):
      policy[s] = np. argmax( [R[s, a] + discount_factor * np.sum(P[s, :] * V) for a in range(n_states)])

  return(V,policy)
```

[37]

```python
# Number of states
n_states = 2

# Number of actions
n_actions = 2

# Transition matrix
P = np.array ( [[0, 1], [0.9, 0.1]])

# Reward matrix
R = np. array ( [[10, 1], [-15, 0]])
  # Test the function
for discount_factor in [0, 0.9, 0.999]:
      V, policy = iterative_value_evaluation (P, R, discount_factor, epsilon=1e-4)
      print(f"Discount Factor = {discount_factor}: Optimal Policy = {policy}, Values = {V}")

```

Figure 1: A screenshot of our code on python

The results we get :

```
Discount Factor = 0: Optimal Policy = [0 1], Values = [10.  0.]
Discount Factor = 0.9: Optimal Policy = [0 1], Values = [50.27541761 44.75055573]
Discount Factor = 0.999: Optimal Policy = [0 1], Values = [4739.51366834 4734.24801619]
```

Figure 2: A screenshot of the result we get when running the python code

# 3 Find the optimal policy by Policy Iteration

We implemented this Python code that uses iteration policy to find the optimal policy for different values of $\gamma$

```python
import numpy as np

# Number of states
n_states = 2

# Number of actions
n_actions = 2

# Transition matrix
P = np.array([[10, 1], [0.9, 0.1]])

# Reward matrix
R = np.array([[10, 1], [-15, 0]])

# Function that evaluates a policy using value iteration.
def iterative_policy_evaluation(P, R, gamma, epsilon=1e-4):
    # Initialization of V
    V = np.zeros(n_states)

    # Value iteration
    while True:
        V_prev = V.copy()
        for s in range(n_states):
          V[s] = sum([policy[s] * (R[s, a] + gamma * np.dot(P[:, s, a], V_prev)) for a in range(n_actions)])
        if np.abs(V - V_prev).max() < epsilon:
            break
    return V

def iterative_policy_improvement(P, R, V, gamma):
    # Function that improves a policy using policy iteration
    # Initialization of the policy
    policy = np.zeros(n_states, dtype=int)

    # Policy improvement
    for s in range(n_states):
        policy[s] = np.argmax([R[s, a] + gamma * np.dot(P[:, s, a], V)] for a in range(n_states))
    return policy

```

```python


def iterative_policy_iteration(P, R, gamma, epsilon=1e-4):
    # Function that improves a policy using policy iteration
    # Initialization of the policy
    policy = np.zeros(n_states, dtype=int)

    # Policy iteration
    while True:
        # Policy evaluation
        V = iterative_policy_evaluation(P, R, gamma, epsilon)

        # Policy improvement
        policy_prev = policy.copy()
        policy = iterative_policy_improvement(P, R, V, gamma)

        # Stop if the policy converges
        if (policy == policy_prev).all():
            break
    return V, policy

```

Figure 3: A screenshot of the python code