CS640 Assignment 4: Markov Decision Process

In this assignment, you are asked to implement value iteration and policy iteration. We provide a script of skeleton code.

Your tasks are the following.

1. Implement the algorithms following the instruction.
2. Run experiments and produce results.

Do **not** modify the existing code, especially the variable names and function headers.

Submission

Everything you need to complete for this assignment is in this notebook. Once you finish, please save this file as PDF and submit it via Gradescope. Make sure the outputs are saved when you create the PDF file!

Collaboration

You must complete this assignment independently, but feel free to ask questions on Piazza. In particular, any questions that reveal your answers must be made private.

Packages

The package(s) imported in the following block should be sufficient for this assignment, but you are free to add more if necessary. However, keep in mind that you **should not** import and use any MDP package. If you have concern about an addition package, please contact us via Piazza.

In [ ]:

```
import numpy as np
import sys

np.random.seed(4) # for reproducibility
```

Examples for testing

The following block contains two examples used to test your code. You can create more for debugging, but please add it to a different block.

In [ ]:

```
# a small MDP
states = [0, 1, 2]
actions = [0, 1] # 0 : stay, 1 : jump
jump_probabilities = np.matrix([[0.1, 0.2, 0.7],
                                [0.5, 0, 0.5],
                                [0.6, 0.4, 0]])
for i in range(len(states)):
    jump_probabilities[i, :] /= jump_probabilities[i, :].sum()

rewards_stay = np.array([0, 8, 5])
rewards_jump = np.matrix([[-5, 5, 7],
                          [2, -4, 0],
                          [-3, 3, -3]])

T = np.zeros((len(states), len(actions), len(states)))
R = np.zeros((len(states), len(actions), len(states)))
for s in states:
    T[s, 0, s], R[s, 0, s] = 1, rewards_stay[s]
    T[s, 1, :], R[s, 1, :] = jump_probabilities[s, :], rewards_jump[s, :]

example_1 = (states, actions, T, R)

# a larger MDP
states = [0, 1, 2, 3, 4, 5, 6, 7]
actions = [0, 1, 2, 3, 4]
T = np.zeros((len(states), len(actions), len(states)))
R = np.zeros((len(states), len(actions), len(states)))
for a in actions:
    T[:, a, :] = np.random.uniform(0, 10, (len(states), len(states)))
    for s in states:
        T[s, a, :] /= T[s, a, :].sum()
    R[:, a, :] = np.random.uniform(-10, 10, (len(states), len(states)))

example_2 = (states, actions, T, R)
```

Value iteration

Implement value iteration by finishing the following function, and then run the cell.

In [ ]:

```
def value_iteration(states, actions, T, R, gamma = 0.1, tolerance = 1e-2, max_steps = 100):
```

```python
    Vs = [] # all state values
    Vs.append(np.zeros(len(states))) # initial state values
    steps, convergent = 0, False
    Q_values = np.zeros((len(states),len(actions)))
    while not convergent:
        ########################################################################
        # TO DO: compute state values, and append it to the list Vs
        # V_(k+1) = max_a sum_(next state s') T[s,a,s'] * (R[s,a,s'] + gamma * V-k(s))
        V_next = np.zeros(len(states))

        for s in states:
            V_next[s] = -sys.maxsize
            for a in actions:
                Q_value = 0
                for s_ in states:
                    Q_value += T[s,a,s_] * (R[s,a,s_] + gamma * Vs[-1][s])
                V_next[s] = max(V_next[s],Q_value)
                Q_values[s,a] = Q_value
        Vs.append(V_next)
        ############################ End of your code ##########################
        steps += 1
        convergent = np.linalg.norm(Vs[-1] - Vs[-2]) < tolerance or steps >= max_steps
    ########################################################################
    # TO DO: extract policy and name it "policy" to return
    # Vs should be optimal
    # the corresponding policy should also be the optimal one
    policy = np.argmax(Q_values,axis=1)
    ############################ End of your code ##########################
    return Vs, policy, steps

print("Example MDP 1")
states, actions, T, R = example_1
gamma, tolerance, max_steps = 0.1, 1e-2, 100
Vs, policy, steps = value_iteration(states, actions, T, R, gamma, tolerance, max_steps)
for i in range(steps):
    print("Step " + str(i))
    print("state values: " + str(Vs[i]))
    print()
print("Optimal policy: " + str(policy))
print()
print()
print("Example MDP 2")
states, actions, T, R = example_2
gamma, tolerance, max_steps = 0.1, 1e-2, 100
```

```
Vs, policy, steps = value_iteration(states, actions, T, R, gamma, tolerance, max_steps)
for i in range(steps):
    print("Step " + str(i))
    print("state values: " + str(Vs[i]))
    print()
print("Optimal policy: " + str(policy))
```

```
Example MDP 1
Step 0
state values: [0. 0. 0.]

Step 1
state values: [5.4 8.  5. ]

Step 2
state values: [5.94 8.8  5.5 ]

Step 3
state values: [5.994 8.88  5.55 ]

Step 4
state values: [5.9994 8.888  5.555 ]

Optimal policy: [1 0 0]


Example MDP 2
Step 0
state values: [0. 0. 0. 0. 0. 0. 0. 0.]

Step 1
state values: [2.23688505 2.67355205 2.18175138 4.3596377  3.41342719 2.97145478
 2.60531101 4.61040891]

Step 2
state values: [2.46057355 2.94090725 2.39992652 4.79560147 3.75476991 3.26860026
 2.86584211 5.0714498 ]

Step 3
state values: [2.4829424  2.96764277 2.42174403 4.83919785 3.78890418 3.2983148
 2.89189522 5.11755389]

Optimal policy: [0 2 0 3 3 3 2 3]
```

Policy iteration

Implement policy iteration by finishing the following function, and then run the cell.

In [ ]:

```
def policy_iteration(states, actions, T, R, gamma = 0.1, tolerance = 1e-2, max_steps = 100):
    policy_list = [] # all policies explored
    initial_policy = np.array([np.random.choice(actions) for s in states]) # random policy
    policy_list.append(initial_policy)
    Vs = [] # all state values
    Vs = [np.zeros(len(states))] # initial state values
    steps, convergent = 0, False
    while not convergent:
        ########################################################################
        # TO DO:
        # 1. Evaluate the current policy, and append the state values to the list Vs
        # V[policy_i][k+1][s] = sum_(s_) T[s,policy_i[s],s_] * ( R[s,policy_i[s],s_ + gamma * V[policy_i][k][s_] )
        V_next = np.zeros(len(states))

        for s in states:
            tmp = 0
            for s_ in states:
                tmp += T[s,policy_list[-1][s],s_] * ( R[s,policy_list[-1][s],s_] + gamma * Vs[-1][s_] )
            V_next[s] = tmp
        Vs.append(V_next)
        # 2. Extract the new policy, and append the new policy to the list policy_list
        # policy_list[i+1][s] = argmax_(a) sum_(s_) T[s,a,s_] * ( R[s,a,s_] + gamma * Vs[s_] )
        policy_new = np.array([np.random.choice(actions) for s in states])
        for s in states:
            new_tmp = np.zeros((len(actions)))
            for a in actions:

                sum = 0
                for s_ in states:
                    sum += T[s,a,s_] * ( R[s,a,s_] + gamma * Vs[-1][s_] )
                new_tmp[a] = sum
            policy_new[s] = np.argmax(new_tmp)
        policy_list.append(policy_new)
        ############################ End of your code ##########################
        steps += 1
        convergent = (policy_list[-1] == policy_list[-2]).all() or steps >= max_steps
    return Vs, policy_list, steps

print("Example MDP 1")
```

```python
states, actions, T, R = example_1
gamma, tolerance, max_steps = 0.1, 1e-2, 100
Vs, policy_list, steps = policy_iteration(states, actions, T, R, gamma, tolerance, max_steps)
for i in range(steps):
    print("Step " + str(i))
    print("state values: " + str(Vs[i]))
    print("policy: " + str(policy_list[i]))
    print()
print()
print("Example MDP 2")
states, actions, T, R = example_2
gamma, tolerance, max_steps = 0.1, 1e-2, 100
Vs, policy_list, steps = policy_iteration(states, actions, T, R, gamma, tolerance, max_steps)
for i in range(steps):
    print("Step " + str(i))
    print("state values: " + str(Vs[i]))
    print("policy: " + str(policy_list[i]))
    print()
```

```
Example MDP 1
Step 0
state values: [0. 0. 0.]
policy: [0 1 1]

Step 1
state values: [ 0.   1.  -0.6]
policy: [1 0 0]


Example MDP 2
Step 0
state values: [0. 0. 0. 0. 0. 0. 0. 0.]
policy: [3 2 1 4 3 3 4 0]

Step 1
state values: [ 1.79546043  2.67355205 -0.08665637 -4.92536024  3.41342719  2.97145478
 -1.69624246  2.48967841]
policy: [0 2 0 3 3 3 2 3]
```

More testing

The following block tests both of your implementations. Simply run the cell.

In [ ]:

```
steps_list_vi, steps_list_pi = [], []
for i in range(20):
    states = [j for j in range(np.random.randint(5, 30))]
    actions = [j for j in range(np.random.randint(2, states[-1]))]
    T = np.zeros((len(states), len(actions), len(states)))
    R = np.zeros((len(states), len(actions), len(states)))
    for a in actions:
        T[:, a, :] = np.random.uniform(0, 10, (len(states), len(states)))
        for s in states:
            T[s, a, :] /= T[s, a, :].sum()
        R[:, a, :] = np.random.uniform(-10, 10, (len(states), len(states)))
    Vs, policy, steps_v = value_iteration(states, actions, T, R)
    Vs, policy_list, steps_p = policy_iteration(states, actions, T, R)
    steps_list_vi.append(steps_v)
    steps_list_pi.append(steps_p)
print("Numbers of steps in value iteration: " + str(steps_list_vi))
print("Numbers of steps in policy iteration: " + str(steps_list_pi))
```

```
Numbers of steps in value iteration: [4, 4, 5, 4, 5, 5, 5, 4, 4, 4, 5, 5, 4, 5, 5, 4, 5, 4, 5, 5]
Numbers of steps in policy iteration: [2, 2, 2, 2, 2, 3, 2, 3, 2, 2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2]
```