### Solution of Jack's Car Rental

#### Model building

Think of this problem as a continuous finite MDP, where the time step is days. The first thing to do is to define what the "state" and "action" are. The state is the number of cars left at each location at the end of each day, and the action is the net number of cars moved between the two locations each night. Then we use dynamic programming to solve it.

In the entire MDP, the transition between states is fixed (independent of actions, but follows a Poisson arrival process), so we can first construct the transition probability table between states. The possible states (number of cars) at each rental location at night are , the two rental locations correspond to crossing *21\*21=441* situations. Therefore, the state table "Tp" is a *21\*21* matrix. The corresponding rewards for state transitions are also fixed and recorded in the table "reward".

From a certain state, after renting and returning cars, we transition to a new state. We can calculate the probability of transitioning to the next state.

#### Solution approach

##### Policy iteration

For policy iteration, initialize a policy, then perform policy evaluation and improvement until convergence.

**step 1: Policy evaluation**

Elementwise form:

Stop when or is sufficently large or is sufficiently small.

**step 2: Policy improvement**

The elementwise form of

is

Here, is the action value under policy . Let

Then, the greedy policy is

**Pseudocode:**

While the policy has not converged, for the kth iteration, do

Policy evaluation:

Initialization: an arbitrary initial guess

While has not convered, for the jth iteration

For every state , do

Policy improvement:

For every state , do

For every acion , do

*1* if *,* and *0* otherwise

##### Value iteration

For value iteration, initialize a value function and iteratively update it until convergence.

**step 1: Policy update**

Elementwise form:

The optimal policy solving the optimazation problem is

where *.*

**step 2: Value update**

Since is greedy, the above equation is simply

**Pseudocode:**

While the has not converged in the sence that is greater than a predefined small threshold, for the *k*th iteration, do

For every state , do

For every acion , do

Maximum action value:

Policy update: *1* if *,* and *0* otherwise

Value update:

#### Core code

# policy iteration

def policy\_iteration():

    iteration = 0

    while True:

    # policy evaluattion

        while True:

            old\_value = value.copy()

            for i in range(max\_car\_num + 1):

                for j in range(max\_car\_num + 1):

                    new\_state\_value = value\_update([i, j], policy[i, j], value)

                    value[i, j] = new\_state\_value

            max\_value\_change = abs(old\_value - value).max()

            print(f'max value change: {max\_value\_change}')

            if max\_value\_change < 1e-4:

                break

        # policy improvement

        policy\_stable = True

        for i in range(max\_car\_num + 1):

            for j in range(max\_car\_num + 1):

                old\_action = policy[i, j]

                action\_value = []

                for action in actions:

                    if -j <= action <= i:  # valid action

                        action\_value.append(value\_update([i, j], action, value))

                    else:

                        action\_value.append(-np.inf)

                action\_value = np.array(action\_value)

                # greedy policy

                new\_action = actions[np.where(action\_value == action\_value.max())[0]]

                policy[i, j] = np.random.choice(new\_action)

                if policy\_stable and (old\_action not in new\_action):

                    policy\_stable = False

        iteration += 1

        print('iteration: {}, policy stable {}'.format(iteration, policy\_stable))

        draw\_fig(value, policy, iteration, 0)

        if policy\_stable:

            break

# value iteration

def value\_iteration():

    iteration = 0

    while True:

        delta = 0

        for i in range(max\_car\_num + 1):

            for j in range(max\_car\_num + 1):

                old\_value = value[i, j]

                action\_value = []

                for action in actions:

                    if -j <= action <= i:  # valid action

                        action\_value.append(value\_update([i, j], action, value))

                    else:

                        action\_value.append(-np.inf)

                action\_value = np.array(action\_value)

                # greedy policy

                # policy update

                new\_action = actions[np.where(action\_value == action\_value.max())[0]]

                policy[i, j] = np.random.choice(new\_action)

                # value update

                new\_value = action\_value.max()

                value[i, j] = new\_value

                delta = max(delta, abs(old\_value - new\_value))

        iteration += 1

        print('iteration: {}, delta: {}'.format(iteration, delta))

        draw\_fig(value, policy, iteration, 1)

        if delta < 0.1:

            break

#### Result



