日期	科目	班级	姓名	学号
2023 年 3 月 10 日	强化学习	强基数学 002	吴天阳	2204210460

第四章作业

题目 1. 例 4.2（杰克租车问题） Jack 管理一家有两个地点 A、B 的汽车公司，每天会有用户前往一个地点租车，如果租出去一辆车，则可获得 10 美元收益，如果在那个地点没有汽车则无法租出去；同时每天还会有人还车，还的车在第二天变得可用. 我们假设每天租车数量和还车数量分别满足期望为 λ, σ 的 Poisson 分布 $\text{Poi}(\lambda) \sim \frac{\lambda^n}{n!}\text{e}^{-\lambda}$，

同时 Jack 还可以在每天晚上将 A、B 的汽车进行移动，每次移动一辆车代价为 2 美元，并且每天最多移动 5 辆车，且假设一个地点的车辆数目不能超过 20 辆车，额外的车会自动消失.

我们令折扣率 $\gamma = 0.9$，状态为每天结束时每个地点的车辆数目，动作为夜间在两个地点移动的车辆数目.

解答. 首先用数学符号将状态、动作和收益表示出来：

状态：设 nA_t, nB_t 分别表示第 t 天结束时 A, B 两地的车辆数目，则状态 $S_t = (nA_t, nB_t)$. 假设第 t 天 A 地归还的车辆数为 $inA_t \sim \text{Poi}(3)$，B 地归还的车辆数为 $inB_t \sim \text{Poi}(2)$，于是状态 $S_t = (nA_t, nB_t)$ 满足：

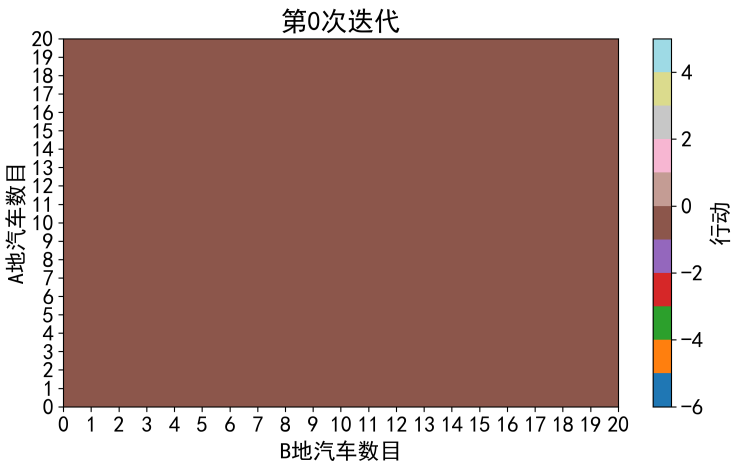
$$nA_t = \min\{nA_{t-1} - outA_t + move_t + inA_t, 20\}$$
$$nB_t = \min\{nB_{t-1} - outB_t - move_t + inB_t, 20\}$$

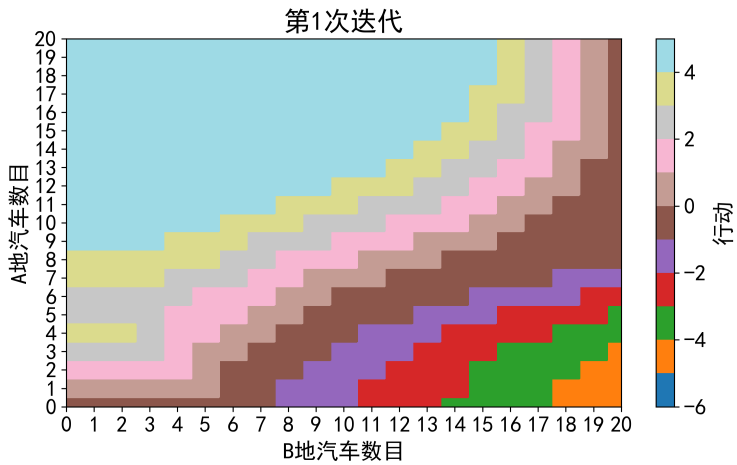
动作：设 $move_t \in [-5, 5]$，表示第 t 天晚上移动的车辆，当 $move_t > 0$ 表示从 A 处移出 $move_t$ 辆车到 B 处；当 $move_t < 0$ 表示从 B 处移出 $-move_t$ 辆车到 A 处.

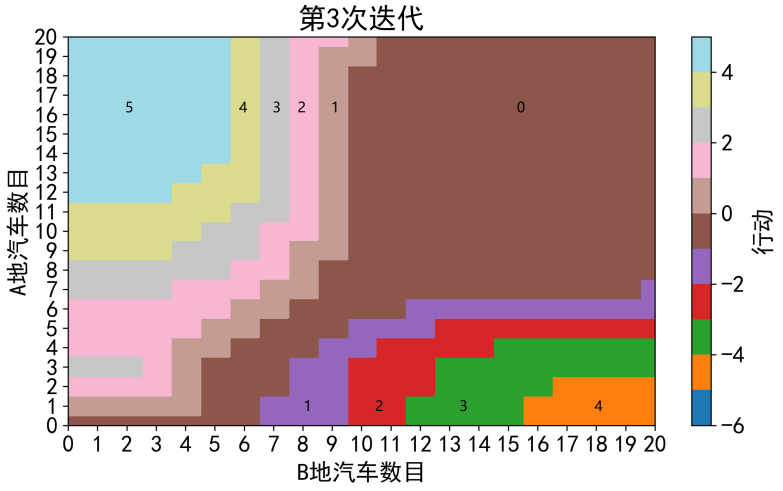
收益：设 A, B 两地第 t 天成功租出的车数量分别为 $outA_t = \min(xA, nA_{t-1}), outB_t = \min(xB, nB_{t-1})$，其中 $xA \sim \text{Poi}(3), xB \sim \text{Poi}(4)$ 分别表示第 t 天来租车的人数. 则第 t 天的收益为

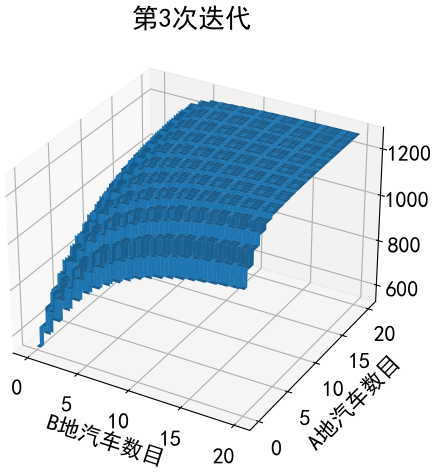
$$R_t = 10(outA + outB) - 2|move_t|$$

通过 3 次迭代策略达到稳定，第 0,1,3 次策略和最终的状态价值函数如下图所示：









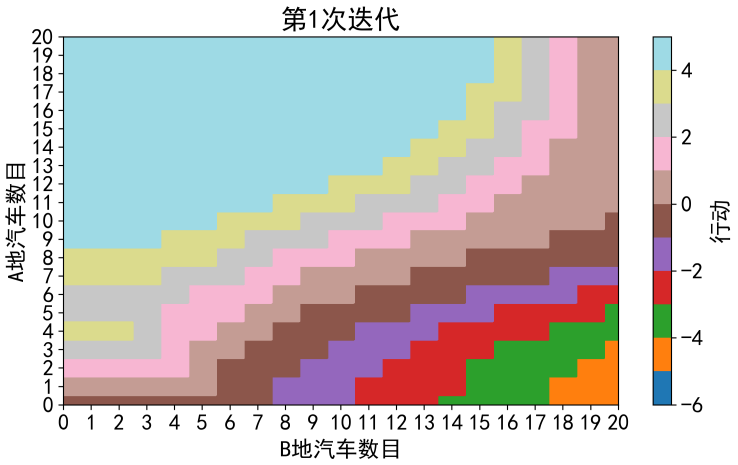
题目 2. 练习 4.7 使用动态规划方法解决以下更改过的 Jack 租车问题，Jack 一个员工在 A 地点上班，每晚会做公车达到 B 地点，她可以免费将 A 地点的一辆车移动到 B 地点，其他车辆的移动仍需要 2 美元，并且 Jack 的停车场有限，如果在某个地点过夜的车辆数目大于 10 辆，则需要在另一个停车场支付 4 美元（无论多少车）来停车.

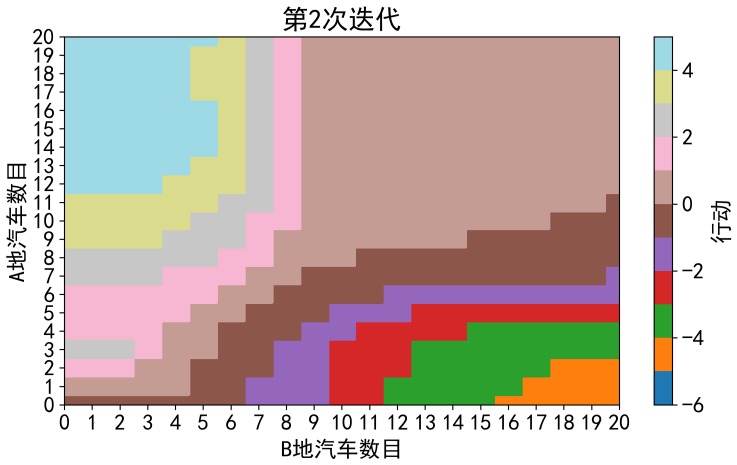
解答. 只需对例 4.2 的基础上对 R_t 稍加修改即可：

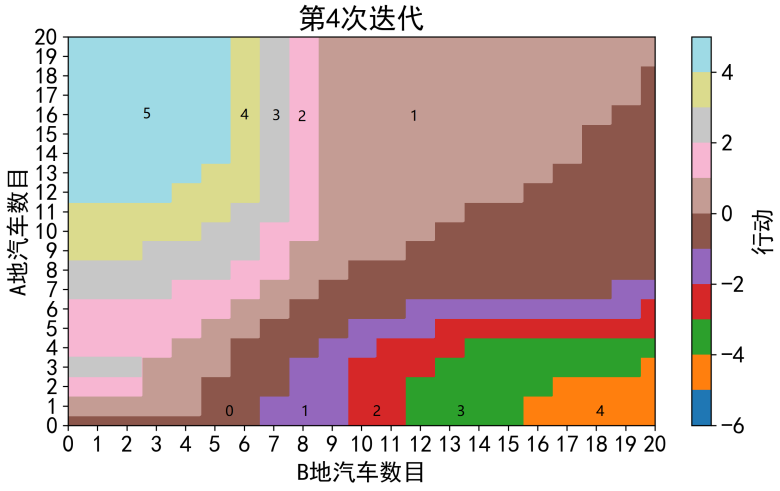
$$R_t = \begin{cases} 10(outA + outB) - 2(move_t - 1) - 4 \cdot \mathbb{1}_{nA_t>10} - 4 \cdot \mathbb{1}_{nB_t>10}, & move_t > 0, \\ 10(outA + outB) + 2move_t - 4 \cdot \mathbb{1}_{nA_t>10} - 4 \cdot \mathbb{1}_{nB_t>10}, & move_t \leqslant 0, \end{cases}$$

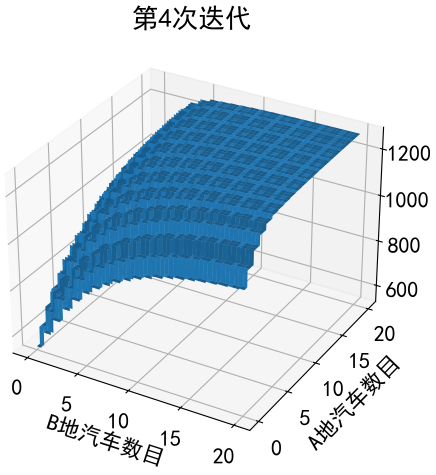
其中 $\mathbb{1}_{nA_t>10}$ 表示当 $nA_t > 10$ 时为 1，否则为 0.

通过 4 次迭代达到稳定，第 1,2,4 次策略和最终的状态价值函数如下









例 4.2 代码：

```python
import math
import numpy as np
import matplotlib as mpl
from tqdm import tqdm
import matplotlib.pyplot as plt

config = {  # matplotlib 绘图配置
    "figure.figsize": (6, 6),  # 图像大小
    "font.size": 16, # 字号大小
    "font.sans-serif": ['SimHei'],   # 用黑体显示中文
    'axes.unicode_minus': False # 显示负号
}
plt.rcParams.update(config)

class Environment:
    def __init__(self, out_lambda, in_lambda, action_sign) -> None:
        self.out_lambda, self.in_lambda = out_lambda, in_lambda
        self.action_sign = action_sign  # 用于判断 action 前的符号

    def step(self, state, action, in_num, out_num):
        out_num = min(out_num, state)
        new_state = max(min(state - out_num + action * self.action_sign + in_num,
        ↪  20), 0)
        reward = 10 * out_num
        return int(new_state), reward

    def poisson(self, x, lamb):
        return np.power(lamb, x) / math.factorial(x) * np.exp(-lamb)

    def get_prob(self, in_num, out_num):
        prob = self.poisson(in_num, self.in_lambda) * self.poisson(out_num,
        ↪  self.out_lambda)
        return prob


```

```python
class Policy_iterate:
    def __init__(self, T=10, sample_num=5000, max_storageA=20, max_storageB=20, gamma=0.9) -> None:
        self.T = T
        self.sample_num = sample_num
        self.max_storageA, self.max_storageB = max_storageA, max_storageB
        self.gamma = gamma
        self.min_delta_value = 1e-3  # 价值函数改变量阈值
        self.envA = Environment(out_lambda=3, in_lambda=3, action_sign=-1)
        self.envB = Environment(out_lambda=4, in_lambda=2, action_sign=1)

    def work(self):
        self.policy = np.zeros((self.max_storageA+1, self.max_storageB+1))
        self.value = np.zeros((self.max_storageA+1, self.max_storageB+1))
        for _ in range(self.T):
            self.plot_policy(_)
            self.policy_estimate()
            delta_value, policy_stable = self.policy_improve()
            if delta_value <= self.min_delta_value:
                print(f" 第{_}次迭代，状态价值变换{delta_value:.3f}小于阈值{self.min_delta_value:.3f}，退出迭代")
                break
            if policy_stable is True:
                print(f" 第{_}次迭代，策略已稳定，退出迭代")
                break

    def plot_policy(self, T):
        print(f" 第{T}次迭代")
        print(self.policy)

        # 可视化策略
        plt.figure(figsize=(8, 5))
        x1, x2 = np.meshgrid(  # 创建离散网格点
            np.linspace(0, 20, 500),
            np.linspace(0, 20, 500)
        )
        # 计算高度值
        z = np.zeros_like(x1)
        for i in range(z.shape[0]):
            for j in range(z.shape[1]):
                z[i, j] = self.policy[round(x2[i, j]), round(x1[i, j])]
        # 设置等高线划分点，会根据情况绘制等高线，若没有相应的数据点，则不会进行绘制
        levelz = range(-6, 6)
        plt.contourf(x1, x2, z, levels=levelz, cmap='tab20')  # 向由等高线划分的区域填充颜色
        plt.colorbar(label='行动')  # 制作右侧颜色柱，表示每种颜色对应的值

        plt.xlabel('B 地汽车数目')
        plt.ylabel('A 地汽车数目')
        plt.xticks(range(0, 21))
        plt.yticks(range(0, 21))
        plt.grid(False)
        plt.title(f" 第{T}次迭代")
```

```
        plt.tight_layout()
        plt.savefig(f"policy{T}.png", dpi=300)

        # 可视化状态价值
        fig = plt.figure(figsize=(5, 5))
        ax = fig.add_subplot(projection='3d')
        x1, x2 = np.meshgrid(  # 创建离散网格点
            np.linspace(0, 20, 1000),
            np.linspace(0, 20, 1000)
        )
        z = np.zeros_like(x1)
        for i in range(z.shape[0]):
            for j in range(z.shape[1]):
                z[i, j] = self.value[round(x2[i, j]), round(x1[i, j])]
        ax.plot_surface(x1, x2, z)
        ax.set_xlabel("B 地汽车数目")
        ax.set_ylabel("A 地汽车数目")
        plt.title(f" 第{T}次迭代")
        plt.tight_layout()
        plt.savefig(f"value{T}.png", dpi=300)

        np.savetxt(f"policy{T}.txt", self.policy, fmt="%.0f")
        np.savetxt(f"value{T}.txt", self.value, fmt="%.2f")

    def calculate_value(self, stateA, stateB, action, value):
        new_value = 0
        for in_numA in range(6):
            for out_numA in range(6):
                probA = self.envA.get_prob(in_numA, out_numA)
                if probA < 0.0001:
                    continue
                for in_numB in range(6):
                    for out_numB in range(1, 7):
                        prob = probA * self.envB.get_prob(in_numB, out_numB)
                        new_stateA, rewardA = self.envA.step(stateA, action,
                        ↪  in_numA, out_numA)
                        new_stateB, rewardB = self.envB.step(stateB, action,
                        ↪  in_numB, out_numB)
                        reward = 10 * (rewardA + rewardB) - 2 * abs(action)
                        new_value += prob * (reward + self.gamma *
                        ↪  value[new_stateA, new_stateB])
        return new_value

    def policy_estimate(self):
        min_delta_value = 1e-3
        T = 100
        for _ in range(T):
            value_backup = self.value.copy()
            delta_value = 0
            for stateA in range(self.max_storageA+1):
                for stateB in range(self.max_storageB+1):
                    old_value = self.value[stateA, stateB]
                    action = self.policy[stateA, stateB]
```

```
                    new_value = self.calculate_value(stateA, stateB, action,
                    ↪  value_backup)
                    self.value[stateA, stateB] = new_value
                    delta_value = max(delta_value, abs(new_value - old_value))
            print(delta_value)
            if delta_value <= min_delta_value:
                break

    def policy_improve(self):
        delta_value, policy_stable = 0, True
        for stateA in range(self.max_storageA+1):
            for stateB in range(self.max_storageB+1):
                old_action = self.policy[stateA, stateB]
                max_value = 0
                for action in range(-5, 6):
                    if (action < 0 and stateB < -action) or (action > 0 and
                    ↪  stateA < action):
                        continue
                    new_value = self.calculate_value(stateA, stateB, action,
                    ↪  self.value)
                    if new_value > max_value:
                        max_value = new_value
                        self.policy[stateA, stateB] = action
                if old_action != self.policy[stateA, stateB]:
                    policy_stable = False
                delta_value = max(delta_value, abs(self.value[stateA, stateB] -
                ↪  max_value))
        return delta_value, policy_stable

if __name__ == '__main__':
    np.random.seed(42)
    agent = Policy_iterate()
    agent.work()
```

练习 4.7 代码：将**例 4.2 代码**中 120 行收益函数改为

```
    reward = -4 * (new_stateA > 10) - 4 * (new_stateB > 10)  # 额外停车费
    if action > 0:
        reward = 10 * (rewardA + rewardB) - 2 * (action - 1)
    else:
        reward = 10 * (rewardA + rewardB) + 2 * action
```