问题：

Jack管理着一家有两个场地的小型的租车公司（分别称为first location和second location，每租出一部车，Jack可赚10美金。为了提高车子的出租率，Jack在夜间调配车辆，即把车子从一个场地调配到另外一个场地，成本是2美金每辆。假设每个场地每天租车和还车的数量是泊松随机变量，即其数值是n的概率为$\frac{\lambda^n}{n!}e^{-\lambda}$，其中λλ为期望。假设场地1和场地2租车的的λ分别为3和4，还车的分别λ为3和2。为了简化问题起见，我们假设每个场地最多可停20部车（如果归还的车辆超出了20部，我们假设超出的车辆无偿调配到了别的地方，比如总公司)，并且每个场地每天最多调配5部车子。

请问Jack在每个场地应该部署多少部车子？每天晚上如何调配？

问题分析：

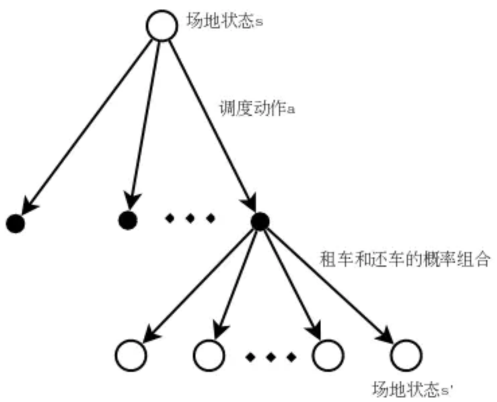
· 状态空间：1号租车点和2号租车点分别拥有的可供租赁的车辆，不大于20辆；

> 状态数量 $21 \times 21 = 441$

· 行为空间：下班后从一个租车点转移到另一个租车点的车辆，不大于5两；

> 动作集合
>
> $A = \{(-5,5),(-4,4),(-3,3),(-2,2),(-1,1),(0,0),(1,-1),(2,-2),$
>
> $(3,-3),(4,-4),(5,-5)\}$

场地状态和动作和租车换车的关系如下：



· 即时奖励：每租出去一辆车获得10美金；

· 转移概率：租车和换车的数量是服从泊松分布的随机变量，参数见题目描述；

· 折扣因子；0.9

· 一个分析1号租车点早晨有10辆车，晚上有0辆车收益的例子：

令收租行为$A_{rent,return}$，则早晨1号租车点有10辆车，晚上有0辆车的收租行为是：

$$A_{rent,return} = \begin{bmatrix} 10 & 0 \\ 11 & 1 \\ \dots & \dots \\ 20 & 10 \end{bmatrix} \tag{1}$$

因为收租事件是相互独立的事件且服从泊松分布，所以要计算某个行为出现的概率直接将两者发生的概率相乘。结合条件概率公式还要与傍晚剩0辆车的概率相除，计算出$P(A_{rent,return}|S^{'}=0)$.

这样子傍晚剩0辆车的收益期望为：

则一天的收益期望可以写为：

$$R(S=10|S^{'}=0) = 10 \begin{bmatrix} \frac{P(A_{rent}=10)P(A_{return}=0)}{P(S^{'}=0)} \\ \frac{P(A_{rent}=11)P(A_{return}=1)}{P(S^{'}=0)} \\ \dots \\ \frac{P(A_{rent}=20)P(A_{return}=10)}{P(S^{'}=0)} \end{bmatrix}^T \tag{2}$$

其中$P(S^{'}=0) = \sum P(A_{rent})P(A_{return})$

加权平均计算：

$$R(S=10) = P(S^{'}=0,1,2,\dots,20)R^T(S=10|S^{'}=0,1,2,\dots,20) \tag{3}$$

将所有状态按照上述方法计算后可以得到两个租车点的奖励：$[R_1(S), R_2(S)]$

代码：

```python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from numpy.core.numeric import _ones_like_dispatcher
import seaborn as sns
from scipy.stats import poisson
```

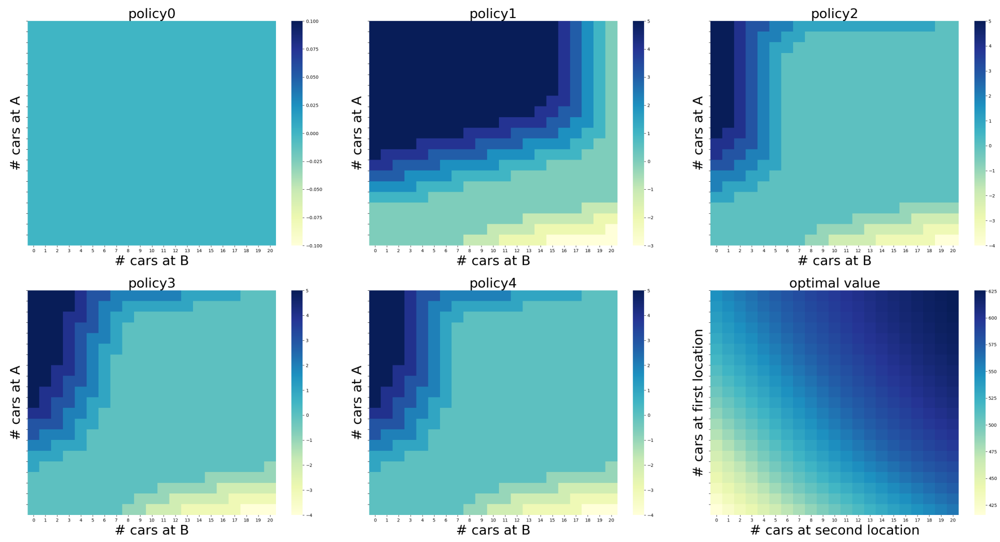
```python

MAX_CARS = 20

MAX_MOVE_CARS = 5

AVG_RENT_A = 3
AVG_RENT_B = 4
AVG_RETURN_A = 3
AVG_RETURN_B = 2

DISCOUNT = 0.9

RENT_BONUS = 10

MOVE_COST = 2

actions = np.arange(-MAX_MOVE_CARS, MAX_MOVE_CARS + 1)

POISSON_UPPER_BOUND = 11

poisson_cahce = dict()

def poisson_probability(n, lam):
    global poisson_cahce
    key = n * 10 + lam
    if key not in poisson_cahce:
        poisson_cahce[key] = poisson.pmf(n,lam)
    return poisson_cahce[key]

# 计算状态价值
def cal_v(state, action, state_value, returned_cars):
    """
    @state：每个地点的车辆数
    @action：移动
    @state_value：状态价值矩阵
    @returned_cars：还车数目
    """
    returns = 0.0
    returns -= MOVE_COST * abs(action)

    NUM_OF_CARS_A = min(state[0] - action, MAX_CARS)
    NUM_OF_CARS_B = min(state[1] + action, MAX_CARS)

```

```
    # 遍历两地全部可能概率下租车数目请求
    for rent_req_A in range(POISSON_UPPER_BOUND):
        for rent_req_B in range(POISSON_UPPER_BOUND):
            prob = poisson_probability(rent_req_A, AVG_RENT_A) *\
                poisson_probability(rent_req_B,AVG_RENT_B)

            num_of_cars_A = NUM_OF_CARS_A
            num_of_cars_B = NUM_OF_CARS_B

            valid_rent_A = min(num_of_cars_A, rent_req_A)
            valid_rent_B = min(num_of_cars_B, rent_req_B)

            reward = (valid_rent_A + valid_rent_B) * RENT_BONUS
            num_of_cars_A -= valid_rent_A
            num_of_cars_B -= valid_rent_B

            # 如果还车的数目为泊松分布的均值
            if returned_cars:
                returned_cars_A = AVG_RETURN_A
                returned_cars_B = AVG_RETURN_B

                num_of_cars_A = min(num_of_cars_A + returned_cars_A, MAX_CARS)
                num_of_cars_B = min(num_of_cars_B + returned_cars_B, MAX_CARS)

                # 策略评估
                returns += prob * (reward + DISCOUNT * state_value[num_of_cars_A,num_of_cars_B])

            # 计算所有泊松概率分布下的还车空间
            else:
                for returned_cars_A in range(POISSON_UPPER_BOUND):
                    for returned_cars_B in range(POISSON_UPPER_BOUND):
                        prob_return = poisson_probability(returned_cars_A, AVG_RETURN_A) * poisson_probability( returned_cars_B, AVG_RETURN_B)
                        num_of_cars_A_ = min(num_of_cars_A + returned_cars_A, MAX_CARS)
```

```python
                        num_of_cars_B_ = min(num_of_cars_B + returned_cars_B, MAX_CARS)
                        # 联合概率
                        prob_ = prob_return * prob
                        returns += prob_ * (reward + DISCOUNT) * state_value[num_of_cars_A_, num_of_cars_B_]
    return returns


def draw(constant_returned_cars = True):
    value = np.zeros((MAX_CARS + 1, MAX_CARS + 1))
    policy = np.zeros(value.shape, dtype=np.int)
    iterations = 0

    # 准备画布，准备多个子图
    _, axes = plt.subplots(2, 3, figsize=(40, 20))
    # 调整子图间距
    plt.subplots_adjust(wspace=0.1, hspace=0.2)
    # 将子图形成1 * 6 的列表
    axes = axes.flatten()
    while True:
        # 使用seaborn的heatmap作图
        fig = sns.heatmap(np.flipud(policy), cmap="YlGnBu", ax=axes[iterations])

        fig.set_ylabel('# cars at A',fontsize = 30)
        fig.set_yticks(list(reversed(range(MAX_CARS + 1))))
        fig.set_xlabel('# cars at B',fontsize = 30)
        fig.set_title('policy{}'.format(iterations), fontsize = 30)

        while True:
            old_value = value.copy()
            for i in range(MAX_CARS + 1):
                for j in range(MAX_CARS + 1):
                    # 更新v(s)
                    new_state_value = cal_v([i,j], policy[i,j], value, constant_returned_cars)

                    value[i,j] = new_state_value
            max_value_change = abs(old_value - value).max()
            print('max value change {}'.format(max_value_change))
```

```python
            if(max_value_change < 1e-4):
                break

        policy_stable = True
        # i,j 为 A、B两地的现有车辆
        for i in range(MAX_CARS + 1):
            for j in range(MAX_CARS + 1):
                old_action= policy[i,j]
                action_returns = []

                for action in actions:
                    if(0 <= action <= i) or (-j <=action <= 0):
                        action_returns.append(cal_v([i,j], action, value, constant_returned_cars))
                    else:
                        action_returns.append(-np.inf)

                new_action = actions[np.argmax(action_returns)]

                policy[i,j] = new_action

                if policy_stable and old_action != new_action:
                    policy_stable = False

        print('policy stable{}'.format(policy_stable))
        if policy_stable:
            fig = sns.heatmap(np.flipud(value), cmap="YlGnBu", ax=axes[-1])
            fig.set_ylabel('# cars at first location', fontsize=30)
            fig.set_yticks(list(reversed(range(MAX_CARS + 1))))
            fig.set_xlabel('# cars at second location', fontsize=30)
            fig.set_title('optimal value', fontsize=30)
            break

        iterations += 1

    plt.savefig('C:/INFO\RL/assignment/2_Jack/fig.png')
    plt.close()

if __name__ == '__main__':
    draw()
```

结果图：



policy0
policy1
policy2
policy3
policy4
optimal value
cars at A
cars at B
cars at first location
cars at second location