



# Disassembling Jack's Car Rental Problem



# Table of Contents

1. Problem Key Points
2. Code – Poisson RV Cache
3. Code – Expected Return
4. Code – Policy Evaluation
5. Code – Policy Improvement
6. Run – Value/Policy Changes
7. Policy Result

# 1. Problem Key Points

	$\lambda_{request}$	$\lambda_{return}$
1 <sup>st</sup> location	3	3
2 <sup>nd</sup> location	4	2

1. Two locations, maximum 20 cars at each
2. A car is rented: \$10 earned (**Reward**)
3. Moving a car overnight to another location: costs \$2 (**Negative Reward**).
4. Max number of cars moved overnight: 5 (**Action**).
5. The number of car requests and returns at each location, per day: **Poisson random variables**

$$\frac{\lambda^n}{n!} e^{-\lambda}$$

6. Discount rate for future returns( $\gamma$ ) = 0.9.
7. The time step = days. (one step in an iteration = a full day)
8. **State**: number of cars at each location at the end of the day
9. **Action**: net number of cars moved between the two locations overnight. (-5~5)

```
MAX_CARS = 20
MAX_MOVE_OF_CARS = 5
```

```
# expected request and returns
RENTAL_REQUEST_FIRST_LOC = 3
RENTAL_REQUEST_SECOND_LOC = 4
RETURNS_FIRST_LOC = 3
RETURNS_SECOND_LOC = 2
```

```
DISCOUNT = 0.9
RENTAL_CREDIT = 10
MOVE_CAR_COST = 2
actions = np.arange(-MAX_MOVE_OF_CARS,
MAX_MOVE_OF_CARS + 1)
```

```
value = np.zeros((MAX_CARS + 1, MAX_CARS + 1))
policy = np.zeros(value.shape, dtype=np.int)
```

## 2. Poisson Random Variable Cache

```
# An up bound for poisson distribution
# If n is greater than this value, then the probability of getting n is truncated to 0
POISSON_UPPER_BOUND = 11

poisson_cache = dict()
def poisson(n, lam):
    global poisson_cache
    key = n * 10 + lam
    if key not in poisson_cache.keys():
        poisson_cache[key] = exp(-lam) * pow(lam, n) / factorial(n)
    return poisson_cache[key]
```

### 3. Expected Return (1/2)

```
def expected_return(state, action, state_value):  
    # initialize total return  
    returns = 0.0  
  
    # cost for moving cars  
    returns -= MOVE_CAR_COST * abs(action)  
  
    # go through all possible rental requests  
    for rental_request_first_loc in range(0, POISSON_UPPER_BOUND):  
        for rental_request_second_loc in range(0, POISSON_UPPER_BOUND):  
            # moving cars  
            num_of_cars_first_loc = int(min(state[0] - action, MAX_CARS))  
            num_of_cars_second_loc = int(min(state[1] + action, MAX_CARS))  
  
            # valid rental requests should be less than actual # of cars  
            real_rental_first_loc = min(num_of_cars_first_loc, rental_request_first_loc)  
            real_rental_second_loc = min(num_of_cars_second_loc, rental_request_second_loc)
```

### 3. Expected Return (2/2)

```
# get credits for renting
```

```
reward = (real_rental_first_loc + real_rental_second_loc) * RENTAL_CREDIT
```

```
num_of_cars_first_loc -= real_rental_first_loc
```

```
num_of_cars_second_loc -= real_rental_second_loc
```

```
# probability for current combination of rental requests
```

```
prob = poisson(rental_request_first_loc, RENTAL_REQUEST_FIRST_LOC) * poisson(rental_request_second_loc,  
RENTAL_REQUEST_SECOND_LOC)
```

```
# get returned cars, those cars can be used for renting tomorrow
```

```
returned_cars_first_loc = RETURNS_FIRST_LOC
```

```
returned_cars_second_loc = RETURNS_SECOND_LOC
```

```
num_of_cars_first_loc = min(num_of_cars_first_loc + returned_cars_first_loc, MAX_CARS)
```

```
num_of_cars_second_loc = min(num_of_cars_second_loc + returned_cars_second_loc, MAX_CARS)
```

```
returns += prob * (reward + DISCOUNT * state_value[num_of_cars_first_loc, num_of_cars_second_loc])
```

```
return returns
```

## 4. Policy Evaluation

```
while True:
    new_value = np.copy(value)
    for i in range(MAX_CARS + 1):
        for j in range(MAX_CARS + 1):
            new_value[i, j] = expected_return([i, j], policy[i, j], new_value, constant_returned_cars)
    value_change = np.abs((new_value - value)).sum()
    print('value change %f' % (value_change))
    value = new_value
    if value_change < 1e-4:
        break
```

# 5. Policy Improvement

```
new_policy = np.copy(policy)
for i in range(MAX_CARS + 1):
    for j in range(MAX_CARS + 1):
        action_returns = []
        for action in actions:
            if (action >= 0 and i >= action) or (action < 0 and j >= abs(action)):
                action_returns.append(expected_return([i, j], action, value, constant_returned_cars))
            else:
                action_returns.append(-float('inf'))
        new_policy[i, j] = actions[np.argmax(action_returns)]
```



## 6. Value/Policy Changes (1/2)

value change 45727.511447  
value change 37274.506706  
value change 28991.916653  
value change 22532.424045  
value change 17731.181203

.  
.

value change 0.000233  
value change 0.000190  
value change 0.000155  
value change 0.000126  
value change 0.000103  
value change 0.000084

policy changed in 318 states



value change 4644.832146  
value change 801.144919  
value change 629.078073  
value change 534.482508  
value change 437.193226

.  
.

value change 0.000272  
value change 0.000221  
value change 0.000180  
value change 0.000146  
value change 0.000119  
value change 0.000097

policy changed in 260 states



...

## 6. Value/Policy Changes (2/2)



value change 17.749357  
value change 13.007169  
value change 10.449136  
value change 8.010925  
value change 5.874832  
.  
.  
value change 0.000264  
value change 0.000215  
value change 0.000175  
value change 0.000142  
value change 0.000116  
value change 0.000094  
policy changed in 10 states



value change 0.435993  
value change 0.360778  
value change 0.255713  
value change 0.167818  
value change 0.105331  
.  
.  
value change 0.000286  
value change 0.000229  
value change 0.000185  
value change 0.000149  
value change 0.000121  
value change 0.000098  
policy changed in 0 states



## 7. Policy Result

