

EXERCISE 3.7:

{ Robot running
a maze }

$\begin{cases} r+1 & \text{escape} \\ & \text{episodic task} \\ 0 & \text{otherwise} \end{cases}$

Maximize

$$G_t = R_{t+1} + R_{t+2} + \dots + R_t \quad (3.7)$$

run \Rightarrow || no improvement why?

The reward is the same regardless of amount of time in the maze. Need to communicate that the faster to escape the better.

EXERCISE 3.8 $\gamma = 0.5$, $R_{n \in [1,5]} = [-1, 2, 6, 3, 2]$ $T=5$ $G_{n \in [0,5]} = ?$

Using discounted
(3.8) return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$G_5 = \sum_{k=0}^{\infty} (0.5)^k R_{6+k} \text{ but } R_{6+k} = 0 \text{ then } G_5 = 0$$

$$G_4 = R_5 + \gamma G_5 = 2 + 0.5(0) = 2$$

$$G_1 = R_2 + \gamma(G_2) = 2 + 0.5(8) = 6$$

$$G_3 = R_4 + \gamma(G_4) = 3 + 0.5(2) = 4$$

$$G_0 = R_1 + \gamma(G_1) = -1 + 0.5(6) = 2$$

$$G_2 = R_3 + \gamma(G_3) = 6 + 0.5(4) = 8$$

$$G_{n \in [0,5]} = [2, 6, 8, 4, 2, 0]$$

EXERCISE 3.9 $\gamma = 0.9$ $R_1 = 2$ $R_{[2,\infty)} = 7$ G_0 & G_1 ?

Using (3.8) $G_1 = \sum_{k=0}^{\infty} (0.9)^k R_{2+k} = \text{infinite series}$
or $7[1 + 0.9 + 0.9^2 + \dots] = 7 \cdot \frac{1}{1-0.9} = 70$

$$G_0 = R_1 + \gamma G_1 = 2 + 0.9(70) = 65$$

$$G_0, G_1 = 65, 70$$

EXERCISE 3.12 Give $V_\pi = f(q_\pi, \pi)$

$$V_\pi = \sum_{a \in A(s)} \pi(a|s) q_\pi(s, a)$$

$\pi(a|s)$ \sim state-action value function

probability of action a given s

Sum through all actions

EXERCISE 3.13 Give $q_\pi = f(V_\pi, p)$

$$q_\pi = \sum_{s' \in S} \sum_{r \in R} p(s', r | s, a) (r + \gamma V_\pi(s'))$$

$\sum_{s' \in S}$ \downarrow Sum all new states

$\sum_{r \in R}$ \downarrow Sum all rewards

$p(s', r | s, a)$ \downarrow probability of new state & reward given state & action

r \downarrow reward

γ \downarrow discount

$V_\pi(s')$ \downarrow value of new state

EXERCISE 3.15

Gridworld $\rightarrow r = +$ goals
 $\rightarrow r = -$ edge
 $\rightarrow r = 0$ otherwise

sgn(r) important? / intervals?

Prove w/ (3.8) $\forall r \neq c \neq$ new values of states

$$\text{find } V_c = V_c(c, \gamma)$$

using (3.8)

$$G_t = \sum_{k=0}^{\infty} \gamma^k (R_{t+k+1})$$

now add c to all Rewards

$$\Rightarrow G'_t = \sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c)$$

$\hookrightarrow R_n$ can be $-, 0, +$

\hookrightarrow can add c such that

R_n is all $+$, all same sign or

$$G'_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \sum_{k=0}^{\infty} c \gamma^k = G_t + \frac{c}{1-\gamma}$$

$$V_c = \frac{c}{1-\gamma}$$

EXERCISE 3.17 Bellman equations for q_π

$$q_\pi(s, a) = f(q_\pi(s', a'))$$

Using result from EXERCISE 3.13 $q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V_\pi(s')] \quad (A)$

$$\& \text{ EXERCISE 3.12 } V_\pi(s) = \sum_{a \in A} \pi(a|s) \cdot q_\pi(s, a) \quad (B)$$

plug in (B) to (A)

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \sum_{a' \in A} \pi(a'|s') \cdot \gamma q_\pi(s', a') \right]$$

EXERCISE 3.25. Give $V_\pi = V_\pi(q_\pi)$

$$V_\pi(s) = \max_{a \in A(s)} q_\pi(s, a)$$

optimal q -factor

EXERCISE 3.26 Give $q_\pi = q_\pi(V_\pi, P)$

$$q_\pi(s, a) = \max_{s' \in S} \sum_{r \in R} P(s', r | s, a) [r + \gamma V_\pi(s')]$$

best value

EXERCISE 3.27 Give $\pi_\pi = \pi_\pi(q_\pi)$

$$\pi_\pi(a|s) = \max_{a \in A} q_\pi(s, a)$$

best state-action value

Exercise 4.7 Write a program for policy iteration & resolve Jack's car rental problem w/ the following changes

- ① Jack's employer at 1st rides ^{but} & lines in 2nd. Shuttle 1 car to 2nd for free
- ② 10 cars limit at each location, 4\$ cost for extra

From Example 4.2 Jack's Car Rental

①
requests $P(n) = \frac{3^n}{n!} e^{-3}$ Poisson

return $P(n) = \frac{3^n}{n!} e^{-3}$

max 20

②
 $P(n) = \frac{4^n}{4!} e^{-4}$

$P(n) = \frac{2^n}{2!} e^{-2}$

max 20

10\$ per car rented

2\$ to move / car

max 5 cars move / night
 $\gamma = 0.9$

Also use Policy Iteration Method.

EXERCISE 9.5 How would policy iteration be defined for action values? Give a complete algorithm for computing q_{π} , analogous to that on page 80 for computing v_{π} . Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.

Policy iteration for action values is essentially trying to find the best policy

1. Initialization

$V(s) \in \mathbb{R}$ & $\pi(s) = A(s)$ arbitrarily $\forall s \in S$; $V(\text{terminal}) = 0$

$Q(s, a)$ arbitrarily $\forall s \in S, a \in A(s)$

Set learning rate & tolerance

while Q not converged

while S not terminal

$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}} Q(s, a)$

Store action, reward & new state

Calculate new $Q(s', a)$ based on $Q(s, a)$ & learning rate

Set s as the s'

return Q

Jack's Capitalism Optimization

Exercise 4.7: Jack's normal day

It can be seen from the figures below that these photos match with the photos found from the book. Both of them stopped after iteration 4 and the corresponding value function using the optimized policy can be seen as well!

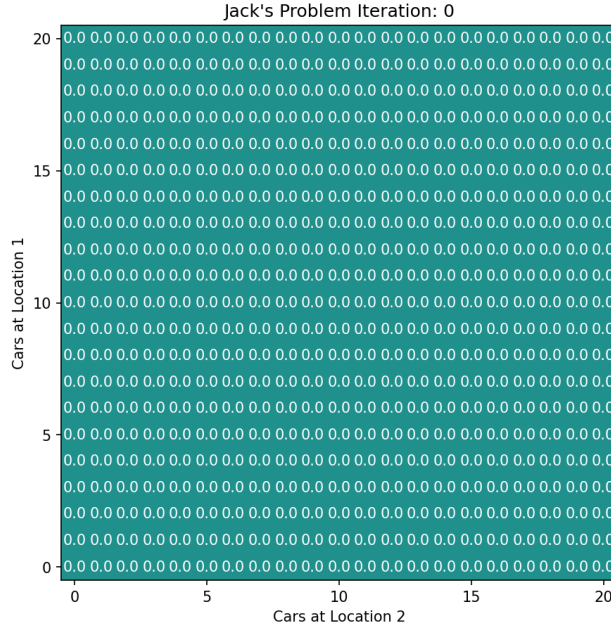


Figure 1: Initialized Policy

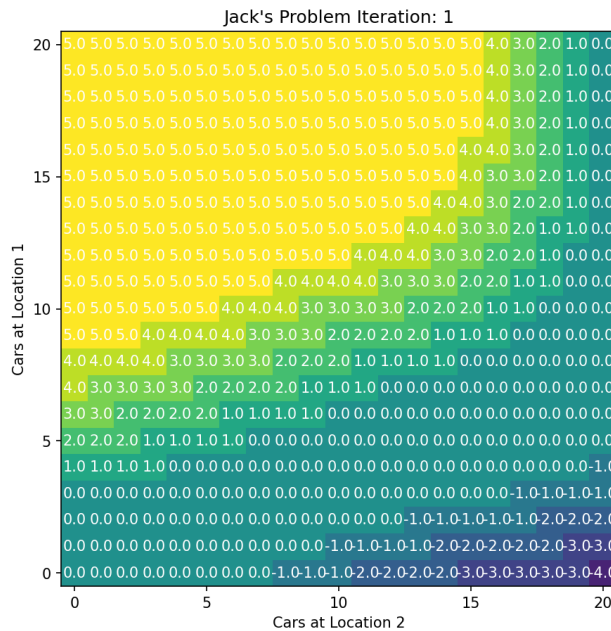


Figure 2: Policy after 1 iteration

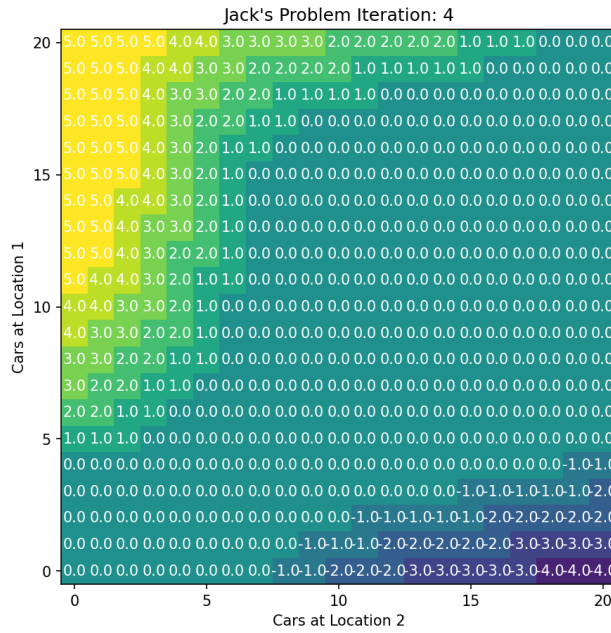


Figure 5: Policy after 4 iteration

Value Function for Jack's Original Problem

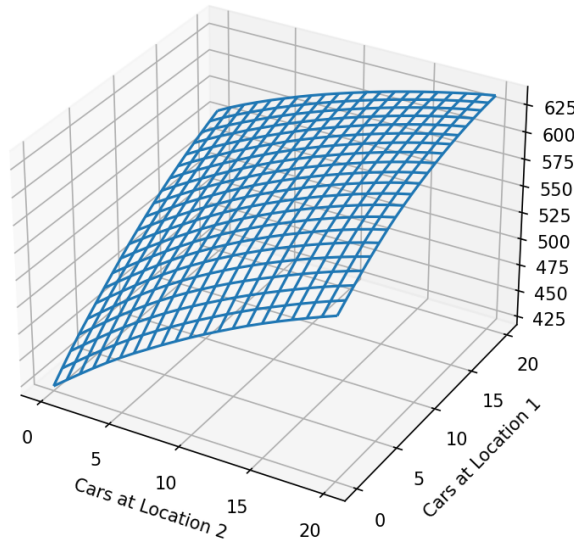


Figure 6: Value with optimized policy

Exercise 4.7: Jack's special day

For the special problem, there was a couple of extra steps that needed to be coded but overall the process is still the same. Starting with Policy Evaluation and then Policy Iteration in order to get a "steady-state" policy. It can be seen that the final policy looks different from the original problem and that the full on 20 doesn't have a full on move 5 anymore. This can be due to the fact that it tries to avoid the extra payment or it can also take advantage of it because of the fixed rate.

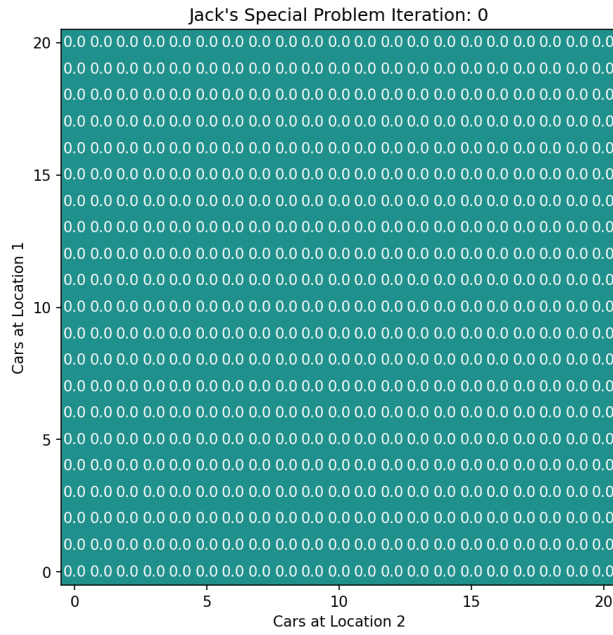


Figure 7: Initialized Policy

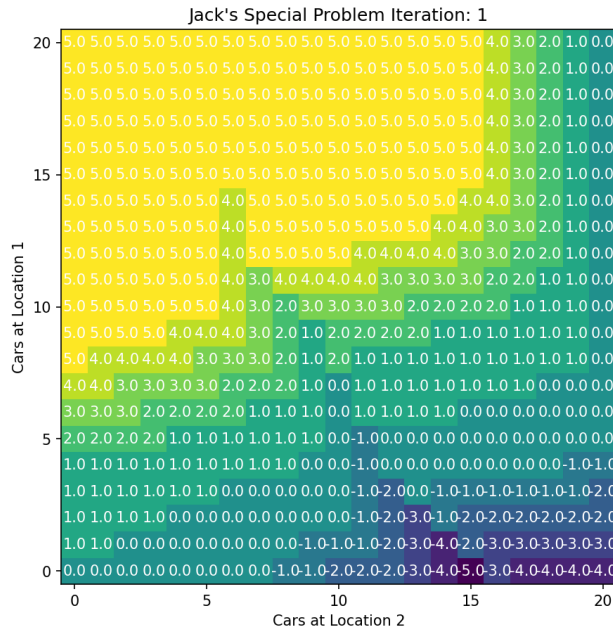


Figure 8: Policy after 1 iteration

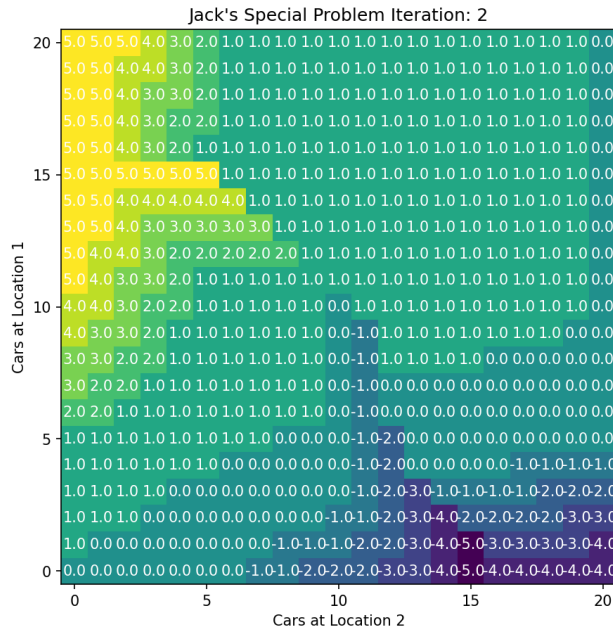


Figure 9: Policy after 2 iteration

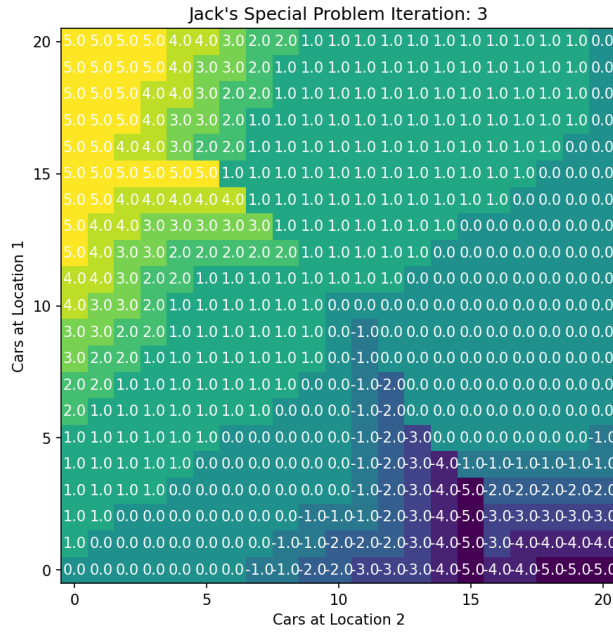


Figure 10: Policy after 3 iteration

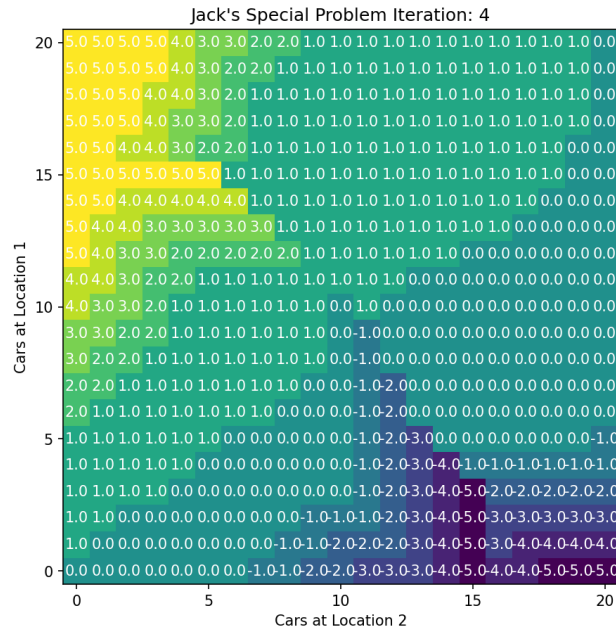


Figure 11: Policy after 4 iteration

Value Function for Jack's Special Problem

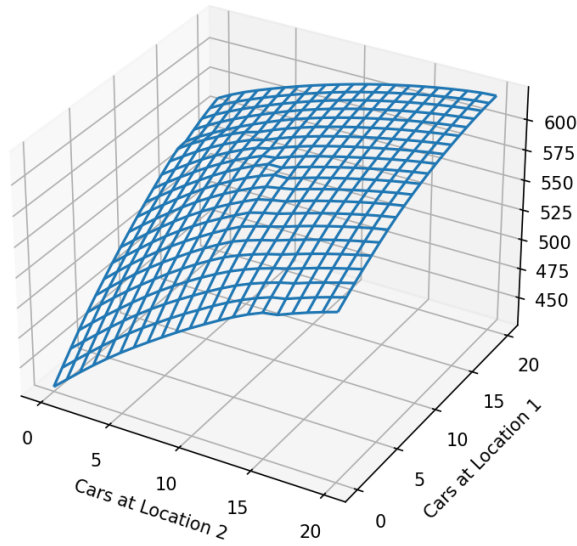


Figure 12: Value with optimized policy

Appendix

Python Code for Original

```
1 '''
2 Justine Serdoncillo
3 IE 5571 - Dynamic Programming
4 HW 2 Exercise 4.7
5 October 9, 2023
6 '''
7
8 """
9 Write a program for policy iteration and re-solve Jacks car
10 rental problem with the following changes. One of Jacks employees at the first location
11 rides a bus home each night and lives near the second location. She is happy to shuttle
12 one car to the second location for free. Each additional car still costs $2, as do all cars
13 moved in the other direction. In addition, Jack has limited parking space at each location.
14 If more than 10 cars are kept overnight at a location (after any moving of cars), then an
15 additional cost of $4 must be incurred to use a second parking lot (independent of how
16 many cars are kept there). These sorts of nonlinearities and arbitrary dynamics often
17 occur in real problems and cannot easily be handled by optimization methods other than
18 dynamic programming. To check your program, first replicate the results given for the
19 original problem.
20 """
21
22 import numpy as np
23 import matplotlib.pyplot as plt
24 import random
25 import math
26
27 # %% Use the Poisson Equation for the probability
28 def poisson(lam, n):
29     return lam**n * math.exp(-lam) / math.factorial(n)
30
31 # Return a list of all possible actions
32 def possibleActions(state):
33     # Get the state of the cars
34     firLoc = state[0]
35     secLoc = state[1]
36     # breakpoint()
37     # list all possible & impossible actions
38     allActions = [x for x in range(-maxMove, maxMove+1)]
39     possibleActions = []
40
41     for action in allActions:
42         newFir = firLoc - action
43         newSec = secLoc + action
44         if newFir > maxCar or newFir < 0 or newSec > maxCar or newSec < 0:
45             pass
46         else:
47             possibleActions.append(action)
48
49     return possibleActions
50
51 # %% Evaluate a state and action based on a policy
52 def evalVal(state, action):
53     # Initialize
54     value = 0
55
56     # Get the new state of the cars and compute the cost
57     firLoc = state[0] - action
58     secLoc = state[1] + action
59     cost = costCar * abs(action)
60
61     # Summation over all the possible new states
62     ps1req, ps2req, ps1ret, ps2ret = 1, 1, 1, 1
63     for reqs1 in range(firLoc+1):
64         p1 = poisson(3, reqs1) if reqs1 != firLoc else ps1req
65         ps1req -= p1
66         g = maxCar + reqs1 - firLoc
67         r1 = creditCar * p1
```

```

68         for reqs2 in range(secLoc+1):
69             p2 = poisson(4, reqs2) if reqs2 != secLoc else ps2req
70             ps2req -= p2
71             h = maxCar + reqs2 - secLoc
72             r2 = creditCar * p2
73             for rets1 in range(g + 1):
74                 p3 = poisson(3, rets1) if rets1 != g else ps1ret
75                 ps1ret -= p3
76                 for rets2 in range(h + 1):
77                     p4 = poisson(2, rets2) if rets2 != h else ps2ret
78                     ps2ret -= p4
79
80                 # Sum all over the possible rewards
81                 pTot = p1 * p2 * p3 * p4
82                 reward = r1 + r2 - cost + gamma*values[firLoc-reqs1+rets1, secLoc-reqs2+
rets2]
83                 value += pTot * reward
84
85             #breakpoint()
86             return value
87
88 # %% Evaluate the Value of the policy
89 def policyEvaluation():
90     print("Policy Evaluation")
91     print("=====")
92     ite = 0
93     maxIte = 10
94     while ite < maxIte:
95         ite += 1
96         delta = 0
97         #breakpoint()
98         for s in states:
99             v = values[s]
100             values[s] = evalVal(s, policy[s])
101             #breakpoint()
102             delta = max(delta, abs(v - values[s]))
103         print(f"Current Delta at Iteration {ite}: {delta}")
104         if delta < theta:
105             break
106
107 # %% Actual Problem statement
108 # Problem parameters
109 maxCar = 20
110 maxMove = 5
111 creditCar = 10
112 costCar = 2
113
114 # Learning Parameters
115 gamma = 0.9
116 theta = 1E-2
117 maxIte = 5
118 ite = 0
119
120 # list comprehension for the states
121 states = [(x,y) for x in range(maxCar+1) for y in range(maxCar+1)]
122 policy = np.zeros((maxCar+1, maxCar+1), dtype=np.int8)
123 values = np.zeros((maxCar+1, maxCar+1))
124 #policy[10,0] = 5
125
126 fig, ax = plt.subplots(figsize=(8,6), dpi=150)
127 ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
128 ax.set_title(f"Jack's Problem Iteration: {ite}")
129 fig.tight_layout()
130 ax.set_xlabel("Cars at Location 2")
131 ax.set_ylabel("Cars at Location 1")
132 ax.set_xticks(np.arange(0, maxCar+1, 5))
133 ax.set_yticks(np.arange(0, maxCar+1, 5))
134 for yy in range(policy.shape[1]):
135     for xx in range(policy.shape[0]):
136         text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")
137
138 while ite < maxIte:
139     ite += 1

```

```

139 print(f"Iteration: {ite}")
140 print( "~~~~~")
141
142 # Policy Evaluation
143 print("Policy Evaluation")
144 print("=====")
145 ite = 0
146 maxIte = 50
147 while ite < maxIte:
148     ite += 1
149     delta = 0
150     #breakpoint()
151     for s in states:
152         v = values[s]
153         values[s] = evalVal(s, policy[s])
154         #breakpoint()
155         delta = max(delta, abs(v - values[s]))
156     print(f"Current Delta at Iteration {ite}: {delta}")
157     if delta < theta:
158         break
159
160 # Policy Improvement
161 policy_stable = True
162 for s in states:
163     # copy policy to become old policy
164     old = policy[s].copy()
165
166     # create values dictionary based on chosen action
167     vvalues = {a: evalVal(s,a) for a in possibleActions(s)}
168     # random choose an action from the actions that map to the max values
169     bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.
values()))]
170     policy[s] = np.random.choice(bestActions)
171
172     # compare if old action is same as current action
173     if old != policy[s]:
174         policy_stable = False
175 if policy_stable:
176     break
177
178 # Visualize current policy
179 fig, ax = plt.subplots(figsize=(8,6), dpi=150)
180 ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
181 ax.set_title(f"Jack's Problem Iteration: {ite}")
182 fig.tight_layout()
183 ax.set_xlabel("Cars at Location 1")
184 ax.set_ylabel("Cars at Location 2")
185 ax.set_xticks(np.arange(0, maxCar+1, 5))
186 ax.set_yticks(np.arange(0, maxCar+1, 5))
187 for yy in range(policy.shape[1]):
188     for xx in range(policy.shape[0]):
189         text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")
190
191
192 # Print the optimal value function
193 fig = plt.figure(figsize=(8,6), dpi=150)
194 ax = plt.axes(projection='3d')
195 X, Y = np.meshgrid(range(maxCar+1), range(maxCar+1))
196 ax.plot_wireframe(X, Y, values, rstride=1, cstride=1)
197 ax.set_xlabel("Cars at Location 2")
198 ax.set_ylabel("Cars at Location 1")
199 ax.set_zlabel("V", rotation=0)
200 ax.set_title("Value Function for Jack's Original Problem")
201 ax.set_xticks(np.arange(0, maxCar+1, 5))
202 ax.set_yticks(np.arange(0, maxCar+1, 5))
203 fig.savefig("values.png")

```

Python Code for Special

```
1 '''
2 Justine Serdoncillo
3 IE 5571 - Dynamic Programming
4 HW 2 Exercise 4.7
5 October 9, 2023
6 '''
7
8 """
9 Write a program for policy iteration and re-solve Jacks car
10 rental problem with the following changes. One of Jacks employees at the first location
11 rides a bus home each night and lives near the second location. She is happy to shuttle
12 one car to the second location for free. Each additional car still costs $2, as do all cars
13 moved in the other direction. In addition, Jack has limited parking space at each location.
14 If more than 10 cars are kept overnight at a location (after any moving of cars), then an
15 additional cost of $4 must be incurred to use a second parking lot (independent of how
16 many cars are kept there). These sorts of nonlinearities and arbitrary dynamics often
17 occur in real problems and cannot easily be handled by optimization methods other than
18 dynamic programming. To check your program, first replicate the results given for the
19 original problem.
20 """
21
22 import numpy as np
23 import matplotlib.pyplot as plt
24 import random
25 import math
26
27 # %% Use the Poisson Equation for the probability
28 def poisson(lam, n):
29     return lam**n * math.exp(-lam) / math.factorial(n)
30
31 # Return a list of all possible actions
32 def possibleActions(state):
33     # Get the state of the cars
34     firLoc = state[0]
35     secLoc = state[1]
36     # breakpoint()
37     # list all possible & impossible actions
38     allActions = [x for x in range(-maxMove, maxMove+1)]
39     possibleActions = []
40
41     for action in allActions:
42         newFir = firLoc - action
43         newSec = secLoc + action
44         if newFir > maxCar or newFir < 0 or newSec > maxCar or newSec < 0:
45             pass
46         else:
47             possibleActions.append(action)
48
49     return possibleActions
50
51 # %% Evaluate a state and action based on a policy
52 def evalVal(state, action):
53     # Initialize
54     value = 0
55
56     # Get the new state of the cars and compute the cost
57     firLoc = state[0] - action
58     secLoc = state[1] + action
59     if action > 0:
60         action -= 1
61         cost = costCar * abs(action)
62     else:
63         if firLoc > limCar or secLoc > limCar:
64             sad = 1
65         else:
66             sad = 0
67         cost = costCar * abs(action) + overCost * sad
68
69     # Summation over all the possible new states
70     ps1req, ps2req, ps1ret, ps2ret = 1, 1, 1, 1
```



```

71     for reqs1 in range(firLoc+1):
72         p1 = poisson(3, reqs1) if reqs1 != firLoc else ps1req
73         ps1req -= p1
74         g = maxCar + reqs1 - firLoc
75         r1 = creditCar * p1
76         for reqs2 in range(secLoc+1):
77             p2 = poisson(4, reqs2) if reqs2 != secLoc else ps2req
78             ps2req -= p2
79             h = maxCar + reqs2 - secLoc
80             r2 = creditCar * p2
81             for rets1 in range(g + 1):
82                 p3 = poisson(3, rets1) if rets1 != g else ps1ret
83                 ps1ret -= p3
84                 for rets2 in range(h + 1):
85                     p4 = poisson(2, rets2) if rets2 != h else ps2ret
86                     ps2ret -= p4
87
88                     # Sum all over the possible rewards
89                     pTot = p1 * p2 * p3 * p4
90                     reward = r1 + r2 - cost + gamma*values[firLoc-reqs1+rets1, secLoc-reqs2+
rets2]
91                     value += pTot * reward
92     return value
93
94 # %% Evaluate the Value of the policy
95 def policyEvaluation():
96     print("Policy Evaluation")
97     print("=====")
98     ite = 0
99     maxIte = 10
100    while ite < maxIte:
101        ite += 1
102        delta = 0
103        #breakpoint()
104        for s in states:
105            v = values[s]
106            values[s] = evalVal(s, policy[s])
107            #breakpoint()
108            delta = max(delta, abs(v - values[s]))
109        print(f"Current Delta at Iteration {ite}: {delta}")
110        if delta < theta:
111            break
112
113 # %% Actual Problem statement
114 # Problem parameters
115 maxCar = 20
116 maxMove = 5
117 creditCar = 10
118 costCar = 2
119
120 # Extra Rules
121 limCar = 10
122 overCost = 4
123
124 # Learning Parameters
125 gamma = 0.9
126 theta = 1E-2
127 maxIte = 0
128 ite = 0
129 stay = True
130
131 # list comprehension for the states
132 states = [(x,y) for x in range(maxCar+1) for y in range(maxCar+1)]
133 policy = np.zeros((maxCar+1, maxCar+1), dtype=np.int8)
134 values = np.zeros((maxCar+1, maxCar+1))
135 #policy[10,0] = 5
136
137 fig, ax = plt.subplots(figsize=(8,6), dpi=150)
138 ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
139 ax.set_title(f"Jack's Problem Iteration: {ite}")
140 fig.tight_layout()
141 ax.set_xlabel("Cars at Location 2")

```

```

142 ax.set_ylabel("Cars at Location 1")
143 ax.set_xticks(np.arange(0, maxCar+1, 5))
144 ax.set_yticks(np.arange(0, maxCar+1, 5))
145 for yy in range(policy.shape[1]):
146     for xx in range(policy.shape[0]):
147         text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")
148
149 while ite < maxIte:
150     ite += 1
151     print(f"Iteration: {ite}")
152     print("~~~~~")
153
154     # Policy Evaluation
155     print("Policy Evaluation")
156     print("=====")
157     ite = 0
158     maxIte = 50
159     while ite < maxIte:
160         ite += 1
161         delta = 0
162         #breakpoint()
163         for s in states:
164             v = values[s]
165             values[s] = evalVal(s, policy[s])
166             #breakpoint()
167             delta = max(delta, abs(v - values[s]))
168         print(f"Current Delta at Iteration {ite}: {delta}")
169         if delta < theta:
170             break
171
172     # Policy Improvement
173     policy_stable = True
174     for s in states:
175         # copy policy to become old policy
176         old = policy[s].copy()
177
178         # create values dictionary based on chosen action
179         vvalues = {a: evalVal(s,a) for a in possibleActions(s)}
180         # random choose an action from the actions that map to the max values
181         bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.
182 values()))]
183         policy[s] = np.random.choice(bestActions)
184
185         # compare if old action is same as current action
186         if old != policy[s]:
187             policy_stable = False
188     if policy_stable:
189         break
190
191     # Visualize current policy
192     fig, ax = plt.subplots(figsize=(8,6), dpi=150)
193     ax.imshow(policy, interpolation='lower', vmin=-maxMove, vmax=maxMove)
194     ax.set_title(f"Jack's Special Problem Iteration: {ite}")
195     fig.tight_layout()
196     ax.set_xlabel("Cars at Location 1")
197     ax.set_ylabel("Cars at Location 2")
198     ax.set_xticks(np.arange(0, maxCar+1, 5))
199     ax.set_yticks(np.arange(0, maxCar+1, 5))
200     for yy in range(policy.shape[1]):
201         for xx in range(policy.shape[0]):
202             text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")
203
204 # Print the optimal value function
205 fig = plt.figure(figsize=(8,6), dpi=150)
206 ax = plt.axes(projection='3d')
207 X, Y = np.meshgrid(range(maxCar+1), range(maxCar+1))
208 ax.plot_wireframe(X, Y, values, rstride=1, cstride=1)
209 ax.set_xlabel("Cars at Location 2")
210 ax.set_ylabel("Cars at Location 1")
211 ax.set_zlabel("V", rotation=0)
212 ax.set_title("Value Function for Jack's Original Problem")

```

```
213 ax.set_xticks(np.arange(0, maxCar+1, 5))
214 ax.set_yticks(np.arange(0, maxCar+1, 5))
215 fig.savefig("values.png")
```