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!
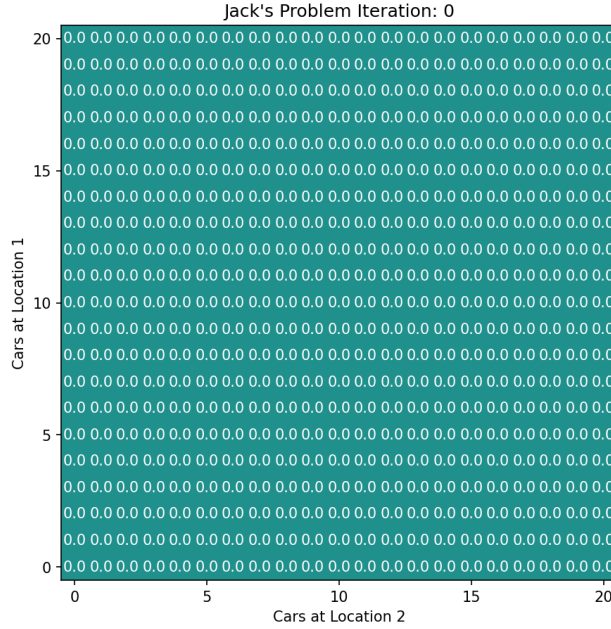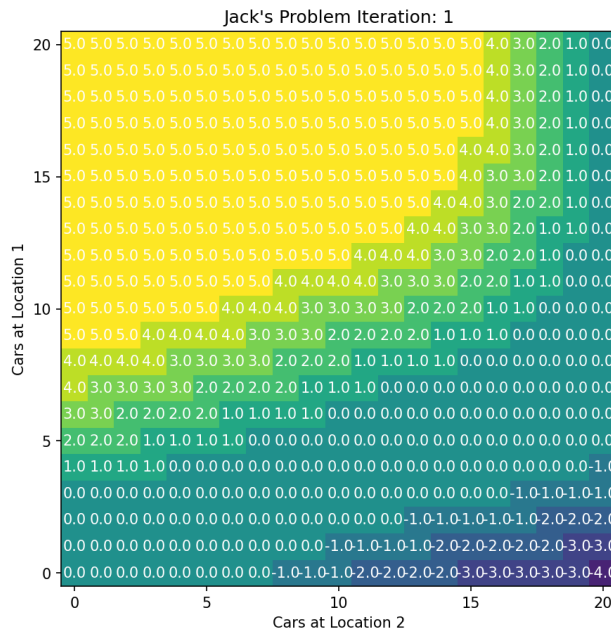

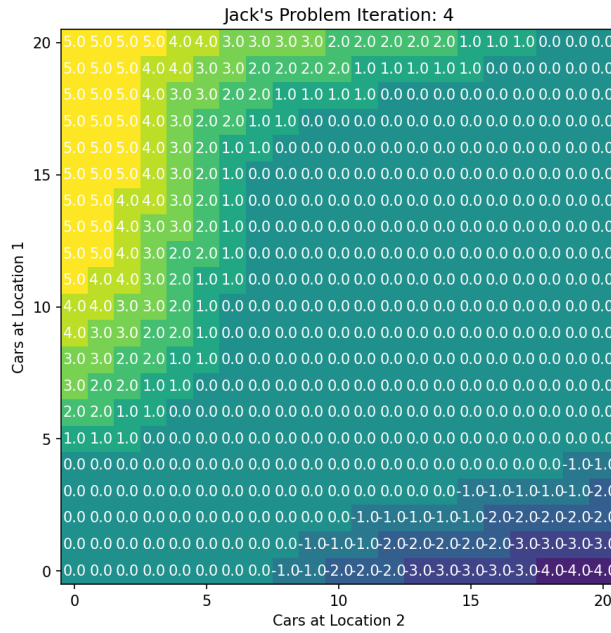Figure 1: Initialized Policy

Figure 2: Policy after 1 iteration

Figure 3: Policy after 2 iteration

Figure 4: Policy after 3 iteration
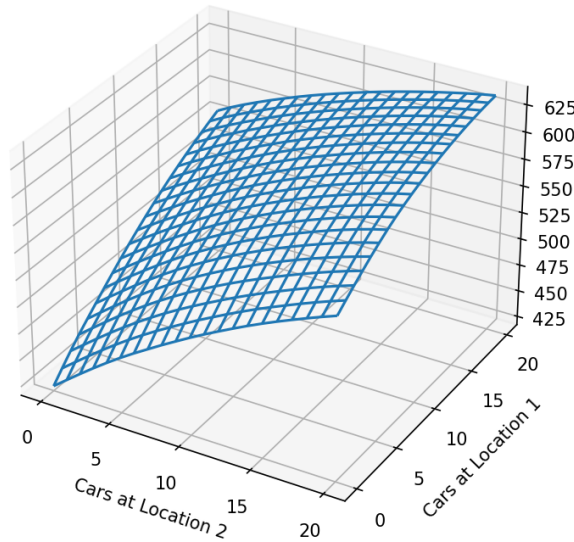
Figure 5: Policy after 4 iteration

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.
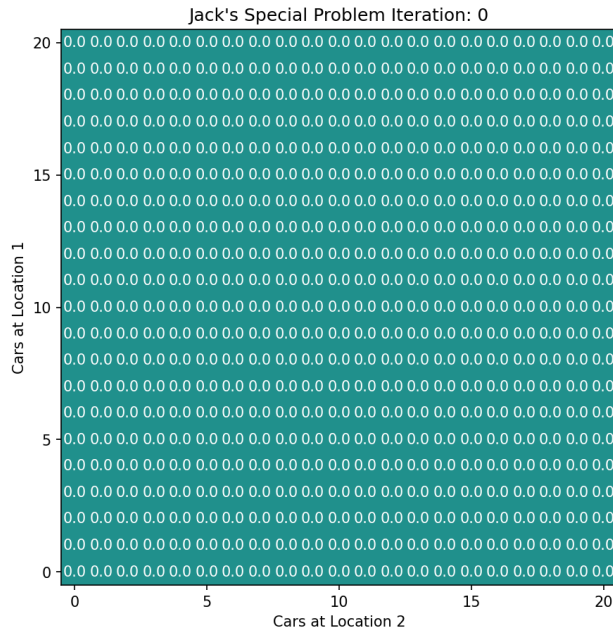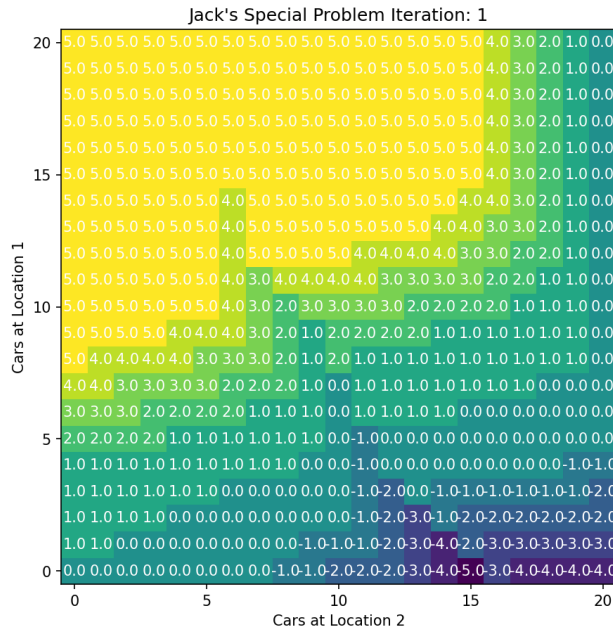
Figure 7: Initialized Policy

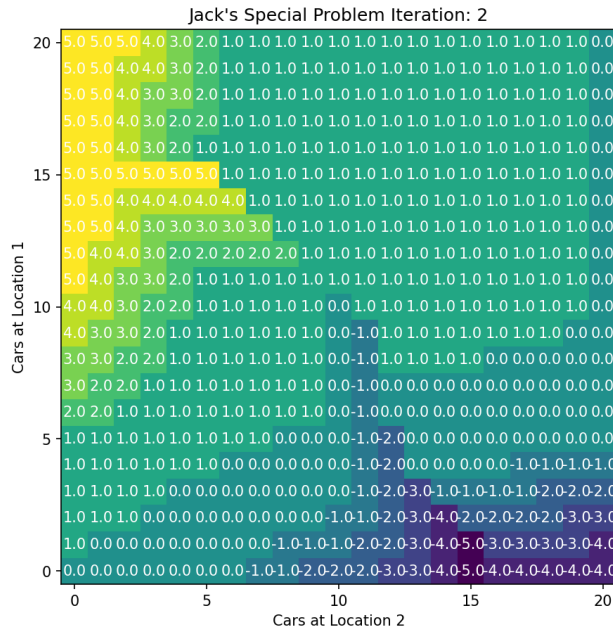Figure 8: Policy after 1 iteration

4

Figure 9: Policy after 2 iteration

Figure 10: Policy after 3 iteration

Figure 11: Policy after 4 iteration
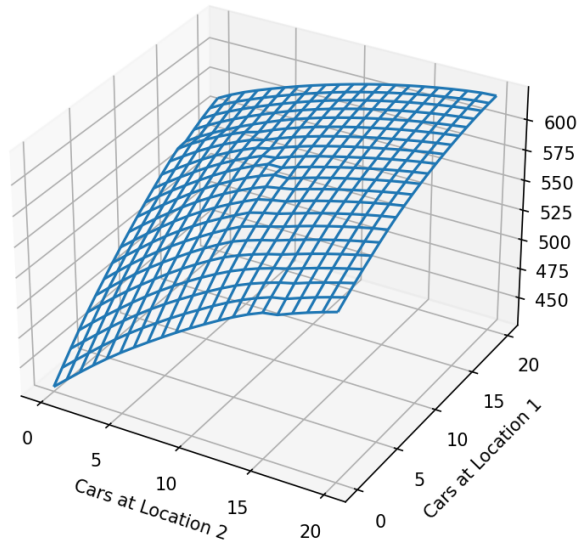

Figure 12: Value with optimized policy

Appendix

Python Code for Original

```python
'''
Justine Serdoncillo
IE 5571 - Dynamic Programming
HW 2 Exercise 4.7
October 9, 2023
'''

"""
Write a program for policy iteration and re-solve  J a c k s  car
rental problem with the following changes. One of  J a c k s  employees at the first location
rides a bus home each night and lives near the second location. She is happy to shuttle
one car to the second location for free. Each additional car still costs $2, as do all cars
moved in the other direction. In addition, Jack has limited parking space at each location.
If more than 10 cars are kept overnight at a location (after any moving of cars), then an
additional cost of $4 must be incurred to use a second parking lot (independent of how
many cars are kept there). These sorts of nonlinearities and arbitrary dynamics often
occur in real problems and cannot easily be handled by optimization methods other than
dynamic programming. To check your program, first replicate the results given for the
original problem.
"""

import numpy as np
import matplotlib.pyplot as plt
import random
import math

# %% Use the Poisson Equation for the probability
def poisson(lam, n):
    return lam**n * math.exp(-lam) / math.factorial(n)

# Return a list of all possible actions
def possibleActions(state):
    # Get the state of the cars
    firLoc = state[0]
    secLoc = state[1]
    #breakpoint()
    # list all possible & impossible actions
    allActions = [x for x in range(-maxMove, maxMove+1)]
    possibleActions = []

    for action in allActions:
        newFir = firLoc - action
        newSec = secLoc + action
        if  newFir > maxCar or newFir < 0 or newSec > maxCar or newSec < 0:
            pass
        else:
            possibleActions.append(action)

    return possibleActions

# %% Evaluate a state and action based on a policy
def evalVal(state, action):
    # Initialize
    value = 0

    # Get the new state of the cars and compute the cost
    firLoc = state[0] - action
    secLoc = state[1] + action
    cost = costCar * abs(action)

    # Summation over all the possible new states
    ps1req, ps2req, ps1ret, ps2ret = 1, 1, 1, 1
    for reqs1 in range(firLoc+1):
        p1 = poisson(3, reqs1) if reqs1 != firLoc else ps1req
        ps1req -= p1
        g = maxCar + reqs1 - firLoc
        r1 = creditCar * p1
```

```python
        for reqs2 in range(secLoc+1):
            p2 = poisson(4, reqs2) if reqs2 != secLoc else ps2req
            ps2req -= p2
            h = maxCar + reqs2 - secLoc
            r2 = creditCar * p2
            for rets1 in range(g + 1):
                p3 = poisson(3, rets1) if rets1 != g else ps1ret
                ps1ret -= p3
                for rets2 in range(h + 1):
                    p4 = poisson(2, rets2) if rets2 != h else ps2ret
                    ps2ret -= p4

                    # Sum all over the possible rewards
                    pTot = p1 * p2 * p3 * p4
                    reward = r1 + r2 - cost + gamma*values[firLoc-reqs1+rets1, secLoc-reqs2+
    rets2]
                    value += pTot * reward
    #breakpoint()
    return value

# %% Evaluate the Value of the policy
def policyEvaluation():
    print("Policy Evaluation")
    print("=================")
    ite = 0
    maxIte = 10
    while ite < maxIte:
        ite += 1
        delta = 0
        #breakpoint()
        for s in states:
            v = values[s]
            values[s] = evalVal(s, policy[s])
            #breakpoint()
            delta = max(delta, abs(v - values[s]))
        print(f"Current Delta at Iteration {ite}: {delta}")
        if delta < theta:
            break

# %% Actual Problem statement
# Problem parameters
maxCar = 20
maxMove = 5
creditCar = 10
costCar = 2

# Learning Parameters
gamma = 0.9
theta = 1E-2
maxIte = 5
ite = 0

# list comprehension for the states
states = [(x,y) for x in range(maxCar+1) for y in range(maxCar+1)]
policy = np.zeros((maxCar+1, maxCar+1), dtype=np.int8)
values = np.zeros((maxCar+1, maxCar+1))
#policy[10,0] = 5

fig, ax = plt.subplots(figsize=(8,6), dpi=150)
ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
ax.set_title(f"Jack's Problem Iteration: {ite}")
fig.tight_layout()
ax.set_xlabel("Cars at Location 2")
ax.set_ylabel("Cars at Location 1")
ax.set_xticks(np.arange(0, maxCar+1, 5))
ax.set_yticks(np.arange(0, maxCar+1, 5))
for yy in range(policy.shape[1]):
    for xx in range(policy.shape[0]):
        text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")

while ite < maxIte:
    ite += 1
```

```python
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

```python
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  J a c k s  car
10 rental problem with the following changes. One of  J a c k s  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
```

```python
     for reqs1 in range(firLoc+1):
         p1 = poisson(3, reqs1) if reqs1 != firLoc else ps1req
         ps1req -= p1
         g = maxCar + reqs1 - firLoc
         r1 = creditCar * p1
         for reqs2 in range(secLoc+1):
             p2 = poisson(4, reqs2) if reqs2 != secLoc else ps2req
             ps2req -= p2
             h = maxCar + reqs2 - secLoc
             r2 = creditCar * p2
             for rets1 in range(g + 1):
                 p3 = poisson(3, rets1) if rets1 != g else ps1ret
                 ps1ret -= p3
                 for rets2 in range(h + 1):
                     p4 = poisson(2, rets2) if rets2 != h else ps2ret
                     ps2ret -= p4

                     # Sum all over the possible rewards
                     pTot = p1 * p2 * p3 * p4
                     reward = r1 + r2 - cost + gamma*values[firLoc-reqs1+rets1, secLoc-reqs2+
     rets2]
                     value += pTot * reward
     return value

# %% Evaluate the Value of the policy
def policyEvaluation():
    print("Policy Evaluation")
    print("=================")
    ite = 0
    maxIte = 10
    while ite < maxIte:
        ite += 1
        delta = 0
        #breakpoint()
        for s in states:
            v = values[s]
            values[s] = evalVal(s, policy[s])
            #breakpoint()
            delta = max(delta, abs(v - values[s]))
        print(f"Current Delta at Iteration {ite}: {delta}")
        if delta < theta:
            break

# %% Actual Problem statement
# Problem parameters
maxCar = 20
maxMove = 5
creditCar = 10
costCar = 2

# Extra Rules
limCar = 10
overCost = 4

# Learning Parameters
gamma = 0.9
theta = 1E-2
maxIte = 0
ite = 0
stay = True

# list comprehension for the states
states = [(x,y) for x in range(maxCar+1) for y in range(maxCar+1)]
policy = np.zeros((maxCar+1, maxCar+1), dtype=np.int8)
values = np.zeros((maxCar+1, maxCar+1))
#policy[10,0] = 5

fig, ax = plt.subplots(figsize=(8,6), dpi=150)
ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
ax.set_title(f"Jack's Problem Iteration: {ite}")
fig.tight_layout()
ax.set_xlabel("Cars at Location 2")
```

```python
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.
    values()))]
182            policy[s] = np.random.choice(bestActions)
183
184            # compare if old action is same as current action
185            if old != policy[s]:
186                policy_stable = False
187        if policy_stable:
188            break
189
190        # Visualize current policy
191        fig, ax = plt.subplots(figsize=(8,6), dpi=150)
192        ax.imshow(policy, origin="lower", interpolation='none', vmin=-maxMove, vmax=maxMove)
193        ax.set_title(f"Jack's Special Problem Iteration: {ite}")
194        fig.tight_layout()
195        ax.set_xlabel("Cars at Location 1")
196        ax.set_ylabel("Cars at Location 2")
197        ax.set_xticks(np.arange(0, maxCar+1, 5))
198        ax.set_yticks(np.arange(0, maxCar+1, 5))
199        for yy in range(policy.shape[1]):
200            for xx in range(policy.shape[0]):
201                text = ax.text(xx, yy, policy[yy, xx], ha="center", va="center", color="w")
202
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")
```