Homework 5

General Comments:

Problem 1:The graph of problem 1 is submitted in a separate pdf file: **problem1_graph.pdf.** The code of problem 1 with explanation is in this file.

Problem 2: I complete this question using my python code. You can simply run the code in anaconda 3 spyder 3.6. The default number of iterations is 40, discount factor is 0.8. The code is submitted called value_iteration.py

Problem 3: This question is also submitted in a separate pdf file: **problem3.pdf**
I calculate this question by hand and I reuse part of the code in problem 2 to help me do the calculations.

Problem 4: problem 4 is in this pdf file.

Problem 1

Below is the python code for my problem1. The code basically stores the graph into some matrixes and will be used in the future. These lines of code are simple and I have we commented. As a result, I will not explain problem 1 in detail

```
'''
 reward matrix
 format is as below:
                    drive      not drive
      top             2          3
     rolling          0          1
     bottom             0          1

'''


reward_matrix = [[0 for x in range(2)] for y in range(3)]
reward_matrix[0][0] = 2
reward_matrix[0][1] = 3
reward_matrix[1][0] = 0
reward_matrix[1][1] = 1
reward_matrix[2][0] = 0
reward_matrix[2][1] = 1

##print(reward_matrix)


## transition matrix (Drive)

'''
             top       rolling    bottom
top           0.9        0.1        0
rolling       0.3        0.6        0.1
bottom        0.6         0         0.4



'''
drive_matrix = [[0 for x in range(3)] for y in range(3)]

drive_matrix[0][0] =  0.9
drive_matrix[0][1] =  0.1
```

drive_matrix[0][2] = 0

drive_matrix[1][0] = 0.3
drive_matrix[1][1] = 0.6
drive_matrix[1][2] = 0.1

drive_matrix[2][0] = 0.6
drive_matrix[2][1] = 0
drive_matrix[2][2] = 0.4

#3 not drive matrix
'''
            top   rolling   bottom
top   0.7   0.3   0
rolling   0   0   1
bottom   0   0   1

'''

not_drive_matrix = [[0 for x in range(3)] for y in range(3)]

not_drive_matrix[0][0] = 0.7
not_drive_matrix[0][1] = 0.3
not_drive_matrix[0][2] = 0

not_drive_matrix[1][0] = 0
not_drive_matrix[1][1] = 0
not_drive_matrix[1][2] = 1

not_drive_matrix[2][0] = 0
not_drive_matrix[2][1] = 0
not_drive_matrix[2][2] = 1

Problem 2

I write python code to solve this problem using the algorithm of value iteration.

The file name is value_iteration.py

```
## Start value iteration process
discount_factor = 0.8

# number of iterations
n = 40
## top rolling bottom (initial value)

v = [0,0,0]
v2 =[0,0,0]
ideal_policy =[0,0,0]

## state
for k in range(n):
    for i in range(3):
        ## apply the value iteration equation

action_drive = reward_matrix[i][0] + discount_factor*( v[0]*drive_matrix[i][0]  +v[1] *
drive_matrix[i][1] +v[2]* drive_matrix[i][2] )

action_not_drive = reward_matrix[i][1] + discount_factor*( v[0]*not_drive_matrix[i][0]  +  v[1] *
not_drive_matrix[i][1] +v[2]* not_drive_matrix[i][2] )
     ## max function
   v2[i] = max(action_drive,action_not_drive)

        #record the policy
        if action_drive  == max(action_drive,action_not_drive):
            ideal_policy[i] = "drive"
        else:
            ideal_policy[i] = "not drive"

    v=v2.copy()
# output the value
     print(ideal_policy)
      print(v2)
   print("number of iteration ",k)
```

I basically used the formula in the lecture slides to do the value iterations The results are stored in the array v2. My the default number of iterations of my code is 40. You can modify the number of iterations by modify the variable n.
You can simply run this code in anaconda 3, Spyder 3.6.

Below is the last few lines of output when n is set to 40.

['not drive', 'not drive', 'drive']
[10.635842964339389, 7.003694587214196, 7.50601297979534]
['not drive', 'not drive', 'drive']
[10.636958760961466, 7.004810383836272, 7.507128776417415]
['not drive', 'not drive', 'drive']
[10.637851398259127, 7.005703021133932, 7.508021413715077]
['not drive', 'not drive', 'drive']
[10.638565508097255, 7.0064171309720615, 7.508735523553206]
['not drive', 'not drive', 'drive']
[10.639136795967758, 7.006988418842565, 7.509306811423709]
['not drive', 'not drive', 'drive']
[10.63959382626416, 7.0074454491389675, 7.509763841720112]
['not drive', 'not drive', 'drive']
[10.639959450501284, 7.007811073376089, 7.510129465957232]
['not drive', 'not drive', 'drive']
[10.640251949890981, 7.008103572765786, 7.51042196534693]
number of literation 39

It is obvious that the value converges. In my code, the first index stands for TOP, second index stands for Rolling, third index stands for Bottom.

As a result, the optimal policy and value is concluded in the following chart:

	Top	**Rolling**	**Bottom**
Policy	**Not Drive**	**Not Drive**	**Drive**
value	**10.64**	**7.01**	**7.51**

Problem 4
I will modify my value iteration code to answer this question.

-- Changing discount factor

I set the discount factor to 0.4 in my value iteration code.
Below is the last few lines of code when iteration is set to 40.

['not drive', 'not drive', 'not drive']
[4.444444444444437, 1.666666666666659, 1.666666666666659]
['not drive', 'not drive', 'not drive']
[4.444444444444441, 1.6666666666666636, 1.6666666666666636]
['not drive', 'not drive', 'not drive']
[4.444444444444443, 1.6666666666666656, 1.6666666666666656]
['not drive', 'not drive', 'not drive']
[4.444444444444444, 1.6666666666666663, 1.6666666666666663]
['not drive', 'not drive', 'not drive']
[4.444444444444445, 1.6666666666666665, 1.6666666666666665]
number of literation 39

The optimal policy changes from "not drive(top), not drive(rolling), drive(bottom)" to "not drive(top), not drive(rolling), not drive(bottom)". I think the reason for this is that the decrease of discount factor will discourage the people to make efforts to get back to the top. Driving at the bottom will cost 1 compared to not driving since the far future is less important from now due to the discount factor, the optimal policy changes. In other cases, it is the same reason that people are not willing to drive due to the fact that the far future is less important.

-- Changing transition probability

	Top	rolling	bottom (drive)
rolling	**0.3**	**0.6**	**0.1**

If we change the transition probability matrix(drive) from the above to:

	Top	rolling	bottom (drive)
rolling	**0.8**	**0.1**	**0.1**

Which means there are 80% of the chance that if drive, we can get to the top.

Below is the output for this change:

[11.610470978549268, 8.788688800331446, 8.194629394390851]
['not drive', 'drive', 'drive']
[11.611149060067136, 8.789366881849315, 8.195307475908722]
['not drive', 'drive', 'drive']
[11.611691525281431, 8.789909347063611, 8.195849941123017]
['not drive', 'drive', 'drive']
[11.612125497452869, 8.790343319235047, 8.196283913294453]
['not drive', 'drive', 'drive']
[11.612472675190018, 8.790690496972196, 8.196631091031604]
number of literation 39

The optimal policy now becomes" ['not drive', 'drive', 'drive]" which means people are willing to drive during the rolling step compared to before. This is intuitively correct because there is a high possibility that they can get back to the top and staying at top can have high rewards. People at the top won't drive because they know that there is a high possibility of getting back to the top if rolling. As a result, people will wait until the rolling step to drive.

-- change reward for a single action

For this question, I will increase the reward for not driving at the bottom. I will change the reward for not driving for 1 to 4. The optimal policy becomes ['not drive', 'not drive', 'not drive'] , Which means in all cases, people are not willing to drive.

Below is the output for this case:

['not drive', 'not drive', 'not drive']
[16.08571679655394, 16.99480770314147, 19.99480770314147]
['not drive', 'not drive', 'not drive']
[16.08675525482416, 16.995846162513175, 19.995846162513175]
['not drive', 'not drive', 'not drive']
[16.087586021704688, 16.99667693001054, 19.99667693001054]
['not drive', 'not drive', 'not drive']
[16.088250635357156, 16.997341544008435, 19.997341544008435]
number of literation 39

This output makes sense because staying at bottom has a much higher reward now and people will not stay at the top or drive at rolling stage. They can just stay at the bottom and not drive which does not cost anything and will earn the highest reward. As a result, there is no reason for people to drive at any stages.