CSE250 HW9 Q4

December 4, 2018

```
In [16]: import numpy as np
         import math
         import matplotlib.pyplot as plt
         import pandas as pd
```

```
In [86]: rewards = pd.read_csv('rewards.txt', header=None).values
         p1_sparse = pd.read_csv('prob_a1.txt', delim_whitespace = True, header=None).values
         p2_sparse = pd.read_csv('prob_a2.txt', delim_whitespace = True, header=None).values
         p3_sparse = pd.read_csv('prob_a3.txt', delim_whitespace = True, header=None).values
         p4_sparse = pd.read_csv('prob_a4.txt', delim_whitespace = True, header=None).values
         s = rewards.shape[0]   #number of states
         gamma = 0.99
```

```
In [87]: def convertToMatrix(p_sparse,s):
             result = np.zeros((s,s))
             for i in range(p_sparse.shape[0]):
                 result[int(p_sparse[i,0])-1,int(p_sparse[i,1])-1] = p_sparse[i,2]
             return result
```

```
In [88]: p1 = convertToMatrix(p1_sparse,s)
         p2 = convertToMatrix(p2_sparse,s)
         p3 = convertToMatrix(p3_sparse,s)
         p4 = convertToMatrix(p4_sparse,s)
```

0.1 Policy iteration

```
In [92]: def ComputeValueFunction(policy,p1,p2,p3,p4,gamma,rewards,s):
             p = np.zeros((s,s))   #transition matrix
             for i in range(s):    #loop over all policies
                 if policy[i,0] == 1:
                     p[i,:] = p1[i,:]
                 elif policy[i,0] == 2:
                     p[i,:] = p2[i,:]
                 elif policy[i,0] == 3:
                     p[i,:] = p3[i,:]
                 else:
                     p[i,:] = p4[i,:]
             I = np.eye(s)
```

```
            value = np.dot(np.linalg.inv(I-gamma*p),rewards)
            return value
```

```
In [93]: def CalcGreedyPolicy(p1,p2,p3,p4,value,policy):
            new_policy = policy.copy()
            s = new_policy.shape[0]
            for i in range(s):
                new_policy[i,0] = FindBestAction(p1,p2,p3,p4,value,i)
            return new_policy

         def FindBestAction(p1,p2,p3,p4,value,state):
            max_value = np.zeros((4,1))
            max_value[0,0] = np.dot(p1[state,:],value)
            max_value[1,0] = np.dot(p2[state,:],value)
            max_value[2,0] = np.dot(p3[state,:],value)
            max_value[3,0] = np.dot(p4[state,:],value)
            best_action = max_value.argmax() + 1
            return best_action
```

```
In [166]: #initialize policy
          policy = np.ones((s,1)).astype(int)
          iterations = 100
          for i in range(iterations):
              value = ComputeValueFunction(policy,p1,p2,p3,p4,gamma,rewards,s)
              policy = CalcGreedyPolicy(p1,p2,p3,p4,value,policy)
```

```
In [162]: value = value.reshape(9,9).T
          policy = policy.reshape(9,9).T
```

```
In [163]: value
```

```
Out[163]: array([[   0.        ,    0.        ,    0.        ,    0.        ,
                     0.        ,    0.        ,    0.        ,    0.        ,
                     0.        ],
                  [   0.        ,   65.77308407,   67.13647421,   77.84605   ,
                     79.84451583,   72.47511769, -100.        ,    0.        ,
                    100.        ],
                  [   0.        ,   55.88294346, -100.        ,   70.30818136,
                     81.34440225,   83.04847989,   84.88054612,   96.87232244,
                     98.71875987],
                  [   0.        ,   54.92298013,   50.47656297,   59.66641187,
                      0.        ,   80.95826449,    0.        ,   97.04482865,
                     98.72729893],
                  [  53.50968756,   54.14557214,    0.        , -100.        ,
                   -100.        ,   61.77980767, -100.        ,   88.22035599,
                    100.        ],
                  [   0.        ,   52.50402036,   43.9359876 ,   51.09137525,
                     61.00715483,   71.78642614,   73.94661407,   85.18458536,
                     97.57257319],
```

```
       [   0.        ,   43.77254574, -100.        ,    0.        ,
           0.        ,   70.35142939,    0.        , -100.        ,
          88.40593622],
       [   0.        ,   47.95296148,   48.76871928,   58.14735126,
          59.39003194,   60.1688947 , -100.        ,    0.        ,
         100.        ],
       [   0.        ,    0.        ,    0.        ,    0.        ,
           0.        ,    0.        ,    0.        ,    0.        ,
           0.        ]])
```

```
In [164]: policy
```

```
Out[164]: array([[1, 1, 1, 1, 1, 1, 1, 1, 1],
                 [1, 3, 3, 3, 4, 4, 1, 1, 1],
                 [1, 2, 1, 3, 3, 3, 3, 3, 2],
                 [1, 2, 3, 2, 1, 2, 1, 3, 4],
                 [3, 2, 1, 1, 1, 2, 1, 3, 1],
                 [1, 2, 1, 3, 3, 3, 3, 3, 2],
                 [1, 2, 1, 1, 1, 2, 1, 1, 4],
                 [1, 3, 3, 3, 3, 2, 1, 1, 1],
                 [1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

0.2 Value Iteration

```
In [179]: def ComputeOptimalValue(p1,p2,p3,p4,gamma,rewards,s,iterations):
              value = np.zeros((s,1))
              for i in range(iterations):
                  value = rewards + gamma*FindMaxValue(p1,p2,p3,p4,value,s)
              return value
```

```
In [180]: def FindMaxValue(p1,p2,p3,p4,value,s):
              col_1 = np.dot(p1,value)
              col_2 = np.dot(p2,value)
              col_3 = np.dot(p3,value)
              col_4 = np.dot(p4,value)
              matrix = np.concatenate((col_1, col_2, col_3, col_4), axis=1)
              max_value = np.amax(matrix, axis=1).reshape(s,-1)
              return max_value
```

```
In [200]: optimal_value = ComputeOptimalValue(p1,p2,p3,p4,gamma,rewards,s,1200)
          value_iter_policy = CalcGreedyPolicy(p1,p2,p3,p4,optimal_value,policy)
```

```
In [201]: optimal_value = optimal_value.reshape(9,9).T
          value_iter_policy = value_iter_policy.reshape(9,9).T
```

```
In [202]: optimal_value
```

```
Out[202]: array([[  0.        ,   0.        ,   0.        ,   0.        ,
                    0.        ,   0.        ,   0.        ,   0.        ,
```

```
        0.        ],
       [  0.        ,  65.77265447,  67.13604131,  77.84555742,
         79.84401676,  72.47466897, -99.99942159,   0.        ,
         99.99942159],
       [  0.        ,  55.88257   , -99.99942159,  70.30773948,
         81.34389984,  83.04797372,  84.88003597,  96.87174802,
         98.71818168],
       [  0.        ,  54.92260903,  50.47623177,  59.66602951,
          0.        ,  80.95776484,   0.        ,  97.04425388,
         98.72672072],
       [ 53.50931682,  54.1452014 ,   0.        , -99.99942159,
        -99.99942159,  61.77941892, -99.99942159,  88.21983929,
         99.99942159],
       [  0.        ,  52.50365573,  43.93567471,  51.09105085,
         61.00677664,  71.78598851,  73.94616987,  85.18408062,
         97.57200189],
       [  0.        ,  43.77223302, -99.99942159,   0.        ,
          0.        ,  70.35099507,   0.        , -99.99942159,
         88.40541907],
       [  0.        ,  47.95264021,  48.76839752,  58.14697651,
         59.38965408,  60.16851666, -99.99942159,   0.        ,
         99.99942159],
       [  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ,   0.        ,   0.        ,
          0.        ]])
```

In [203]:
```
value_iter_policy
```

Out[203]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 3, 3, 3, 4, 4, 1, 1, 1],
       [1, 2, 1, 3, 3, 3, 3, 3, 2],
       [1, 2, 3, 2, 1, 2, 1, 3, 4],
       [3, 2, 1, 1, 1, 2, 1, 3, 1],
       [1, 2, 1, 3, 3, 3, 3, 3, 2],
       [1, 2, 1, 1, 1, 2, 1, 1, 4],
       [1, 3, 3, 3, 3, 2, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1]])
```