lab-3

November 12, 2018

1 Learning and Decision Making

1.1 Laboratory 3: Partially observable Markov decision problems

In the end of the lab, you should submit all code/answers written in the tasks marked as "Activity n. XXX", together with the corresponding outputs and any replies to specific questions posed to the e-mail adi.tecnico@gmail.com. Make sure that the subject is of the form [<group n.>] LAB <lab n.>.

1.1.1 1. Modeling

Consider once again the POMDP problem from the homework and represented in the transition diagram below.

Recall that:

- All transitions occur with probability 1 except those from state A, where the probabilities are indicated under the edge label.
- At each step, the agent makes an observation corresponding to the letter in the state designation. Such observation occurs with probability 1.

Consider throughout that $\gamma = 0.9$.

Activity 1. Implement your POMDP in Python. In particular,

- Create a list with all the states;
- Create a list with all the actions;
- Create a list with all the observations
- For each action, define a `numpy` array with the corresponding transition probabilities;
- For each action, define a `numpy` array with the corresponding observation probabilities;
- Define a `numpy` array with the cost describing the problem.

The order for the states and actions used in the transition probability and cost matrices should match that in the lists of states and actions.

Note: Don't forget to import `numpy`.

```
In [91]: import numpy as np


         X = [ 'A', 'B1', 'B2', 'C', 'D', 'E', 'F']

         A = ['a', 'b', 'c']

         Z = ['A', 'B', 'C', 'D', 'E', 'F']

         # Preserving the order in X

         Pa =np.array([[0, 0.5, 0.5, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 1, 0],
               [0, 0, 0, 0, 0, 0, 1],
               [0, 1, 0, 0, 0, 0, 0],
               [0, 0, 1, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0]
              ])

         Pb =np.array([[0, 0.5, 0.5, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 1],
               [0, 0, 0, 0, 0, 1, 0],
               [0, 1, 0, 0, 0, 0, 0],
               [0, 0, 1, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0]
              ])



         Pc =np.array([[0, 0.5, 0.5, 0, 0, 0, 0],
               [0, 0, 0, 1, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0],
               [0, 1, 0, 0, 0, 0, 0],
               [0, 0, 1, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0]
              ])

         Za = np.array([
             [1, 0, 0, 0, 0, 0],
             [0, 1, 0, 0, 0, 0],
             [0, 1, 0, 0, 0, 0],
             [0, 0, 1, 0, 0, 0],
             [0, 0, 0, 1, 0, 0],
             [0, 0, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 1]
```

```
   ])

Zb = np.array([
    [1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 1]
   ])

Zc = np.array([
    [1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 1]
   ])

C = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1],
    [0, 0, 0]
])
```

1.1.2 2. Sampling

You are now going to sample random trajectories of your POMDP and observe the impact it has on the corresponding belief.

Activity 2. Generate a random POMDP trajectory using a uniformly random policy. In particular, from a random initial state x_0 generate:

1. A sequence of 10,000 states by selecting the actions uniformly at random;
2. The corresponding sequence of 10,000 actions;
3. The corresponding sequence of 10,000 observations.

```
In [92]: states = np.zeros( (10001, 1) , dtype = np.int8)
         actions = np.zeros( (10000, 1), dtype = np.int8)
         observations = np.zeros( (10000, 1), dtype = np.int8)

         Policy = [1.0/3.0, 1.0/3.0, 1.0/3.0]

         Probs = [Pa, Pb, Pc]

         Obs = [Za, Zb, Zc]


         states[0] = np.random.choice(7, p=[1.0/7.0,1.0/7.0,1.0/7.0,1.0/7.0,1.0/7.0,1.0/7.0,1.0

         #print(states[0][0])
         #print(np.random.choice(3, p=policy))
         #print(np.random.choice(3, p=Obs[0][0]))


         for i in range(0, 10000):
             action = np.random.choice(3, p=Policy)

             actions[i] = action
             states[i+1] = np.random.choice(7, p=Probs[action][states[i][0]])
             observations[i] = np.random.choice(6, p=Obs[action][states[i+1][0]])

         '''
         for i in range(0,10):
             print(states[i])
         for i in range(0,10):
             print("a", actions[i])
         for i in range(0,10):
             print("o",observations[i])
         '''
         print(actions)
         print(states)
         print(observations)

[[0]
 [1]
 [0]
 ...
 [2]
 [0]
 [2]]
[[4]
 [2]
 [5]
 ...
 [3]
 [1]
 [3]]
[[1]
 [4]
 [0]
 ...
 [2]
 [1]
 [2]]
```

Activity 3. For the POMDP trajectory generated in Activity 2, compute the corresponding sequence of beliefs, assuming that the agent does not know its initial state. Report the resulting beliefs, ignoring duplicate beliefs or beliefs whose distance is smaller than 10^{-3}.

Note 1: You may want to define a function `belief_update` that receives a belief, an action and an observation and returns the updated belief.

Note 2: To compute the distance between vectors, you may find useful `numpy`'s function `linalg.norm`.

```
In [93]: b0=[1/7,1/7,1/7,1/7,1/7,1/7,1/7]

         error = 1e-3

         beliefs = []
         beliefs.append(b0)

         def belief_update(belief, action, observation):
             newBelief = np.matmul(np.matmul(belief, action), observation)


             aux= sum(newBelief)

             if(aux!=0):
                 newBelief=newBelief/aux
             return newBelief

         def check_new_belief(newb):
             append = True
             for b in beliefs:
                 if( np.linalg.norm(b - newb) <= error):
                     append = False
             if(append == True):
                 beliefs.append(newb)
```

```
        for i in range(0, 10000):
            for b in beliefs:
                newb = belief_update(b,
                                     Probs[ actions[i][0] ],
                                     np.diag( Obs[ actions[i][0] ][ :, observations[i][0] ] ))
                check_new_belief(newb)

        print(beliefs)
        print(len(beliefs), "beliefs")
```

```
[[0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.142857
10 beliefs
```

1.1.3 3. Solution methods

In this section you are going to compare different non-exact solution methods.

Activity 4 Compute the solution for the underlying MDP and report the corresponding optimal policy and optimal cost-to-go.

** Note:** You may reuse code from previous labs.

```
In [94]: gama = 0.9
         pi = np.ones( (7,3) ) / 7
         Q = np.zeros( (7,3) )
         quit = False
         i = 0
         I = np.eye(7)


         ca = C[:,0,None]
         cb = C[:,1,None]
         cc = C[:,2,None]


         while not quit:
             Cpi = np.diag(pi[:, 0]).dot(ca) + np.diag(pi[:, 1]).dot(cb) + np.diag(pi[:, 2]).do
             Ppi = np.diag(pi[:, 0]).dot(Pa) + np.diag(pi[:, 1]).dot(Pb) + np.diag(pi[:, 2]).do
             J = np.linalg.inv(I - gama * Ppi).dot(Cpi)

             Qa = ca + gama * Pa.dot(J)
             Qb = cb + gama * Pb.dot(J)
```

```
            Qc = cc + gama * Pc.dot(J)

            Q[:, 0, None] = Qa
            Q[:, 1, None] = Qb
            Q[:, 2, None] = Qc

            pinew = np.zeros((7, 3))
            pinew[:, 0, None] = np.isclose(Qa, np.min([Qa, Qb, Qc], axis = 0), atol=1e-8, rto
            pinew[:, 1, None] = np.isclose(Qb, np.min([Qa, Qb, Qc], axis = 0), atol=1e-8, rto
            pinew[:, 2, None] = np.isclose(Qc, np.min([Qa, Qb, Qc], axis = 0), atol=1e-8, rto

            pinew = pinew / np.sum(pinew, axis=1, keepdims=True)

            quit = (pi == pinew).all()
            pi = pinew
            i += 1


        #print("Q = \n%s\n" % Q)
        #print(i)
        print("pi = \n%s\n" % pi)

        Policy = pi

        Ppi = Policy[:, 0, None]*Pa + Policy[:, 1, None]*Pb + Policy[:, 2, None]*Pc
        Cpi = (Policy * C).sum(axis=1)

        #print(Pp)
        #print(Cp)


        Jpi = np.dot(np.linalg.inv(I - gama * Ppi), Cpi)

        print("Cost-to-go =\n%s\n" % Jpi)
```

```
pi =
[[0.33333333 0.33333333 0.33333333]
 [0.         1.         0.        ]
 [1.         0.         0.        ]
 [0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]]

Cost-to-go =
[7.01107011 6.67896679 6.67896679 7.01107011 7.01107011 7.3099631
 6.3099631 ]
```

Activity 5 For each of the beliefs computed in Activity 3, compute the action prescribed by:

- The MLS heuristic;
- The AV heuristic;
- The Q-MDP heuristic.

```
In [95]: OptimalCost = Q
         OptimalPolicy = pi
         print(OptimalCost)

         mls = []
         av = []
         q_mdp = []

         def MLS(belief):
             state = np.argmax(belief)
             mls.append( np.argmax(OptimalPolicy[state]) )
             return np.argmax(OptimalPolicy[state])

         def AV(belief):
             av.append( np.argmax(np.dot(belief, OptimalPolicy)) )
             return np.argmax(np.dot(belief, OptimalPolicy))


         def Q_MDP(belief):
             h = np.dot(belief, OptimalCost)
             q_mdp.append( np.argmin(h) )
             return np.argmin(h)



         i = 0
         for b in beliefs:
             print("i = %d" % i)
             print("MLS(%s) = %s" % (b, MLS(b)))
             print("AV(%s) = %s" % (b, AV(b)))
             print("Q_MDP(%s) = %s" % (b, Q_MDP(b)))
             i+=1
```

```
[[7.01107011 7.01107011 7.01107011]
 [7.57896679 6.67896679 7.3099631 ]
 [6.67896679 7.57896679 7.3099631 ]
 [7.01107011 7.01107011 7.01107011]
 [7.01107011 7.01107011 7.01107011]
 [7.3099631  7.3099631  7.3099631 ]
 [6.3099631  6.3099631  6.3099631 ]]
i = 0
MLS([0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.1428
AV([0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14285
Q_MDP([0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14285714285714285, 0.14
i = 1
MLS([0.  0.5 0.5 0.  0.  0.  0. ]) = 1
AV([0.  0.5 0.5 0.  0.  0.  0. ]) = 0
Q_MDP([0.  0.5 0.5 0.  0.  0.  0. ]) = 0
i = 2
MLS([0. 0. 0. 0. 0. 0. 0.]) = 0
AV([0. 0. 0. 0. 0. 0. 0.]) = 0
Q_MDP([0. 0. 0. 0. 0. 0. 0.]) = 0
i = 3
MLS([0. 0. 0. 0. 0. 1. 0.]) = 0
AV([0. 0. 0. 0. 0. 1. 0.]) = 0
Q_MDP([0. 0. 0. 0. 0. 1. 0.]) = 0
i = 4
MLS([1. 0. 0. 0. 0. 0. 0.]) = 0
AV([1. 0. 0. 0. 0. 0. 0.]) = 0
Q_MDP([1. 0. 0. 0. 0. 0. 0.]) = 0
i = 5
MLS([0. 0. 0. 0. 1. 0. 0.]) = 0
AV([0. 0. 0. 0. 1. 0. 0.]) = 0
Q_MDP([0. 0. 0. 0. 1. 0. 0.]) = 0
i = 6
MLS([0. 0. 1. 0. 0. 0. 0.]) = 0
AV([0. 0. 1. 0. 0. 0. 0.]) = 0
Q_MDP([0. 0. 1. 0. 0. 0. 0.]) = 0
i = 7
MLS([0. 0. 0. 0. 0. 0. 1.]) = 0
AV([0. 0. 0. 0. 0. 0. 1.]) = 0
Q_MDP([0. 0. 0. 0. 0. 0. 1.]) = 0
i = 8
MLS([0. 0. 0. 1. 0. 0. 0.]) = 0
AV([0. 0. 0. 1. 0. 0. 0.]) = 0
Q_MDP([0. 0. 0. 1. 0. 0. 0.]) = 0
i = 9
MLS([0. 1. 0. 0. 0. 0. 0.]) = 1
AV([0. 1. 0. 0. 0. 0. 0.]) = 1
Q_MDP([0. 1. 0. 0. 0. 0. 0.]) = 1
```

Activity 6 Suppose that the optimal cost-to-go function for the POMDP can be represented using the α-vectors

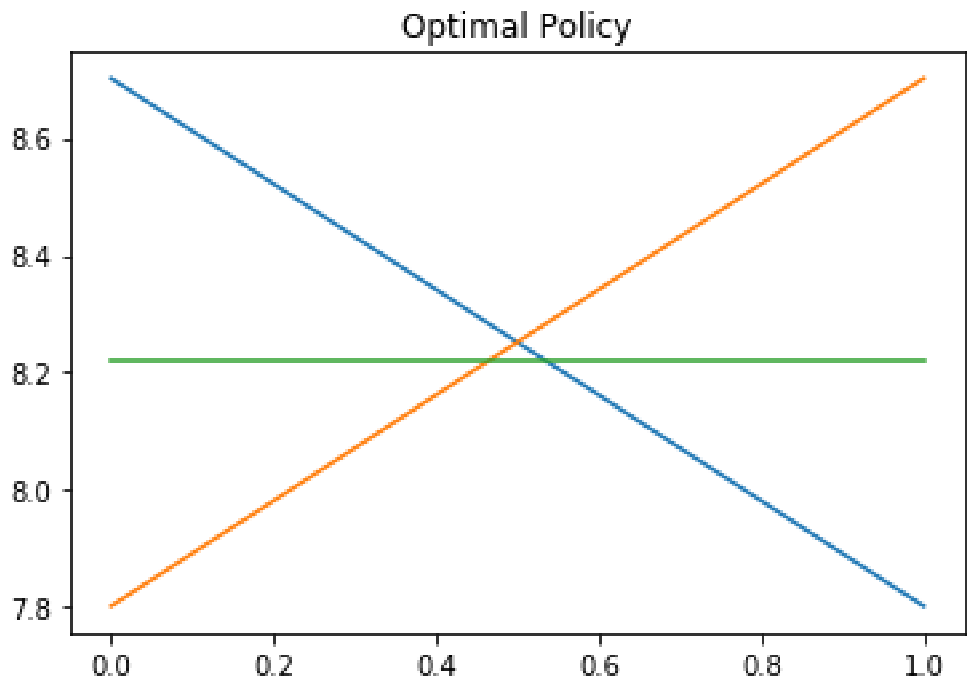
$$\left\{ \begin{bmatrix} 8.39727208 \\ 8.70168739 \\ 7.80168739 \\ 8.39727208 \\ 8.39727208 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.42645332 \\ 8.70168739 \\ 7.80168739 \\ 8.02145332 \\ 8.83145332 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.42645332 \\ 8.70168739 \\ 7.80168739 \\ 8.83145332 \\ 8.02145332 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.39727208 \\ 7.80168739 \\ 8.70168739 \\ 8.39727208 \\ 8.39727208 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.42645332 \\ 7.80168739 \\ 8.70168739 \\ 8.02145332 \\ 8.83145332 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.42645332 \\ 7.80168739 \\ 8.70168739 \\ 8.83145332 \\ 8.02145332 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.39727208 \\ 8.2192667 \\ 8.2192667 \\ 8.39727208 \\ 8.39727208 \\ 8.55748144 \\ 7.55748144 \end{bmatrix}, \begin{bmatrix} 8.4 \\ 8. \\ 8. \\ 8.0 \\ 8.8 \\ 8.5 \\ 7.5 \end{bmatrix}$$

where the first 3 vectors correspond to action a, the middle three vectors correspond to action b and the last 3 vectors correspond to action c. Using the α-vectors above,

- Represent the the optimal cost-to-go function for all beliefs of the form $\mathbf{b} = [0, \epsilon, 1 - \epsilon, 0, 0, 0, 0]$, with $\epsilon \in [0, 1]$.
- Compare the optimal policy with the MDP heuristics from Activity 5 in the beliefs computed in Activity 3.

** Note: ** Don't forget to import `matplotlib`, and use the magic `%matplotlib notebook`.

```
In [107]: import matplotlib.pyplot as plt

          b1 = np.array([[0.  , 0.5, 0.5, 0.  , 0.  , 0.  , 0. ]])
          b2 = np.array([[0., 1., 0., 0., 0., 0., 0.]])
          b3 = np.array([[0., 0., 1., 0., 0., 0., 0.]])


          OptimalCost = np.array([
              [8.39727208, 8.42645332, 8.42645332, 8.39727208, 8.42645332, 8.42645332, 8.39727
              [8.70168739, 8.70168739, 8.70168739, 7.80168739, 7.80168739, 7.80168739, 8.21926
              [7.80168739, 7.80168739, 7.80168739, 8.70168739, 8.70168739, 8.70168739, 8.21926
              [8.39727208, 8.02145332, 8.83145332, 8.39727208, 8.02145332, 8.83145332, 8.39727
              [8.39727208, 8.83145332, 8.02145332, 8.39727208, 8.83145332, 8.02145332, 8.39727
              [8.55748144, 8.55748144, 8.55748144, 8.55748144, 8.55748144, 8.55748144, 8.55748
              [7.55748144, 7.55748144, 7.55748144, 7.55748144, 7.55748144, 7.55748144, 7.55748
          ])

          alpha1 = [ 8.70168739, 7.80168739]
          alpha2 = [ 7.80168739, 8.70168739]
          alpha3 = [ 8.2192667, 8.2192667]

          alphas = [alpha1, alpha2, alpha3]

          plt.plot(alpha1)
          plt.plot(alpha2)
          plt.plot(alpha3)

          plt.title("Optimal Policy")

          plt.show()

          print("De 0 a cerca de 0.5 que é a intersecção entre acção a e acção b escolhe-se a a

          def optimalCostToGo(belief):
              J = np.dot(belief, OptimalCost)
              index = np.argmin(J)
              return index//3

          J1 = optimalCostToGo(b1)
          J2 = optimalCostToGo(b2)
          J3 = optimalCostToGo(b3)

          Jays = []
          Jays.append(J1)
          Jays.append(J2)
          Jays.append(J3)


          lastbeliefs = []
          lastbeliefs.append(b1)
          lastbeliefs.append(b2)
          lastbeliefs.append(b3)


          i = 0
          for b in lastbeliefs:
              print("i = %d" % i)
              print("MLS(%s) = %s" % (b, MLS(b)))
              print("AV(%s) = %s" % (b, AV(b)))
              print("Q_MDP(%s) = %s" % (b, Q_MDP(b)))
              print("Optimal Policy(%s) = %s"%(b, Jays[i]))
              i+=1
```



```
De 0 a cerca de 0.5 que é a intersecção entre acção a e acção b escolhe-se a acção a. Entre ce
i = 0
MLS([[0.  0.5 0.5 0.  0.  0.  0. ]]) = 1
AV([[0.  0.5 0.5 0.  0.  0.  0. ]]) = 0
Q_MDP([[0.  0.5 0.5 0.  0.  0.  0. ]]) = 6
Optimal Policy([[0.  0.5 0.5 0.  0.  0.  0. ]]) = 2
i = 1
MLS([[0. 1. 0. 0. 0. 0. 0.]]) = 1
AV([[0. 1. 0. 0. 0. 0. 0.]]) = 1
Q_MDP([[0. 1. 0. 0. 0. 0. 0.]]) = 3
Optimal Policy([[0. 1. 0. 0. 0. 0. 0.]]) = 1
i = 2
MLS([[0. 0. 1. 0. 0. 0. 0.]]) = 0
AV([[0. 0. 1. 0. 0. 0. 0.]]) = 0
Q_MDP([[0. 0. 1. 0. 0. 0. 0.]]) = 0
Optimal Policy([[0. 0. 1. 0. 0. 0. 0.]]) = 0
```

Insert your comments here