Dynamic Programming and Reinforcement Learning
Assignment 2

Submitted by:
Abhishek Subramanian Iyer - 2724035
Ankush Gulia - 2732561

INTRODUCTION

In this report, we will be describing the solutions we have achieved for the given markov decision problem
This report includes the following :

- The solutions we have achieved for all the three questions posed in this assignment
- The methods used to arrive at those particular solutions

1 A : STATIONARY DISTRIBUTION AND LONG RUN AVERAGE REPLACEMENT COST

We use the information given in the lecture and the problem statement to come to the conclusion that the system traverses 91 states. For the calculation of stationary distribution (Pi) and long run average replacement cost (Phi) we take a probability matrix and a Pi vector where we keep the first (in index terms - zeroth) element as 1 and we use a nested for loop to fill in the probability matrix values. We also take a reward vector (which in this problem statement is a cost vector). For calculating the stationary distribution we simply take a for loop and iterate over the number of states and use the formula **Pi = Pi * Probability**. Then long run average replacement (phi) is calculated by using the formula phi = reward * Pi. **The result we get = 0.1461.**

```
# --- PART A : FINIDNG STATIONARY DISTRIBUTION AND LONG RUN AVERAGE COST --- #
states = 91
P = np.zeros((states,states))
Pi = np.zeros((states))
Pi[0] = 1
for i in range(states):
    P[i][0] = 0.1 + i*0.01
    if i < 90:
        P[i][i+1] = 0.9 - i*0.01
print("Probability Matrix : ",P,"\n")
r = np.zeros((states))
r[0] = 1
for i in range(states):
    Pi=np.matmul(Pi,P)
np.set_printoptions(formatter={'float': lambda x: "{0:0.4f}".format(x)})
print("Stationary distribution : ", Pi,"\n")
phi = r@Pi
print("long run average cost i.e phi : ", round(phi,4),"\n")
```

1 B : SOLVING POISSON EQUATION

$$V + \varphi e = r + PV$$

The poisson equation, as stated above, takes into consideration Value function - V of current state (x), long time average reward - phi * e(which is just an identity matrix), r - reward vector, P - probability matrix filled with probabilities of going from state x to y, Value function - V of the next state (y).

```
# --- PART B : SOLVING THE POISSON EQUATION --- #
def Poisson(r,P):
    reward = np.append(r,0)
    poisson = np.identity(91)-P
    poisson = np.pad(poisson, [(0, 1), (0, 1)], mode='constant', constant_values=1)
    poisson[91][91] = 0
    poisson = np.linalg.solve(poisson, reward)
    return poisson
print("Avg cost after solving poisson eqn :",round(Poisson(r,P)[-1],4),"\n")
```

We solve the poisson equation by transforming the equation and we take a variable called poisson which is V and phi and equate it to I (identity matrix) - Probability and for the reward part in the equation we add that in np.linalg.solve, where we get 91 equations for the 91 states and then we get the long run average replacement cost (phi).

The result we get = 0.1461

1 C : POLICY ITERATION AND VALUE ITERATION

In this part we work with a preventative action which has a cost of 0.5. We consider two actions, first one being that the system keeps going to the next state with gradual degradation and the next one being that we take preventative replacement.

```
Prob_actions1 = P
reward_actions1 = np.zeros((states))
for i in range(states):
    reward_actions1[i] = i*0.01 + 0.1

Prob_actions2 = np.zeros((states,states))
reward_actions2 = np.zeros((states))
for i in range(states):
    Prob_actions2[i][0] = 1
    reward_actions2[i] = 0.5
```

We consider 2 probability matrices : Prob_actions1 which is the probability that action 1 will be taken for the current which is just equal to the original probability matrix and Prob_actions2 which is the probability that action 2 will be taken for a current state. Since there are two actions, we also have to take two reward vectors, where reward_actions1 is just the reward when action 1 is taken which is gradual degradation of the system and reward_actions2 is the reward for taking the preventative replacement which is 0.5.

We then solve the policy iteration step wise where we fix a policy and then we use the V and phi from poisson equation and then we iterate over the two sets of actions that we consider and find alpha ~ = minimum(action 1 = reward_actions1 + Prob_actions1@poisson ,actions2 = reward_actions2 + Prob_actions2@poisson) [poisson is the variable name for V in this case] (we would do argmax for maximizing rewards but here we have cost so we find minimum) and we keep doing this till our policy we consider is equal to alpha ~. Once we have that, that is the optimal policy. **The average optimal policy we get is 0.14491 at the state 13.**

For Value iteration we follow the same steps as policy iteration, just that instead of finding optimal policy values we find optimal value function values by iterating over both action sets for all the states. **The average optimal value function we get is 0.14491**

2 : IMAGES

1 - The probability matrix and stationary distributions look as follows -

```
Probability Matrix :  [[0.1000 0.9000 0.0000 ... 0.0000 0.0000 0.0000]
 [0.1100 0.0000 0.8900 ... 0.0000 0.0000 0.0000]
 [0.1200 0.0000 0.0000 ... 0.0000 0.0000 0.0000]
 ...
 [0.9800 0.0000 0.0000 ... 0.0000 0.0200 0.0000]
 [0.9900 0.0000 0.0000 ... 0.0000 0.0000 0.0100]
 [1.0000 0.0000 0.0000 ... 0.0000 0.0000 0.0000]]

Stationary distribution :  [0.1461 0.1315 0.1170 0.1030 0.0896 0.0771 0.0655 0.0550 0.0457 0.0374
 0.0303 0.0243 0.0192 0.0150 0.0115 0.0087 0.0066 0.0049 0.0035 0.0026
 0.0018 0.0013 0.0009 0.0006 0.0004 0.0003 0.0002 0.0001 0.0001 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000]
```

2 - The policy matrix looks as follows -

```
Policy - :
 [1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
 1.0000 1.0000 1.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000 2.0000
 2.0000]
preventive actions starts from :
 [13]
```