Dynamic Programming and Reinforcement Learning : Assignment 1

Submitted by:
Abhishek Subramanian Iyer - 2724035
Venkat Mohit Sornapudi - 2721697

INTRODUCTION

In this report, we will be describing the solutions we have achieved for the given revenue management problem.
This report includes the following :

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

1 A : TOTAL EXPECTED REVENUE AND OPTIMAL POLICY

$$V_t(x) = \max_{a=1,\dots,n} \left\{ \sum_{i=1}^{a} \lambda_t(i)\left(f_a + V_{t+1}(x-1)\right) + \left(1 - \sum_{i=1}^{a} \lambda_t(i)\right) V_{t+1}(x) \right\}$$

We use the above mentioned formula to get the total expected revenue. We go backwards in time and we go forwards in capacity for calculating the total expected revenue. We implement a nested for loop where the outermost loop is for time and it runs in a reversed order. The inner runs over the states, here capacities. The last loop runs over the classes (0,1,2). In this loop, we implemented the above mentioned formula. After the value function matrix is filled, we used argmax function to fill in our optimal policy.

The solution we get for total expected revenue ($V_1(C)$) is 30318.08. Value function:

	0	1	2	3	4	...	595	596	597	598	599	600
0	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
1	500.000000	500.000000	500.000000	500.000000	500.000000	...	459.706243	434.614238	393.228539	324.531707	209.758502	0.0
2	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	...	789.802844	699.624902	577.084127	417.876184	209.758502	0.0
3	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	...	953.261768	799.844831	624.367145	417.876184	209.758502	0.0
4	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	1012.602935	829.245358	624.367145	417.876184	209.758502	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
96	29799.627936	29795.239298	29790.829803	29786.399326	29781.947745	...	1032.524664	829.245358	624.367145	417.876184	209.758502	0.0
97	29931.655630	29927.162565	29922.648237	29918.112520	29913.555287	...	1032.524664	829.245358	624.367145	417.876184	209.758502	0.0
98	30062.085414	30057.486814	30052.866540	30048.224465	30043.560461	...	1032.524664	829.245358	624.367145	417.876184	209.758502	0.0
99	30190.900474	30186.195230	30181.467898	30176.718348	30171.946452	...	1032.524664	829.245358	624.367145	417.876184	209.758502	0.0
100	30318.083973	30313.270972	30308.435466	30303.577324	30298.696413	...	1032.524664	829.245358	624.367145	417.876184	209.758502	0.0

Optimal policy (as optimal prices):

	0	1	2	3	4	5	6	7	8	9	...	591	592	593	594	595	596	597	598	599	600
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	None
1	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	...	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	300.0	None
2	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	...	500.0	500.0	500.0	500.0	500.0	500.0	500.0	300.0	300.0	None
3	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	...	500.0	500.0	500.0	500.0	500.0	500.0	300.0	300.0	300.0	None
4	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	...	500.0	500.0	500.0	500.0	300.0	300.0	300.0	300.0	300.0	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	...	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	None
97	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	...	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	None
98	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	...	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	None
99	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	...	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	None
100	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	...	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	300.0	None

1 B : PLOT

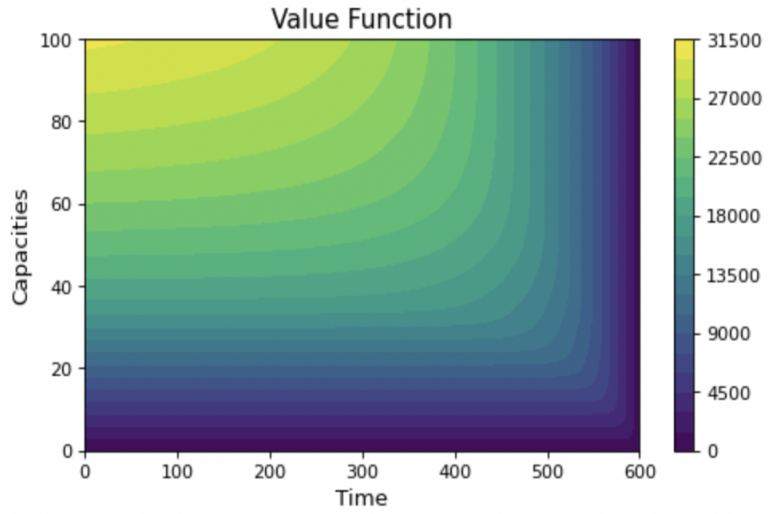


Fig: Plot showing capacity vs time for the Value function

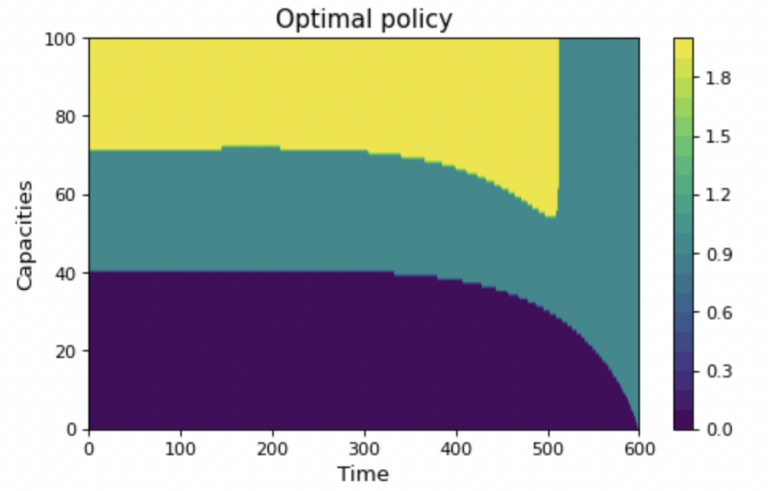


Fig: Plot showing capacities vs time for Optimal policy (as optimal class values)

1 C : SIMULATION

The following table shows the results that we get when we run for 1000 and 10,000 simulations:

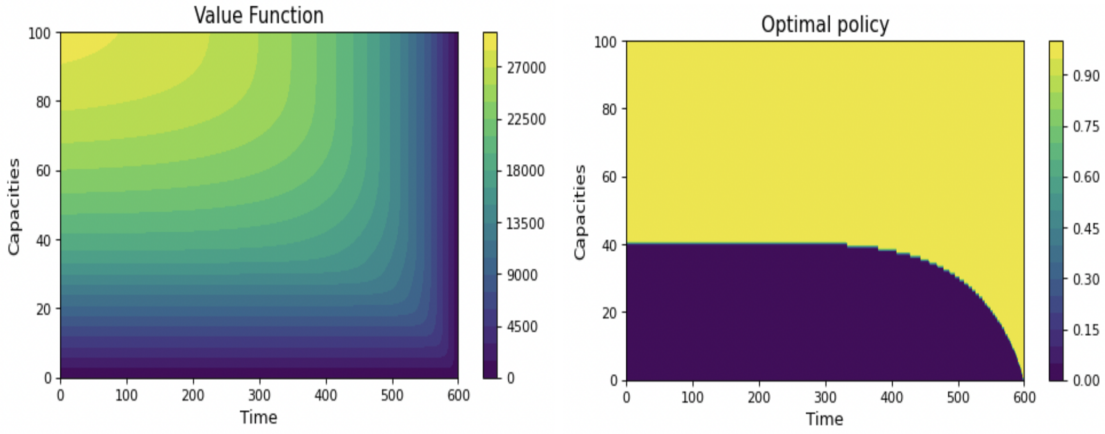
No.of simulations	Max Profit	Avg Profit	Min Profit
1000	37192.9	41100	32700
10,000	37147.2	42100	31800

For simulating the demands, we ran a nested loop where the outermost loop runs for the number of simulations and created random demands (from [0,200,300,500]). The innermost loop runs for the time period and in this loop we have two if statements. The first one checks if the capacity at any given instance is < or = 0, if it is we break the loop. The second one checks if the demand at the given instance is greater than or equal to action i.e., the price at that instance, in that case, the ticket is sold and capacity is reduced.

1 D : PRICE CANNOT GO DOWN -

We applied 'prices cannot go down' constraint on A and C. Now, the results are :

- For (A), we get the total estimated revenue as 28982.40. This is a 4.40 % price reduction compared to the case in which do not apply the condition. The plots below show time vs capacities for value function and optimal policy respectively.



- For (C), on 1000 simulations we get the avg profit as 39639.2 and on 10,000 simulations we get the avg profit as 39613.8.

For this constraint we simply take a boolean variable price_not_down that is used in if statements to see if it is true or false. If it is false, then normal A and C are executed. If it is true, then we check the prices over T and don't let them dip.