Assignment 1

Michele Vannucci (2819493)
Arturo Abril (2813498)

November 2024

The code used to obtain the results of this report can be found here[1].

Question A The states $x \in \mathcal{X}$ correspond to the possible number of items left in the inventory. We can define the state space $\mathcal{X}$ formally as:

$$\mathcal{X} = \{i \mid i \in \mathbb{N}_0, 0 \leqslant i \leqslant 100\}$$

x_t is the the number of remaining items at time t.
The actions $a \in \mathcal{A}$ represent the different prices at which the product can be offered. Since in the initial description of the problem the price can be changed at each point in time, the action space is:

$$\mathcal{A} = \{a_1, a_2, a_3\}$$

Where: $a_1 = 200$ (price with sale probability of 0.1), $a_2 = 100$ (price with a sale probability of 0.5) and $a_3 = 50$ (price with a sale probability of 0.8). Note that the actions are independent of both time and state.

Question B The following defines the problem as a stochastic finite-horizon DP: **Actions** and **States** have already been defined,

- **Time**: $t \in \mathcal{T} = \{0, ..., 500\}$. t represents the number of time steps *left*.
- **Expected rewards**: $r_t(x, a) = \mathbf{1}_{\{x_t \geqslant 1\}} \cdot p(a_t) \cdot a_t$. $p(a)$ represents the probability of selling the item when setting price a.
- **Transition probabilities**: $\tilde{p}_t(x-1|x, a) = \mathbf{1}_{\{x_t \geqslant 1\}} \cdot p(a_t)$ and $\tilde{p}_t(x|x, a) = (1 - \mathbf{1}_{\{x_t \geqslant 1\}} \cdot p(a_t))$. All other transition probabilities are 0.

We want to find an *optimal policy* that tells us at what price we should sell our item, given the number of remaining items in the inventory and the number of time steps left, to maximize our total expected rewards. Formally, we want to find $\alpha^* : \mathcal{T} \times \mathcal{X} \to \mathcal{A}$. For this, we first define the *value function* $v_t(x)$, which gives the maximum total expected reward for a state x and time t. This value function is (for all $t \geqslant 1$):

$$v_t(x) = \mathbf{1}_{\{x \geqslant 1\}} \cdot \max_{a_i \in \mathcal{A}} \{p(a_i) \cdot a_i + p(a_i) \cdot v_{t-1}(x-1) + (1 - p(a_i)) \cdot v_{t-1}(x)\}$$

Note that $v_0(x) = v_t(0) = 0 \ \forall \ x, t$. With these boundary conditions, we can solve the problem recursively. A common way to do this is constructing a *value matrix* $\mathbf{V}$, in which rows represent the state and columns the time step (a 101×501 matrix in our case), and its values are $v_{xt} = v_t(x)$. For each state x and time step t there will be an action a that maximizes the total expected rewards. This actions are stored in a matrix of the same dimensions as the value matrix, which represent the optimal policy α^*. Formally (for $t \geqslant 1$ and $x \geqslant 1$)[2]:

$$\alpha_t^*(x) = \arg\max_{a_i \in \mathcal{A}} \{p(a_i) \cdot a_i + p(a_i) \cdot v_{t-1}(x-1) + (1 - p(a_i)) \cdot v_{t-1}(x)\}$$

See Appendix for implementation.

We are interested in the maximum total expected reward when we have 100 items and 500 time steps left, which is given by $v_{500}(100)$. After filling the value matrix $\mathbf{V}$, we find this value to be **13715.793**. This value is

[1] https://github.com/michelexyz/RL-assignments/tree/main/product_sell
[2] Since $v_0(x) = v_t(0) = 0 \ \forall \ x, t$, the values of $\alpha_t^*(0)$ and $\alpha_0^*(x)$ are irrelevant.

higher than the maximum value one would obtain if every item was sold at a safe price for every t and x. You can think of it as if we were given the opportunity to sell each of the 100 items 5 times (500 time steps divided by 100). If you were to always sell at 100, you would expect to sell almost everything, resulting in $\lesssim 10,000$ in revenue. The optimal policy can be seen in figure 1. We can interpret the results in terms of "risk". The more time steps left we have, the riskier we can be (or the greedier), trying to sell for 200. The number of items is also important, since the less items we have left, the more time steps we have to sell each, therefore we can risk more by selling for 200. Action "50" it's never picked in the optimal policy since the instant expected reward, $p(a_i) \cdot a_i$, it's higher for action "100" . This means that even if we have only 1 time step left selling for 100 would be better than selling for 50 on average. This is even more true if we have more time steps left. Finally, action "200" has a worse instant reward than 100 on average on as single time step, but has a better "long term reward" when we have more time available to sell all the items.

Question C The simulations were run by firstly drawing 1000×500 values in the range $[0; 1]$ from a uniform distribution, where 1000 is the number of simulations we want to compute, these valued are stored in a matrix $\mathbf{Z}$. Secondly, the rewards for each action in the optimal policy are summed together over 500 time steps, this process is repeated until we have gathered 1000 total rewards. The rewards are simulated taking the value $\mathbf{Z}_{s,t}$from the matrix and considering the item sold if $\mathbf{Z}_{s,t} < p(a_{\alpha_t^*(x)})$ where s and t are the simulation number and time step, respectively. The histogram of the total rewards of each of the 1000 simulations of the problem, where the optimal policy was followed, can be observed in figure 2. The dotted lines indicate both the average total reward of the simulations, 13751.0, and the previously calculated total expected reward, 13751.8. These two will be exactly the same as the number of simulations tend to infinity.

Question D The new state space needs to keep in memory the last used price, therefore we have to add an additional dimension. We define it as:

$$\mathcal{X} = \{\langle i, a_{\text{last}}\rangle \mid i \in \mathbb{N}_0, 0 \leqslant i \leqslant 100, a_{\text{last}} \in \{20, 100, 50\}\}$$

Where:i represents the inventory level (number of items remaining) and a_{last} represents the last chosen price. Finally, the action space, that is dependent on the current state $x = \langle i, a_{last}\rangle$, can be defined as:

$$\mathcal{A}_x = \{a \in \{200, 100, 50\} \mid a \leqslant a_{\text{last}}\}$$

Question E The formulation of this problem as a finite-horizon DP is analogous to the one described in **Question B**, except that now the actions available depend on the current state, this is, $a \in \mathcal{A}_x$ where the action space $\mathcal{A}_x$ has been defined in **Question D**. The new update formulas for the value function and the optimal policy are (for all $t \geqslant 1$):

$$v_t(\mathbf{x}) = \mathbf{1}_{\{x_1 \geqslant 1\}} \cdot \max_{a_i \in \mathcal{A}_x} \{p(a_i) \cdot a_i + p(a_i) \cdot v_{t-1}(x_1 - 1, a_i) + (1 - p(a_i)) \cdot v_{t-1}(x_1, a_i)\}$$

$$\alpha_t^*(\mathbf{x}) = \arg\max_{a_i \in \mathcal{A}_x} \{p(a_i) \cdot a_i + p(a_i) \cdot v_{t-1}(x_1 - 1, a_i) + (1 - p(a_i)) \cdot v_{t-1}(x_1, a_i)\}$$

The boundary conditions remain the same as in **Question B**.

We are interested in the maximum total expected reward when we have 100 items and 500 time steps left. Assuming we can start offering the product at the highest price, we look for the value $v_{500}(100, 200)$. This is, the value function for the state with 100 items left and last price set at 200, for time step 500. Solving the problem recursively, we find this value to be **13566.5**. This value is slightly lower than the one obtained in **Question B**, which makes sense since now we are under more strict conditions (we cannot increase the price). Figure 3 displays the optimal policy. Since the state space is now two-dimensional, we split the visualization across the second dimension (last price). Obviously, if we start with the lowest price we cannot change it and the optimal policy (which is the only policy possible) is just keeping the same price until the end. In the case that where our upper bound on the price is 100, we are not interested in decreasing it ever, since as discussed before, it will always be more beneficial, on average, to sell at 100 than 50, regardless of the state we are in. Finally, when we are in a state $x = \langle i, 200\rangle$, we see in Fig. 3 that the policy matrix it's similar to the one shown in Fig. 1, the interpretation for this triangular pattern it's the same we gave for **Question B**. The slight difference in the graph, that it' also reflect by a slight lower expected reward, it's represented by the items being sold for 100 later in time (the green area is smaller) for the second problem definition. This could be motivated by the fact that, once the price has been lowered, we can't go back and it is the optimal strategy to exploit the possibility of selling for 200 as long as we have some time or few items left to sell.

A Implementation

A.1 The backward induction

To parallelize the backward recursion algorithm, we vectorized the computation as detailed in this section. The columns of the value matrix are defined as :

$$\mathbf{v}_t = \begin{pmatrix} v_t(0) \\ \vdots \\ v_t(100) \end{pmatrix}$$

We define the probability vector $\mathbf{p}$ and the probability matrix $\mathbf{P}$ as:

$$\mathbf{p} = \begin{pmatrix} p(a_1) \\ p(a_2) \\ p(a_3) \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} p(a_1) & p(a_2) & p(a_3) \\ 1-p(a_1) & 1-p(a_2) & 1-p(a_3) \end{pmatrix}$$

The action vector $\mathbf{a}$ as:

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Then we can construct a 100×2 matrix $\tilde{\mathbf{V}}_t$:

$$\tilde{\mathbf{V}}_t = \begin{pmatrix} v_t(0) & v_t(1) \\ \vdots & \vdots \\ v_t(499) & v_t(500) \end{pmatrix}$$

which can be thought as stacking vertically the column vectors $\mathbf{v}_t[:-1]$ and $\mathbf{v}_t[1:]$. The value matrix can then be recursively filled by updating its columns as[3]:

$$\mathbf{v}_t = \max\{\mathbf{a} \odot \mathbf{p} + \tilde{\mathbf{V}}_{t-1}\mathbf{P}, \text{axis} = 1\}$$

Note that the two terms in the sum above don't have the same dimensions. The operation assumes that we "broadcast" the first term to match the dimensions of the second. The resulting value matrix is:

$$\mathbf{V} = \begin{pmatrix} \mathbf{v}_0 & \dots & \mathbf{v}_{500} \end{pmatrix} = \begin{pmatrix} v_0(0) & v_{500}(0) \\ \vdots & \vdots \\ v_0(100) & v_{500}(100) \end{pmatrix}$$

The optimal policy matrix α^* is filled in the similar a similar way, updating the columns as

$$\alpha_t = \arg\max\{\mathbf{a} \odot \mathbf{p} + \tilde{\mathbf{V}}_{t-1}\mathbf{P}, \text{axis} = 1\}$$

[3]Note that the result of the operation $\mathbf{a} \odot \mathbf{p} + \tilde{\mathbf{V}}_{t-1}\mathbf{P}$ is a 100×3 matrix. To get a valid update for the column vector $\mathbf{v}_t$ we take the maximum value along the columns, resulting in a 100×1 column vector.

All this can be done in Python as

```
import numpy as np

MAX_ITEMS = 100
MAX_T = 500

actions = np.asarray([200.0, 100.0, 50.0])
p = np.asarray([0.1, 0.5, 0.8])
p_matrix = np.asarray([p, 1 - p])
v_matrix = np.zeros(shape=(MAX_ITEMS + 1, MAX_T + 1))
alpha_matrix = np.ones(shape=v_matrix.shape) * 99

for t in range(1, MAX_T + 1):
    a = (p * actions) + np.asarray([v_matrix[:-1, t - 1], \
        v_matrix[1:, t - 1]]).T.dot(p_matrix)
    v_matrix[1:, t] = np.max(a, axis=1)
    alpha_matrix[1:, t] = np.argmax(a, axis=1)
```

A.2 Simulation under optimal policy

The following is the implementation of the algorithm, explained in **Question C**, to simulate the dynamic decision problem using the given policy passed as parameter.

```
def return_simulated_rewards(alpha_matrix, params):
    ran_vals = np.random.rand(1000, 500)
    actions = params["actions"]
    p = params["p"]

    def _return_simulated_reward(ran_vals):
        items = 100
        t = 500
        reward = 0
        while items > 0 and t > 0:
            if ran_vals[t - 1] < p[int(alpha_matrix[items, t])]:
                reward += actions[int(alpha_matrix[items, t])]
                items -= 1
                t -= 1
            else:
                t -= 1
        return reward

    return np.asarray([_return_simulated_reward(ran_val) for ran_val in ran_vals])
```

A.3 Two-dimensional state space problem

The following algorithm is an adaptation of the one presented in section A.1 to solve **Question E**. Here we compute the value matrix values in a different way for each α_{last} of the state space.

```
import numpy as np

MAX_ITEMS = 100
MAX_T = 500
actions = np.asarray([200.0, 100.0, 50.0])
p = np.asarray([0.1, 0.5, 0.8])
p_matrix = np.asarray([p, 1 - p])
num_of_actions = len(actions)
v_matrix = np.zeros(shape=(MAX_ITEMS + 1, num_of_actions, MAX_T + 1))
alpha_matrix = np.empty(shape=v_matrix.shape)
alpha_matrix[:] = np.nan
for t in range(1, MAX_T + 1):
    last_50 = (p[2] * actions[2]) + np.asarray(
        [v_matrix[:-1, 2, t - 1], v_matrix[1:, 2, t - 1]]
    ).T.dot(
        p_matrix[:, 2]
    )   # if the last action is 50 only consider the 50 states in the next step
    v_matrix[1:, 2, t] = last_50
    last_100 = np.column_stack(
        [
            (p[1] * actions[1])
            + np.asarray([v_matrix[:-1, 1, t - 1], v_matrix[1:, 1, t - 1]]).T.dot(
                p_matrix[:, 1]
            ),
            (p[2] * actions[2])
            + np.asarray([v_matrix[:-1, 2, t - 1], v_matrix[1:, 2, t - 1]]).T.dot(
                p_matrix[:, 2]
            ),
        ]
    )   # if the last action is 100 consider both 50 and 100 states in the next step
    v_matrix[1:, 1, t] = np.max(
        last_100, axis=1
    )
    last_200 = np.column_stack(
        [
            (p[0] * actions[0])
            + np.asarray([v_matrix[:-1, 0, t - 1], v_matrix[1:, 0, t - 1]]).T.dot(
                p_matrix[:, 0]
            ),
            (p[1] * actions[1])
            + np.asarray([v_matrix[:-1, 1, t - 1], v_matrix[1:, 1, t - 1]]).T.dot(
                p_matrix[:, 1]
            ),
            (p[2] * actions[2])
            + np.asarray([v_matrix[:-1, 2, t - 1], v_matrix[1:, 2, t - 1]]).T.dot(
                p_matrix[:, 2]
            ),
        ]
    )   # if the last action is 200 consider all states in the next step
    v_matrix[1:, 0, t] = np.max(
        last_200, axis=1
    )
```

```
alpha_matrix[1:, 2, t] = 2
alpha_matrix[1:, 1, t] = np.argmax(last_100, axis=1) + 1
alpha_matrix[1:, 0, t] = np.argmax(last_200, axis=1)
```

B Figures

Figure 1: Optimal policy when all actions are possible at any given state.

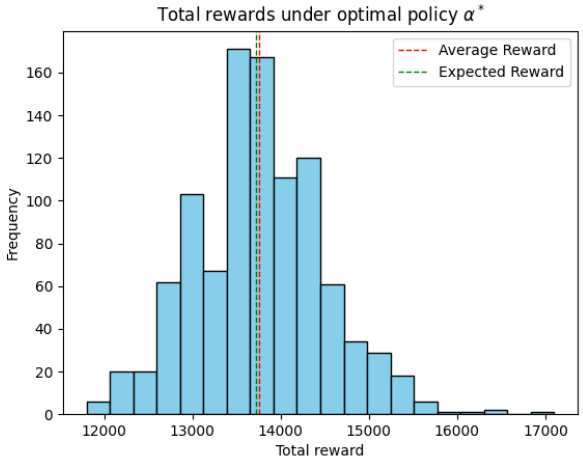

Figure 2: Total expected rewards when all actions are possible. The red dotted line represents the average total reward obtained in the simulations following the optimal policy (13751.0) and the green dotted line the total expected reward (13715.8). In the limit $N \to \infty$, these two numbers will be exactly the same, where N is the number of simulations.

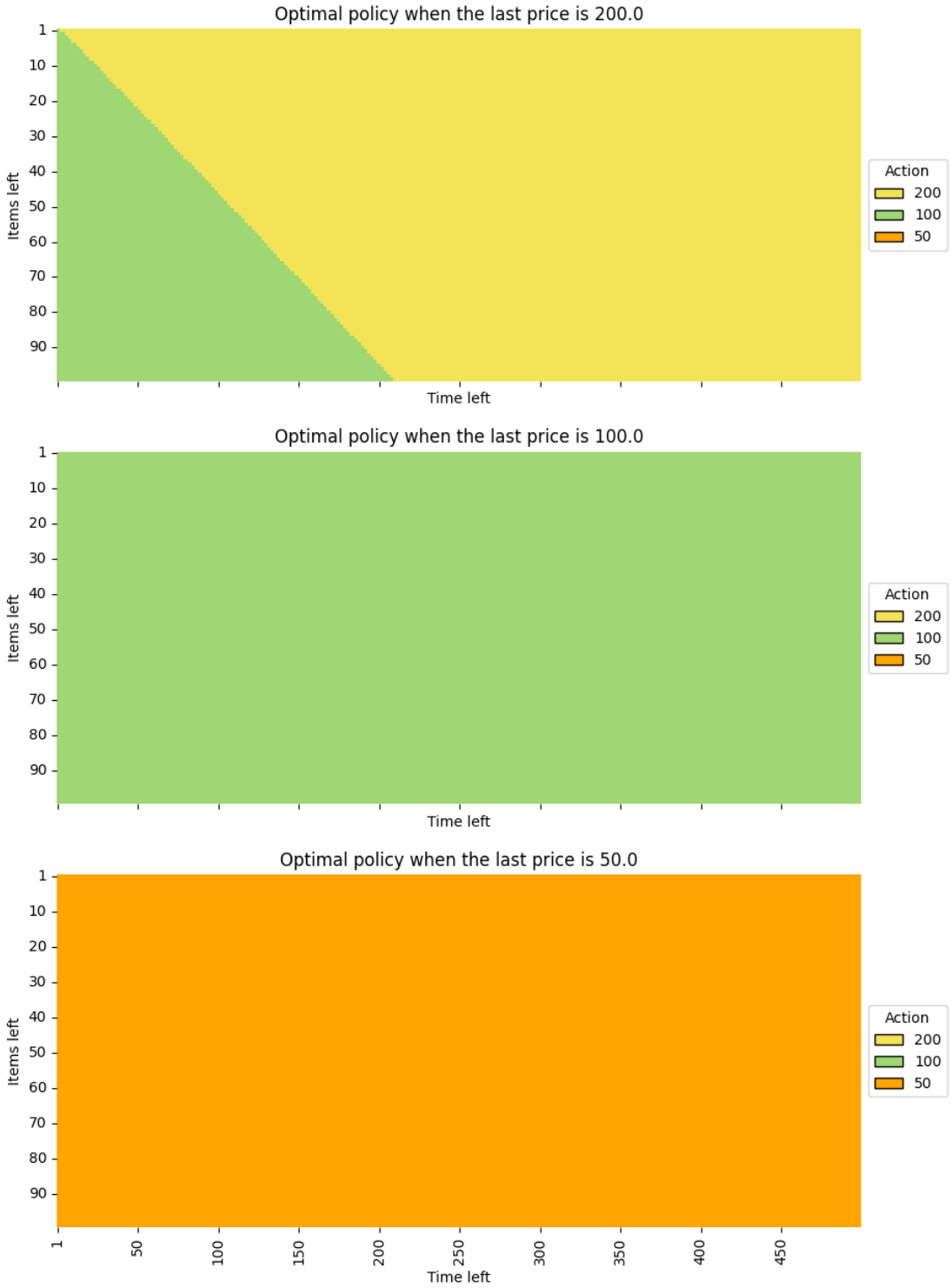

Figure 3: Optimal policy when we cannot increase the last set price.