Assignment 2

Michele Vannucci (2819493)
Arturo Abril (2813498)

November 2024

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

Notation *We will use a non-standard notation with vectors for the sake of clarity and compactness. Vectors are in bold, as usually, so a two-dimensional vector* $\mathbf{x} = (x_1, x_2)^\intercal$ *with* x_1, x_2 *real numbers. When we write operations involving a scalar and a vector, we assume the operation is done element-wise. For example:* $\mathbf{x} - a \equiv (x_1 - a, x_2 - a)^\intercal$ *for* $a \in \mathbb{R}$*. This holds also for inequalities involving vectors and scalars. For example:* $\mathbf{x} \leqslant a \equiv (x_1 \leqslant a, x_2 \leqslant a)^\intercal$*. We also define the following for the indicator function involving an inequality with a vector and a scalar in the condition:* $\vec{\mathbf{1}}_{\{\mathbf{x}=a\}} \equiv (\mathbf{1}_{\{x_1=a\}}, \mathbf{1}_{\{x_2=a\}})^\intercal$ *for* $a \in \mathbb{R}$*.*

Question A The states $\mathbf{x} \in \mathcal{X}$ are all possible combinations of remaining items in the inventory. The maximum number of items possible is 20 and the minimun is 1, for both item types. We can define the state space $\mathcal{X}$ formally as:

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{N}^2 \mid 1 \leqslant \mathbf{x} \leqslant 20\}$$

where $\mathbf{x} = (x_1, x_2)^\intercal$, with $x_1, x_2 \in \mathbb{N}$, and x_1 represents the number of items left for item of type 1. Same for 2. Note that since the states are two dimensional, $|\mathcal{X}| = 20 \cdot 20 = 400$.
The actions $\mathbf{a} \in \mathcal{A}_\mathbf{x}$ are the different order sizes for a given state $\mathbf{x}$. They depend on the state, so the number of items remains bounded as described above. Formally:

$$\mathcal{A}_\mathbf{x} = \{\mathbf{a} \in \mathbb{N}^2 \mid \vec{\mathbf{1}}_{\{\mathbf{x}=1\}} \leqslant \mathbf{a} \leqslant 20 - \mathbf{x}\}$$

where, as for the states, $\mathbf{a} = (a_1, a_2)^\intercal$, with $a_1, a_2 \in \mathbb{N}_0$ and a_1 represents the number of items of type 1 ordered. Same for type 2.

Question B If we fix the policy, namely α, as described in the assignment, the transition probabilities are[2]:

$$p_\alpha(\mathbf{y} \mid \mathbf{x}) = \begin{cases} \frac{1}{4}, & \text{if} \quad \mathbf{x} - 1 + 4 \cdot \vec{\mathbf{1}}_{\{\mathbf{x}=1\}} \leqslant \mathbf{y} \leqslant \mathbf{x} + 4 \cdot \vec{\mathbf{1}}_{\{\mathbf{x}=1\}} \\ 0, & \text{o.w.} \end{cases} \tag{1}$$

To clarify: for each state $\mathbf{x}$ there are just four non-zero transition probabilities. These correspond to a decrease of the type 1 item in 0 or 1 units. Same for type 2. All are equal to $1/4$. This $1/4$ arises from multiplying the transition probabilities of x_1 and x_2. We can do this because the demand of those is independent of each other. To account for the case in which there is just one unit left of either of the items, we introduce the indicator function (per component of $\mathbf{x}$) to top up the inventory of items to 5. In this case 5 becomes the upper bound for the next state and 4 is the lower bound, as indicated in the inequality.

Question C We run the simulation by computing the running cost per every possible initial state in parallel, this is stored in a Matrix $\mathbf{C}$, where every element $\mathbf{C}_{i,j}$ holds the value of the running cost for the initial state $\mathbf{x} = (i, j)^\intercal$ where i and j represent the quantity of first and second product type respectively. Similarly a matrix $\mathbf{S}$ is used to store the current state of every initial state running cost, in this case the elements of the matrix, $\langle \mathbf{S}_{i,j,1}, \mathbf{S}_{i,j,2} \rangle$ store the amounts of product of the first and second type respectively. We run the simulation for 10,000 time-steps. At

[1] https://github.com/michelexyz/RL-assignments/tree/main/Assignment2

[2] The inequality in the "if" condition may be a bit confusing, since according to the notation used so far it would be extended to both components of the vectors involved. The "if" condition is satisfied if **all** vectors components satisfy the inequality.

each time step, the running costs $\mathbf{C}$ are updated to account for the holding cost of both items. The updated costs are computed with the formula: $\mathbf{C}_{i,j}^{(t+1)} = \mathbf{C}_{i,j}^{(t)} + \mathbf{S}_{i,j,1} + 2 \cdot \mathbf{S}_{i,j,2}$, here $\mathbf{C}_{i,j}^{(t)}$ represents the running cost at time t. Similarly, the order costs are considered if items with a quantity of 1 are found in the $\mathbf{S}$ matrix and the latter is then updated by setting them to 5. Finally, the demand is simulated and the states are update with the formula $\mathbf{S}_{i,j,p}^{(t+1)} = \mathbf{S}_{i,j,p}^{(t)} - 1$ if $R_{i,j,p,t+1} < 0.5$, where $R_{i,j,p,t+1}$ is a random value in the interval $[0; 1]$. The average return for every possible initial state are stored in the matrix $\phi_{i,j} = \frac{C_{i,j}^{t_{max}}}{t_{max}}$ where t_{max} is the total number of time steps. We get a similar value for every initial state as expected, in detail we get $\phi_{i,j} \approx 10,2 \quad \forall i, j$. A simple implementation of the simulation (choosing $(20, 20)^T$ as initial state) can be found in the Appendix, together with the vectorize implementation.

Question D To compute the limiting distribution we first need to construct the transition probability matrix $P = P_{xy}$, with the transition probabilities given by Eq. 1. This will be a 400×400 matrix, where the rows indicate the initial state and the columns the next state. The probability distribution vector $\boldsymbol{\pi}$ will be of size 400 and each component will correspond to one of the possible states of the system, so that $\boldsymbol{\pi}^T = (\pi_{1,1}, \pi_{1,2} \dots \pi_{20,20})$. Each subscript represents the number of items left for each item type. The detailed construction of this matrix can be found in tht Appendix. We then apply the formula:

$$\boldsymbol{\pi}_{t+1}^T = \boldsymbol{\pi}_t^T P$$

with an arbitrary $\boldsymbol{\pi}_0$ until convergence. The result is $\boldsymbol{\pi}_*$, which is the limiting distribution. The long run average costs can be computed now easily by multiplying $\boldsymbol{\pi}_*$ by the expected costs $c(\mathbf{x})$. The expected costs for each state $\mathbf{x}$ are just the holding costs for that state plus an extra cost of 5 if either x_1 or x_2 is equal to one. The costs can also be stored in a vector $\mathbf{c}$ of the same dimensions as $\boldsymbol{\pi}_*$ so the long-run average costs are then $\phi_* = \mathbf{c} \cdot \boldsymbol{\pi}_*$. We get a value of 10.172 for every element of ϕ_*.

Question E Before defining the Poisson equation we need to introduce the value function for this problem, $V_t(\mathbf{x})$. This gives the total expected costs for a given state $\mathbf{x}$ when t time steps are taken. As before, we can construct a vector $\mathbf{V}$ with the same dimensions as $\boldsymbol{\pi}$ that stores, for a given t, the total expected costs of every state, so that $\mathbf{V} = (V_{11} \dots V_{2020})$. With this, the Poisson equation is:

$$\mathbf{V} + \boldsymbol{\phi} = \mathbf{c} + P\mathbf{V}$$

Where $\boldsymbol{\phi}$ is just a vector of the right dimensions with all its entries equal to ϕ. We solve the equation using value iteration, we apply $\mathbf{V}_{t+1} = \mathbf{c} + P\mathbf{V}_t$ until $\mathbf{V}_{t+1} - \mathbf{V}_t \leqslant \epsilon$ for $\epsilon = 10^{-5}$. When convergence is reached, $\mathbf{V}_* = \mathbf{V} - \min\{\mathbf{V}\}$ and $\boldsymbol{\phi}_* = \mathbf{c} + P\mathbf{V} - \mathbf{V}$. We get a value of 10.172 for ϕ_*. As expected, results for Questions **c**, **d** and **e** are the same.

Question F The bellman equation is defined as $\mathbf{V} + \phi e = \min_\alpha\{\mathbf{r}_\alpha + P_\alpha \mathbf{V}\}$ since this is a non-linear equation we solve it through value iteration as: $\mathbf{V}_{t+1} = \min_\alpha\{\mathbf{r}_\alpha + P_\alpha \mathbf{V}_t\}$. Since we have $|A_\mathbf{x}| \leqslant 400$ possible actions for every state $\mathbf{x}$, we compute the iteration element wise for every $V(\mathbf{x})$ as: $V(\mathbf{x})_{t+1} = \min_\mathbf{a}\{r_\mathbf{a} + P_\mathbf{a} V(\mathbf{x})_t\}$. This means that we minimize the value function for every state separately, over the action space for that state. After 1000 iterations we compute ϕ_{α_*} as $V(\mathbf{x})_{t+1} - V(\mathbf{x})_t$, and we get the same result for every $\mathbf{x}$: $\phi_{\alpha_*} \approx 7.99$. Figure 1 shows the values of $\mathbf{V}_{\alpha_*}$ computed as $V_{1000}(\mathbf{x}) - \min_\mathbf{x}\{V_{1000}(\mathbf{x})\}$.

Question G Figure 2 shows the optimal policy. The optimal order quantities are displayed separately for the two products for each given state. The optimal policy is to order only when we have to as expected. In fact, we order only when one of the two products has a quantity of 1. This is optimal since we want to have the least amount of products in stock while still minimizing the amount of orders. We can characterize α as:

$$\mathbf{a}_\alpha = \begin{cases} (2,1) & \text{if} \quad x_1 = 1 \ \ \& \ \ x_2 = 2 \\ (1,2) & \text{if} \quad x_1 = 2 \ \ \& \ \ x_2 = 1 \\ 2 \cdot \vec{\mathbf{1}}_{\{\mathbf{x}=1\}} & \text{o.w.} \end{cases}$$

We would have expected for the order quantities to be different since the holding costs are different, we motivate this by saying that this difference might not be large enough to affect the policy. We tried increasing the holding cost of product 2 to 5 and saw that this caused the optimal policy to deviate, so that only one item of the second type was ordered each time. Finally, we can observe that when one of the two products has a quantity of one and the other has a quantity of 2, the policy takes advantage of the current order by ordering one of the other item too.

A Appendix

A.1 Simulation under fixed policy

```
MAX_ITEMS = 20
MAX_ORDER = 5  # Policy fixed to replenish up to 5
PI_0 = np.array((20, 20))  # Initial state (irrelevant)
H = np.array([1, 2])  # Holding costs for each item type
K = 5  # Order costs

def question_c(n=10_000):
    """Simulation"""
    # The initial state doesn't matter
    pi = np.asarray(PI_0)
    # Probs of selling
    probs = np.random.randint(2, size=(2, n))

    # Initialize total cost and visits
    cost = pi @ H
    visits = np.zeros((MAX_ITEMS, MAX_ITEMS))
    visits[*(pi - 1)] += 1

    # Loop over simulations
    for i in range(n):
        # FIRST order (before any potential sales can take place)
        order = MAX_ORDER - pi if np.any(pi == 1) else 0
        # THEN demand
        pi -= probs[:, i]
        # Order arrives at the end of timestep
        pi += order
        # Update cost
        cost += K * np.any(order) + pi @ H
        # Update visits
        visits[*(pi - 1)] += 1

    return cost / n, visits.ravel() / n
```

A.1.1 Simulation, matrix implementation

We simulated the policy also through a matrix implementation that considers every possible initial state and stores the running total cost for each. With a number of iterations big enough, 10.000, we observed that the average cost was the same for every initial state as stated by the communicating chains theorem. The implementation in Python is the following:

```
# consider all possible initial states
running_costs = np.zeros((20,20))
current_state = np.zeros((20,20,2))

# initialize the current state
for i in range(1,21):
    for j in range(1,21):
        current_state[i-1,j-1] = [i,j]

n_timesteps = 10000

# a matrix to store the random values for the simulation
random_values = np.random.rand(20,20,2,n_timesteps)
```

```
for t in range(n_timesteps):

    # simulate the holding costs
    # the holding costs are 1 for the first product and 2 for the second product
    running_costs += current_state[:,:,0] + 2*current_state[:,:,1]

    ## all the elements in the current state that are 1 should be ordered so that the inventory level of b
    running_costs[(current_state[:,:,0] == 1) | (current_state[:,:,1] == 1)] += 5
    current_state[current_state == 1] = 5

    ## simulate the demand for the products
    current_state[:,:,:] -= (random_values[:,:,:,t] < 0.5).astype(int)

average_costs = running_costs / n_timesteps
```

A.2 Construction of transition probability matrix under fixed policy.

```
def compute_prob_fixed(x1, x2, y1, y2):
    cond1 = x1 - 1 + (4 if x1 == 1 else 0) <= y1 <= x1 + (4 if x1 == 1 else 0)
    cond2 = x2 - 1 + (4 if x2 == 1 else 0) <= y2 <= x2 + (4 if x2 == 1 else 0)
    return 1 / 4 if cond1 and cond2 else 0

def prob_matrix_fixed():
    p = np.zeros((MAX_ITEMS**2, MAX_ITEMS**2))
    for i in range(MAX_ITEMS**2):
        for j in range(MAX_ITEMS**2):
            x1, x2 = divmod(i, MAX_ITEMS)
            y1, y2 = divmod(j, MAX_ITEMS)
            x1, x2, y1, y2 = (v + 1 for v in (x1, x2, y1, y2))  # Indices start at 0
            p[i, j] = compute_prob_fixed(x1, x2, y1, y2)
    assert (np.sum(p, axis=1) == 1).all()
    return p
```

A.3 Bellman equation

The following is an approximation of V_{α_*} that we get after 1000 value iterations through the Bellman equation as detailed in **Question F**. We visualize it as a 2d Matrix.

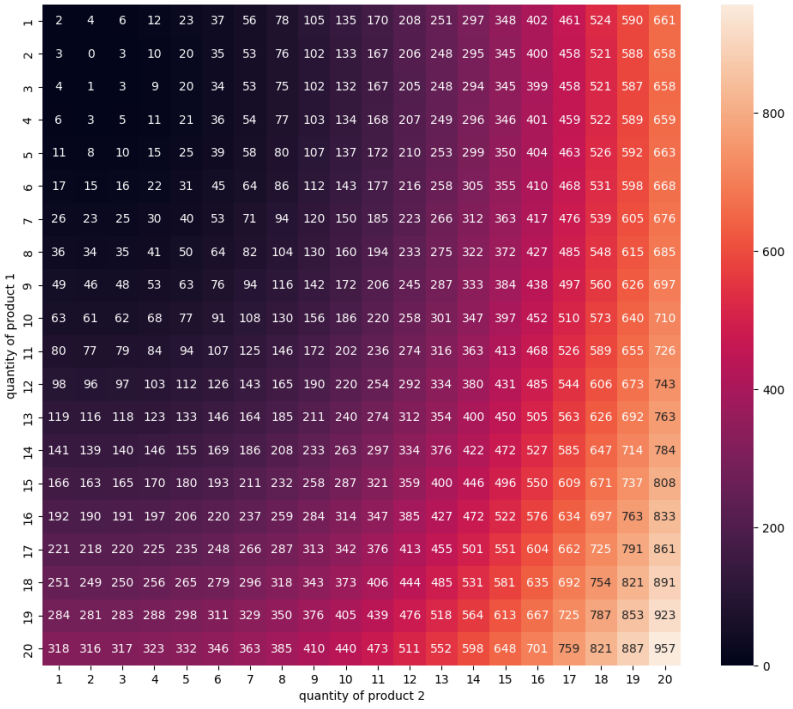
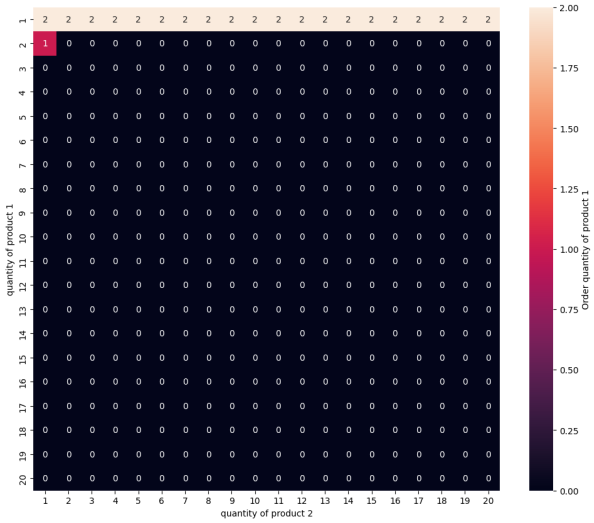
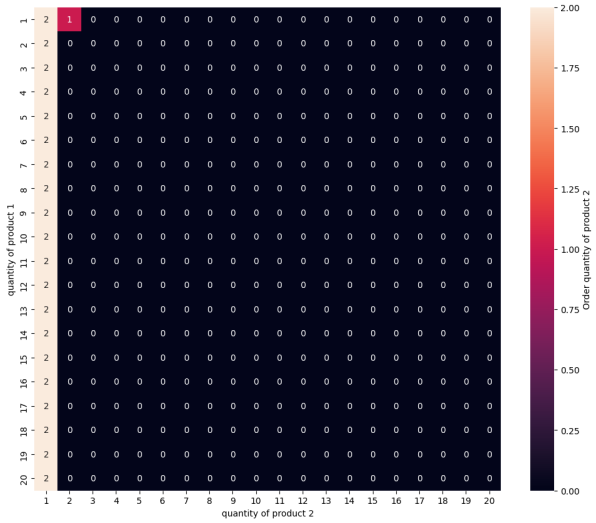


Figure 1: V_{α_*} obtained for **Question F**

The following is the derived optimal policy α_*.



(a) Order quantities within the optimal policies for product 1



(b) Order quantities within the optimal policies for product 2

Figure 2: Optimal policy represented as two heat-maps, showing the order quantity for each product type given the current state.