project_test_pdf

March 31, 2024

CASE STUDY: AIRLINE TICKET SALES

Christopher Lee

26 March 2024

0.0.1 1. Problem

Assume flight AA 001 departs in the evening of May 15. It has seat capacity 10. The airline starts to sell tickets on May 1 and ends on May 15. So, there are 15 days on which tickets to available seats can be sold. On each day two types of tickets can be sold: cheap for 100 dollars and expensive for 200 dollars. On each day airline allocates at most two tickets for sale, and it has to decide how many cheap and expensive tickets (out of the total of at most 2) should be offered for sale on each of the 15 days. On any day, the airline cannot allocate more tickets for sale than there are unsold tickets. It is known that the number of ticket requests that arrive on each day is random and has the following probability distribution:

It is known that the number of ticket requests that arrive on each day is random and has the following probability distribution:

$$P(X = x) = \begin{cases} \frac{1}{4} & \text{for } x = 2, \\ \frac{1}{2} & \text{for } x = 1, \\ \frac{1}{4} & \text{for } x = 0. \end{cases}$$

It is also known that the numbers of requests are independent across days.

On each day, here is how the ticket sale works. If the number of requests exceeds the number of offered tickets, the excess requests are discarded. The remaining requests (whose number does not exceed the number of allocated tickets for sale), first, certainly buy whatever cheap tickets available and, second, leftover requests buy expensive tickets with probability 0.4 and don't buy with probability 0.6.

- Example 1: Airline allocates one cheap and one expensive ticket for sale on a given day. If two requests arrive, one certainly buys the cheap ticket and another one buys expensive ticket with prob. (w.p.) 0.4 and does not buy anything w.p. 0.6. If one request arrives, it will buy cheap ticket. If no requests arrive, nobody buys tickets, of course.
- Example 2: Airline allocates one expensive ticket for sale on a given day. If two requests arrive, one certainly goes away without buying anything, and another one buys the expensive ticket with prob. (w.p.) 0.4 and does not buy anything w.p. 0.6. If one request arrives, it buys the expensive ticket with prob. (w.p.) 0.4 and does not buy anything w.p. 0.6.

- Example 3: Airline allocates two expensive tickets for sale on a given day. If two requests arrive, each of them will buy an expensive ticket with prob. (w.p.) 0.4 and will not buy anything w.p. 0.6. If one request arrives, it buys an expensive ticket with prob. (w.p.) 0.4 and does not buy anything w.p. 0.6.
- Example 4: Airline allocates one cheap ticket for sale on a given day. If two requests arrive, one certainly buys the cheap ticket and another goes away without buying anything. If one request arrives, it buys the cheap ticket

Project goal: Find an optimal ticket sale policy, to maximize the expected total revenue. No discount, i.e. the discount factor $= 1$.

0.0.2 2. Model

The ticket sale process is modeled as a Markov Decision Process (MDP), X_n, where n represents time. The state of this process is defined as the number of remaining (unsold) tickets, with the state space being $\{0, 1, ..., 10\}$. The time horizon for this process is $n = 0, 1, ..., 14$, where $n = 0$ corresponds to May 1 and $n = 14$ corresponds to May 15. The action taken at each time step is denoted by $k = (p, q)$, where p and q represent the number of cheap and expensive tickets offered for sale, respectively. It is important to note that both p and q are non-negative integers and the sum $p + q \leq 2$. The set of possible actions is defined as $A = \{(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2)\}$. However, not all actions can be taken in every state due to the constraint that more tickets cannot be allocated for sale than what is left. This constraint must be taken into account when modeling the MDP. For the sake of clarity in coding, the actions $(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2)$ can be enumerated as $0, 1, 2, 3, 4, 5$, respectively. Nevertheless, for explanatory purposes, actions will continue to be referred to by their pair notation (p, q).

What is the reward corresponding to each action k? The reward corresponding to each action k, denoted as $g_k(i)$, represents the expected revenue produced by taking action k in state i. The availability of actions depends on the current state i: - If $i \geq 2$, any action can be taken. - If $i = 1$, only actions $(0, 0)$, $(1, 0)$, and $(0, 1)$ are available. - If $i = 0$, only action $(0, 0)$ is available.

The reward $g_k(i)$ does not depend on i as long as action k is available in state i.

Example: Calculating $g(1, 1)$ Consider the reward $g(1, 1)$ of taking action $(1, 1)$ in any state $i \geq 2$. Given the probabilities of selling 2 tickets $(\frac{1}{4})$, 1 ticket $(\frac{1}{2})$, and 0 tickets $(\frac{1}{4})$, the expected revenue can be calculated as follows:

$$g(1, 1) = \left(\frac{1}{4}\right) \cdot 0 + \left(\frac{1}{2}\right) \cdot 100 + \left(\frac{1}{4}\right) \cdot [100 + 0.4 \cdot 200 + 0.6 \cdot 0] = 95.$$

This calculation assumes that selling one cheap ticket yields 100 in revenue, and selling one expensive ticket can yield either 200 or 0, with probabilities of 0.4 and 0.6, respectively.

We will have to compute the rewards for all other actions

What about transition probabilities $P_k(i, j)$ **for each action k?** The transition probabilities $P_k(i, j)$ describe the probability of transitioning from state i to state j given action $k = (p, q)$. This action can be applied in state s only if $i \geq p + q$. If the number of tickets for sale $p + q = 2$, then transitions from i are possible to states i, $i - 1$, and $i - 2$.

Example: Calculating $P_{(1,1)}(i,j)$ for Action $(1,1)$ For action $k=(1,1)$, the transition probabilities from state i given the number of tickets sold are as follows:

- $P_{(1,1)}(i,i) = \frac{1}{4} = 0.25$; This represents the probability of selling no tickets.
- $P_{(1,1)}(i,i-1) = \frac{1}{2} + \left(\frac{1}{4} \cdot 0.6\right) = 0.65$; This accounts for the probability of selling exactly one ticket, either a cheap or an expensive one with a 60% chance of the latter not being sold.
- $P_{(1,1)}(i,i-2) = \left(\frac{1}{4} \cdot 0.4\right) = 0.1$; This is the probability of selling both tickets, with a 40% chance of the expensive ticket being sold.

These transition probabilities are applicable for any state $i \geq 2$, reflecting the outcomes of selling 0, 1, or 2 tickets when offering one cheap and one expensive ticket.

- If the number of tickets for sale $p+q=1$, then transitions from i are possible to states i, $i-1$.
- If the number of tickets for sale $p+q=0$, i.e. the action is $(0,0)$, then, of course $P_{(0,0)}(i,i)=1$

We will have to compute the transition probabilities for all actions.

0.0.3 Main Assignment:

Write a python code, which computes the optimal policy, maximizing the total expected revenue. (You also have to calculate the rewards and transition probabilities above – but for that you do not have to write a code, can just compute and explain numbers in your report.) The output of the code has to show an optimal policy, i.e. action which should be taken, depending on time and state; a convenient way to show this is a matrix. It also has to show the optimal value function; this is also convenient to show as a matrix

You must submit BOTH the report explaining what you do and how, and the working code. Just a report, without code, does NOT give you a partial credit.

[ ]:
```
import pandas as pd
import numpy as np
```

[ ]:
```
NUM_DAYS = 15
NUM_TICKETS = 10
PROBS = np.array([0.25, 0.5, 0.25]) # P(X = x) = {1/4 for x=0, 1/2 for x=1, 1/4␣
 ↪for x=2}
EXPENSIVE_TICKET_PROB = 0.4
CHEAP_TICKET_PROB = 0.6
CHEAP_PRICE = 100
EXPENSIVE_PRICE = 200
ACTIONS = [(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2)]
```

0.0.4 Computing the rewards for all other actions:

[ ]:
```
g_00 = 0

# (1, 0) -> 1 cheap ticket
g_10 = PROBS[1] * CHEAP_PRICE + PROBS[2] * CHEAP_PRICE
```

```
# (0, 1) -> 1 expensive ticket
g_01 = (PROBS[1] + PROBS[2]) * EXPENSIVE_TICKET_PROB * EXPENSIVE_PRICE

# (1, 1) -> 1 cheap ticket, 1 expensive ticket
g_11 = PROBS[1] * CHEAP_PRICE + PROBS[2] * (CHEAP_PRICE + EXPENSIVE_TICKET_PROB␣
 ↪* EXPENSIVE_PRICE)

# (2, 0) -> 2 cheap tickets
g_20 = PROBS[1] * CHEAP_PRICE + PROBS[2] * 2 * CHEAP_PRICE

# (0, 2) -> 2 expensive tickets
g_02 = PROBS[1] * EXPENSIVE_TICKET_PROB * EXPENSIVE_PRICE + PROBS[2] * 2 *␣
 ↪EXPENSIVE_TICKET_PROB * EXPENSIVE_PRICE

G_rewards = g_00, g_10, g_01, g_20, g_11, g_02
G_rewards
```

[ ]:
```
(0, 75.0, 60.00000000000001, 100.0, 95.0, 80.0)
```

0.0.5 Computing the transition probabilities for all actions:

[ ]:
```
P_matrices = {
    action: np.zeros((NUM_TICKETS + 1, NUM_TICKETS + 1)) for action in ACTIONS
}
P_matrices[(0, 0)] = np.eye(NUM_TICKETS + 1)


def fill_matrix(action):
    if action == (1, 0):
        for i in range(1, NUM_TICKETS + 1):
            P_matrices[action][i, i] = PROBS[0]
            P_matrices[action][i, i - 1] = PROBS[1] + PROBS[2]
    elif action == (0, 1):
        for i in range(1, NUM_TICKETS + 1):
            P_matrices[action][i, i] = (
                PROBS[0] + PROBS[1] * CHEAP_TICKET_PROB + PROBS[2]
                    * CHEAP_TICKET_PROB
            )
            P_matrices[action][i, i - 1] = (
                PROBS[1] * EXPENSIVE_TICKET_PROB + PROBS[2] *␣
 ↪EXPENSIVE_TICKET_PROB
            )
    elif action == (2, 0):
        for i in range(1, NUM_TICKETS + 1):
            P_matrices[action][i, max(i - 2, 0)] = PROBS[2]
            if i == 1:
                P_matrices[action][i, i - 1] = PROBS[1] + PROBS[2]
```

```
            if i > 1:
                P_matrices[action][i, i - 1] = PROBS[1]
            P_matrices[action][i, i] = PROBS[0]
    elif action == (1, 1):
        for i in range(2, NUM_TICKETS + 1):
            P_matrices[action][i, i - 2] = PROBS[2] * EXPENSIVE_TICKET_PROB
            P_matrices[action][i, i - 1] = PROBS[1] + PROBS[2] *␣
 ↪CHEAP_TICKET_PROB
            P_matrices[action][i, i] = PROBS[0]
        P_matrices[action][1, 0] = PROBS[1] + PROBS[2]  # Special case for i=1
        P_matrices[action][1, 1] = PROBS[0]
    elif action == (0, 2):
        for i in range(2, NUM_TICKETS + 1):
            P_matrices[action][i, i - 2] = PROBS[2] * EXPENSIVE_TICKET_PROB**2
            P_matrices[action][i, i - 1] = (
                PROBS[1] * EXPENSIVE_TICKET_PROB
                + 2 * PROBS[2] * EXPENSIVE_TICKET_PROB * CHEAP_TICKET_PROB
            )
            P_matrices[action][i, i] = (
                PROBS[0]
                + PROBS[1] * CHEAP_TICKET_PROB
                + PROBS[2] * CHEAP_TICKET_PROB**2
            )
        P_matrices[action][1, 0] = (
            PROBS[1] * EXPENSIVE_TICKET_PROB + PROBS[2] * EXPENSIVE_TICKET_PROB
        )  # Special case for i=1
        P_matrices[action][1, 1] = (
            PROBS[0] + PROBS[1] * CHEAP_TICKET_PROB + PROBS[2] *␣
 ↪CHEAP_TICKET_PROB
        )

    P_matrices[action][0, 0] = 1


[fill_matrix(action) for action in ACTIONS]

for action in ACTIONS:
    print(f"Transition Probabilities for Action: {action}:")
    print(P_matrices[action])
    print("\n ----------------------------------------------------------------␣
 ↪\n")
```

```
Transition Probabilities for Action: (0, 0):
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]

 ------------------------------------------------------------

Transition Probabilities for Action: (1, 0):
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.75 0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.75 0.25 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.75 0.25 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.75 0.25 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.75 0.25 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.75 0.25 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.75 0.25 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.75 0.25 0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.75 0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.75 0.25]]

 ------------------------------------------------------------

Transition Probabilities for Action: (0, 1):
[[1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.3 0.7 0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.3 0.7 0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.3 0.7 0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.3 0.7 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.3 0.7 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.3 0.7 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.3 0.7 0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.3 0.7 0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.7 0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.7]]

 ------------------------------------------------------------

Transition Probabilities for Action: (2, 0):
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.75 0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.25 0.5  0.25 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.25 0.5  0.25 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.25 0.5  0.25 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.25 0.5  0.25 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.25 0.5  0.25 0.   0.   0.   0.  ]
```

```
 [0.   0.   0.   0.   0.   0.25 0.5  0.25 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.25 0.5  0.25 0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.25 0.5  0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.25 0.5  0.25]]

 ------------------------------------------------------------

Transition Probabilities for Action: (1, 1):
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.75 0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.1  0.65 0.25 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.1  0.65 0.25 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.1  0.65 0.25 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.1  0.65 0.25 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.1  0.65 0.25 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.1  0.65 0.25 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.1  0.65 0.25 0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.1  0.65 0.25 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.1  0.65 0.25]]

 ------------------------------------------------------------

Transition Probabilities for Action: (0, 2):
[[1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.3  0.7  0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.04 0.32 0.64 0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.04 0.32 0.64 0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.04 0.32 0.64 0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.04 0.32 0.64 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.04 0.32 0.64 0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.04 0.32 0.64 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.04 0.32 0.64 0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.04 0.32 0.64 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.04 0.32 0.64]]

 ------------------------------------------------------------
```

0.0.6 Optimal Ticket Sale Policy:

Given: - A set of actions ($A = \{(0,0),(1,0),(0,1),(2,0),(1,1),(0,2)\}$) corresponding to selling combinations of cheap and expensive tickets. - Immediate rewards ($g(p,q)$) for actions ($(p,q) \in A$). - Transition probability matrices ($P_{(p,q)}$) for each action, dictating the probabilities of moving from state (i) to (j). - A horizon of ($N = 15$) days and a maximum of 10 tickets.

Objective: - To find the optimal policy ($\pi_n^*(i)$) and the maximum expected revenue ($V_n(i)$) for each day (n) and each state (i) (number of tickets left).

Procedure: 1. Initialize ($V_0^*(i)$) for all states (i) as zeros. 2. For each day (n) from 1 to (N), in

reverse order: - For each state (i) (number of tickets left): - Calculate the value of taking each action $((p, q) \in A)$ from state (i), considering both the immediate reward and the expected future revenue, constrained by the action being feasible $((p + q \leq i))$. - Select the action that maximizes this value. 3. Store the optimal action $(\pi_n^*(i))$ and the updated maximum expected revenue $(V_n^*(i))$.

For each state (i) and day (n), we perform:

$$V_n^*(i) = \max_{(p,q)\in A} \left\{ g(p,q) + \sum_{j=0}^{10} P_{(p,q)}(i,j) \cdot V_{n-1}^*(j) \right\}$$

$$\pi_n^*(i) = \text{argmax}_{(p,q)\in A} \left\{ g(p,q) + \sum_{j=0}^{10} P_{(p,q)}(i,j) \cdot V_{n-1}^*(j) \right\}$$

$(V_n^*(i))$ is the expected revenue with (n) days remaining and (i) tickets left, and $(g(p,q))$ represents the immediate revenue from selling tickets according to action $((p,q))$. $(P_{(p,q)}(i,j))$ denotes the transition probability from state (i) to state (j) under action $((p,q))$.

[ ]:
```
optimal_ticket_sale_policy = {}

v_star = np.zeros(NUM_TICKETS + 1)

for day in range(NUM_DAYS):
    v = v_star.copy()
    a_star_list = []
    v_star = np.zeros(NUM_TICKETS + 1)
    for i in range(NUM_TICKETS + 1):
        v_i = np.zeros(len(ACTIONS))
        for j, action in enumerate(ACTIONS):
            if sum(action) <= i:
                v_i[j] = G_rewards[j] + np.dot(P_matrices[action][i], v)

        v_star[i] = np.max(v_i)
        a_star = np.argmax(v_i)
        a_star_list.append(ACTIONS[a_star])

    optimal_ticket_sale_policy[day] = (a_star_list, v_star.tolist())
```

[ ]:
```
optimal_action_df = pd.DataFrame({key: a[0]
                                  for key, a in optimal_ticket_sale_policy.
 ↪items()
                                 }).T
optimal_action_df = optimal_action_df.iloc[::-1]
optimal_action_df.index.name = 'm:'
optimal_action_df = optimal_action_df.round(1)
optimal_action_df = optimal_action_df.astype(str)
optimal_action_df = optimal_action_df.style.set_caption(
    "Optimal Ticket Sale Policy Action Matrix")
```

```
optimal_action_df
```

[ ]: <pandas.io.formats.style.Styler at 0x1a4ebfb7b30>

```
optimal_value_df = pd.DataFrame({key: a[1]
                                 for key, a in optimal_ticket_sale_policy.
 ↪items()
                                }).T
optimal_value_df = optimal_value_df.iloc[::-1]
optimal_value_df.index.name = 'm:'
optimal_value_df = optimal_value_df.round(2)
optimal_value_df = optimal_value_df.astype(str)
optimal_value_df = optimal_value_df.style.set_caption(
    "Optimal Ticket Sale Policy Value Matrix")

optimal_value_df
```

[ ]: <pandas.io.formats.style.Styler at 0x1a4ebf5e420>

0.0.7 Conclusions:

We found an optimal ticket sale policy, to maximize the expected total revenue w/ the discount factor $\alpha = 1$.

The ticket sale process is modeled as a Markov Decision Process (MDP), X_n, where n represents time. The state of this process is defined as the number of remaining (unsold) tickets, with the state space being $\{0, 1, ..., 10\}$. The time horizon for this process is $n = 0, 1, ..., 14$, where $n = 0$ corresponds to May 1 and $n = 14$ corresponds to May 15. The action taken at each time step is denoted by $k = (p, q)$, where p and q represent the number of cheap and expensive tickets offered for sale, respectively. Note that both p and q are non-negative integers and the sum $p + q \leq 2$. The set of possible actions is defined as $A = \{(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2)\}$

What is the reward corresponding to each action k?
$g_k(i)$, represents the expected revenue produced by taking action k in state i. $g_{0,0} = 0$ $g_{1,0} = 75$ $g_{0,1} = 60$ $g_{2,0} = 100$ $g_{1,1} = 95$ $g_{0,2} = 80$

So, the expected revenue from taking action - for exanple (1,1) - in any state where it's available (i.e. i 2) is \$95. This calculation demonstrates how the combination of the probability distribution of requests and the pricing strategy for the tickets influences the expected revenue for a given action.

What about transition probabilities $P_k(i, j)$ for each action k?

If $k = (0, 0)$,

$$P_k(i, i) = 1$$

If $k = (1, 0)$,

$$P_k(i, i) = \frac{1}{4},\ P_k(i, i-1) = \frac{3}{4}$$

If $k = (0, 1)$,

$$P_k(i, i) = \frac{7}{10},\ P_k(i, i-1) = \frac{3}{10}$$

If $k = (2, 0)$,

$$P_k(i, i) = \frac{1}{4},\ P_k(i, i-1) = \frac{1}{2},\ P_k(i, i-2) = \frac{1}{4}$$

If $k = (1, 1)$,

$$P_k(i, i) = \frac{1}{4},\ P_k(i, i-1) = 0.65,\ P_k(i, i-2) = 0.1$$

If $k = (0, 2)$,

$$P_k(i, i) = 0.64,\ P_k(i, i-1) = 0.32,\ P_k(i, i-2) = 0.04$$

Optimal actions $a_{(m)}(i)$ below: This matrix shows the optimal actions (in terms of how many cheap and expensive tickets to offer) for each day (m) and for each possible number of unsold tickets (0 to 10).

[ ]:
```
optimal_action_df
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x1a4ebfb7b30>
```

Optimal values $v_{(m)}(i)$ below: This matrix shows the expected total revenue (in dollars) that can be achieved from a given state (day and number of unsold tickets) by following the optimal policy.

[ ]:
```
optimal_value_df
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x1a4ebf5e420>
```