Dynamic Programming and Stochastic Control - Assignment 2

Zhenrui Zheng

Chinese University of Hong Kong, Shenzhen
225040512@link.cuhk.edu.cn

Contents

Exercise 1: Capacity Expansion over N Periods

1.1 Derivation of the DP Algorithm

Let $J_k(x_k)$ be the optimal expected cost to go from period k to the terminal period N, given that the capacity at the beginning of period k is x_k. The Bellman equation is:

$$
\begin{aligned}
J_k(x_k) &= \min_{u_k,\cdots,u_{N-1}} \mathbb{E}\left[-S(x_N) + \sum_{k=k}^{N-1}\big(C_k(u_k) + P_k(x_k+u_k-w_k)\big)\right] \\
&= \min_{u_k,\cdots,u_{N-1}} \mathbb{E}\left[C_k(u_k) + P_k(x_k+u_k-w_k) - S(x_N) + \sum_{k=k+1}^{N-1}\big(C_k(u_k) + P_k(x_k+u_k-w_k)\big)\right] \\
&= \min_{u_k} \mathbb{E}\left[C_k(u_k) + P_k(x_k+u_k-w_k) + \min_{\cdots,u_{N-1}} \mathbb{E}\left[-S(x_N) + \sum_{k=k+1}^{N-1}\big(C_k(u_k) + P_k(x_k+u_k-w_k)\big)\right]\right] \\
&= \min_{u_k} \mathbb{E}\left[C_k(u_k) + P_k(x_k+u_k-w_k) + J_{k+1}(x_k+u_k)\right]
\end{aligned}
$$

1.2 (s, S)-type Optimal Policy

From the Bellman equation in Section 1.1, we have:

$$J_k(x_k) = \min_{u_k \geq 0} \mathbb{E}\left[C_k(u_k) + P_k(x_k+u_k-w_k) + J_{k+1}(x_k+u_k)\right].$$

Let $y_k = x_k + u_k$ denote the capacity after expansion in period k. Then $u_k = y_k - x_k$ and the minimization becomes:

$$J_k(x_k) = \min_{y_k \geq x_k} \left[C_k(y_k - x_k) + \mathbb{E}[P_k(y_k - w_k)] + J_{k+1}(y_k)\right].$$

Define the function

$$G_k(y) = c_k y + \mathbb{E}[P_k(y - w_k)] + J_{k+1}(y),$$

where we note that $C_k(y - x_k) = K\mathbf{1}_{\{y > x_k\}} + c_k(y - x_k)$ for $y > x_k$, and $C_k(0) = 0$ for $y = x_k$.

Since S is concave with $\lim_{x\to\infty} dS(x)/dx = 0$, P_k are convex, and $c_k y + \mathbb{E}[P_k(y-w_k)] \to \infty$ as $|y| \to \infty$, the function $G_k(y)$ is convex and coercive. Therefore, $G_k(y)$ attains its global minimum at some point S_k^*.

The optimal policy is then:

- If $G(x_k) \geq G(S_k^*) + K$, expand to $y_k = S_k^*$ (i.e., $u_k = S_k^* - x_k$).
- If $G(x_k) < G(S_k^*) + K$, do not expand (i.e., $u_k = 0$).

This is precisely the (s, S)-type policy: when the cost of current capacity x_k falls below the threshold $G(S_k^*) + K$, maintain the current capacity; otherwise, expand to the target level S_k^*.

Exercise 2: Single-leg Revenue Management (Optimal Protection)

Omitting the derivation, we provide that the optimal control at stage $j+1$ is:

$$\mu^*_{j+1}(x, D_{j+1}) = \min\left\{(x - y^*_j)^+, D_{j+1}\right\}$$
$$y^*_j = \max\left\{x : 0 \le x \le C, p_{j+1} \le \Delta V_j(x)\right\}$$

The corresponding cost function is:

$$V_j(x) = \mathbb{E}\left[p_j \min\left\{(x - y^*_{j-1})^+, D_j\right\} + V_{j-1}\left(x - \min\left\{(x - y^*_{j-1})^+, D_j\right\}\right)\right]$$

Which can be implemented as follows:

1. Iterate over stage j from 1 to 10 (Backward).
2. For each stage j, compute the optimal level y^*_j with $O(1)$ amortized complexity.
3. Iterate over capacity x from 0 to C.
4. Compute the expected revenue $V_j(x)$, which is $O(C)$ complexity.

The result of optimal protection levels and total expected revenue are:

$$V_{10}(100) \simeq 35415.63, \quad y^*_j = \{6, 15, 26, 36, 47, 57, 69, 78, 91\}$$

An example of python implementation is provided below:

```python
import numpy as np
from scipy.stats import norm

# Normalized discretized truncated normal distribution
mu, sigma = 10, 2
p = np.zeros(21, dtype=np.float64)
for k in range(21):
  p[k] = norm.cdf((k+0.5-mu)/sigma) - norm.cdf((k-0.5-mu)/sigma)
p = p / np.sum(p)

C = 100
prices = np.array([0, 500, 480, 465, 420, 400, 350, 320, 270, 250, 200], dtype=np.float64)

V = np.zeros((11, C+1), dtype=np.float64)
y_max = 0
def binary_search(l, r, j):
  # find the largest x s.t. V[j, x] - V[j, x-1] >= prices[j+1]
  while l < r-1:
    m = (l + r) // 2
    if V[j, m]-V[j, m-1] >= prices[j+1]:
      l = m
    else:
      r = m
  return l

for j in range(1, 11):
  y_max = binary_search(y_max, C, j-1)
  print(f"y{j-1} = {y_max}")
  for x in range(C+1):
    for D in range(21):
      V[j, x] += p[D] * (prices[j] * min(max(x-y_max, 0), D) + V[j-1, x-min(max(x-y_max, 0), D)])

print(f"V[10, 100] = {V[10, 100]}")
```

Exercise 3: EMSR-b Heuristic

Following the description in the problem, we can compute the optimal protection levels and total expected revenue in a modified version:

1. Iterate over stage j from 1 to 10 (Backward).
2. For each stage j, compute the aggregated future demand $\Sigma_{k=1}^{j-1}\mathbb{E}[D_k]$ and weighted-average revenue $\bar{p}_{j-1} = \frac{\Sigma_{k=1}^{j-1} p_k \mathbb{E}[D_k]}{\Sigma_{k=1}^{j-1}\mathbb{E}[D_k]}$.
3. Since the demand is i.i.d. normal, $\mu_S = (j-1)\times\mu$ and $\sigma_S^2 = (j-1)\times\sigma^2$.
4. Compute the constant $z_\alpha = \Phi^{-1}(1 - p_j/\bar{p}_{j-1})$ and $y_j = \mu_S + z_\alpha \sigma_S$.
5. Iterate over capacity x from 0 to C.
6. Compute the expected revenue $V_j(x)$.

The result of optimal protection levels and total expected revenue are:

$$V_{10}(100) \simeq 35413.23, \quad y_j^* = \{6, 15, 26, 36, 47, 57, 68, 78, 90\} \quad \text{(rounded to nearest integer)}$$

An example of python implementation is provided below:

```
# ESMR-b
mu_sum = 0
sigma_sum = 0
total_revenue = 0
V = np.zeros((11, C+1), dtype=np.float64)
y_max = 0
for j in range(1, 11):
  if j > 1:
    mu_sum += mu
    sigma_sum += sigma**2
    total_revenue += prices[j-1] * mu
    p_bar = total_revenue / mu_sum
    z_alpha = norm.ppf(1-prices[j]/p_bar)
    y_max = mu_sum + z_alpha * sigma_sum**0.5
    y_max = math.ceil(y_max-0.5)
    print(f"y{j-1} = {y_max}")
  for x in range(C+1):
    for D in range(21):
      V[j, x] += p[D] * (prices[j] * min(max(x-y_max, 0), D) + V[j-1, x-min(max(x-y_max, 0), D)])

print(f"V[10, 100] = {V[10, 100]}")
```

Exercise 4: Parking Problem

Each parking spot is independently free with probability p. The cost of parking k spots away from the destination is k; reaching the destination without parking incurs cost C. Let F_k be the minimal expected cost when k spots from the destination, with $F_0 = C$.

4.1 Recursion

At distance k, if the current spot is free (probability p), the driver may

- park now, incurring cost k; or
- pass it and continue, incurring expected cost F_{k-1}.

If the spot is occupied (probability $q = 1 - p$), the driver must continue with cost F_{k-1}. Hence

$$F_k = p \min\{k,\ F_{k-1}\} + q\, F_{k-1}, \quad k = 1, 2, \ldots$$

and $F_0 = C$.

4.2 Threshold-Optimal Policy

From the recursion, when $k \le F_{k-1}$ it is better to park; otherwise it is better to continue. Therefore there exists a threshold $k^\star$ such that

if $k < k^\star$ then park at the first free spot; if $k \ge k^\star$ then never park.

By monotonicity and $F_0 = C$, we obtain

$$k^\star = \min\left\{ i \in \mathbb{Z}_{\ge 0} :\ q^i < \frac{1}{pC + q} \right\},$$

which is equivalent to the expression given in the exercise (note $q = 1 - p$). Thus the threshold parking policy is optimal.