Programowanie dynamiczne – Wyznaczanie optymalnej wielkości partii produkcyjnej

Zadanie 1

Ponieważ zadanie bazuje na metodzie z poprzedniego zadania rozwiązanie rozpocząłem od modyfikacji kodu opisanego w poprzednim sprawozdaniu.

```
def dynamic(g, h, q, y_min, y_max, y_begining, y_end):
    # g - koszt produkcji ustalonej liczby produktów
    # h - koszt składowania
    # q - zapotrzebowania miesięczne na dany produkt
    # y_max/y_min - maksymalna/minimalna zajętość magazynu
    # y_begining/y_end - y na początku/końcu

    # inicjalizacja x jako macierz zer
    # f jako macierz funkcji celu
    # y - wektor dopuszczalnych ilości przedmiotów w magazynie
    x_matrix = np.zeros((y_max - y_min + 1, len(q)))
    f = np.full((y_max - y_min + 1, len(q)), np.inf)
    y = np.arange(y_min, y_max+1)

    # i, j to indeksy z macierzy wart. funkcji celu/ macierzy decyzji
    # nie należy ich odnosić od indeksowania z wykładu
    # i - kolejny wiersz w macierzy, czyli kolejny stan w danym etapie, czyli również
    # kolejna wart. f. celu oraz kolejna decyzja (kolejny x)
    # j - kolejna kolumna mazierzy, czyli kolejny etap procesu - kolejny miesiąc
    for j in range(f.shape[1]):
        for i in range(f.shape[0]):
            # indeksowanie dla macierzy wejściowych
            # czyli dla j = 0, input_index = etap końcowy
            input_index = f.shape[1] - j - 1

           # wyliczenie przedziału dopuszczalnej produkcji
            if j == 0:
                x_max = x_min = y_end + q[input_index] - y[i]
            else:
                x_min = max(y_min + q[input_index] - y[i], 0)
                x_max = min(y_max + q[input_index] - y[i], len(g)-1)
            # jeżeli warunki nie są spełnione przejdź do kolejnej iteracji
            if x_min > x_max or x_max < 0 or x_min < 0 or x_max > len(g)-1:
                x_matrix[i, j] = None
                f[i, j] = np.inf
                continue

            # sprawdzenie wszyskich możliwych ilości x
            for possible_x in range(x_min, x_max+1):
                # obliczenie kosztu
                if j != 0:
                    # - y_min, aby dostosować funkcję przejścia do indeksowania
                    cost = g[possible_x] + h[y[i] + possible_x - q[input_index] - y_min] + \
                           f[y[i] + possible_x - q[input_index] - y_min, j-1]
                else:
                    # cost = g[possible_x] + h[y[i] + possible_x - q[input_index]]
                    cost = g[possible_x]
                # sprawdzenie czy koszt jest mniejszy lub równy
                if cost < f[i, j]:
                    # uaktualnienie
                    x_matrix[i, j] = possible_x
                    f[i, j] = cost
    return x_matrix, f
```

Nieco zmodyfikowałem również funkcję do wypisywania strategii optymalnej:

```
def get_results(q, y_min, y_begining, x_matrix, f):
    state = y_begining
    strategy = f"Total cost = {f[y_begining - y_min, -1]}\n"

    for j in range(f.shape[1] - 1, -1, -1):
        input_index = f.shape[1] - j - 1
        decision = int(x_matrix[state - y_min, j])
        strategy += f"|y{input_index} = {state}, x{input_index} =
{decision}|\n"
        state = int(state + decision - q[input_index])
    strategy += f"|y{len(q)} = {state}|\n"

    return strategy
```

Jako zadanie obliczeniowe przyjąłem opracowanie strategii dla 12 miesięcy, określając parametry zapotrzebowania oraz „mojej" fabryki w następujący sposób:

```
g = np.array([2, 8, 12, 15, 17, 20, 36, 40])
h = np.array([1, 2, 4, 8, 16, 32])
q = np.array([random.randint(0, 10) for i in range(12)])
y_min = 3
y_max = 7
y_begining = 5
y_end = 3
```

Należy jednak pamiętać, że dla wartości drugiego argumentu funkcji *random.randint()* większych od możliwości produkcyjnych fabryki (czyli długości g – 1) program może czasami wyrzucać błąd, dzieje się tak, ponieważ metoda zakłada wypełnianie zapotrzebowania q w pełni w każdym miesiącu, natomiast przy pewnym pechowym losowaniu może wystąpić sytuacja, w której liczba produktów w magazynie w połączeniu z możliwościami produkcyjnymi nie wystarcza na pokrycie zamówień.

Zadanie 2

Zadanie obliczeniowe wykonałem dla kilku zestawów danych. Dwa pierwsze zestawy miały funkcję głównie kontrolną, natomiast 3. zestaw jest zestawem głównym o danych przedstawionych w poprzednim punkcie. Rozwiązania podawane są w formie: najpierw macierz decyzji optymalnych, następnie macierz funkcji dla każdego etapu, na końcu strategia optymalna. Macierze indeksowane są jedynie dla możliwych stanów magazynu tzn. indeks wiersza = 0 odpowiada y = y_min, z kolei kolumny to kolejne etapy indeksowane sposobem jak na wykładzie tzn. kolumna 0 odpowiada f($y_{\text{stan końcowy} - 1}$), analogicznie dla x_matrix

- Zadanie z wykładu – szczególny przypadek omawianego zadania. Dane zadania:

```
g = np.array([0, 15, 18, 19, 20, 24])
h = np.array([i*2 for i in range(6)])
q = np.array([3,  3,  3,  3,  3,  3])
y_min = 0
y_max = 4
y_begining = 0
y_end = 0
```

Rozwiązanie:

```
X:
[[ 3.   3.   4.   3.   3.   4.]
 [ 2.   5.   5.   5.   5.   5.]
 [ 1.   4.   4.   4.   4.   4.]
 [ 0.   0.   0.   0.   0.   0.]
 [nan  0.   0.   0.   0.   0.]]

F:
[[ 19.  38.  52.  71.  90. 104.]
 [ 18.  30.  49.  68.  82. 101.]
 [ 15.  26.  45.  64.  78.  97.]
 [  0.  19.  38.  52.  71.  90.]
 [ inf  20.  32.  51.  70.  84.]]

Total cost = 104.0
|y0 = 0, x0 = 4|
|y1 = 1, x1 = 5|
|y2 = 3, x2 = 0|
|y3 = 0, x3 = 4|
|y4 = 1, x4 = 5|
|y5 = 3, x5 = 0|
|y6 = 0|
```

- Zadanie uproszczone dla n = 4, z danymi z konspektu:

```
g = np.array([2, 8, 12, 15, 17, 20])
h = np.array([1, 2, 3, 4])
q = np.array([4, 2, 6, 5])
y_min = 2
y_max = 5
y_begining = 4
y_end = 3
```

Rozwiązanie:

```
X:
[[nan nan  4.  4.]
 [ 5.  0.  3.  3.]
 [ 4.  5.  2.  2.]
 [ 3.  4.  1.  1.]]

F:
[[inf inf 62. 80.]
 [20. inf 60. 78.]
 [17. 42. 57. 75.]
 [15. 39. 53. 71.]]

Total cost = 75.0
|y0 = 4, x0 = 2|
|y1 = 2, x1 = 4|
|y2 = 4, x2 = 5|
|y3 = 3, x3 = 5|
|y4 = 3|
```

- Zadnie główne dla n = 12, z danymi (wylosowane q jest pokazane wraz z rozwiązaniem):

```
g = np.array([2, 8, 12, 15, 17, 20, 36, 40])
h = np.array([1, 2, 4, 8, 16, 32])
q = np.array([random.randint(0, 10) for i in range(12)])
y_min = 3
y_max = 7
y_begining = 5
y_end = 3
```

Rozwiązanie:

```
q:
[6 8 8 7 0 7 6 8 0 7 8 0]
X:
[[ 0. nan  0.  3. nan  7.  7.  2.  7. nan nan  0.]
 [nan  7.  7.  2.  7.  6.  6.  1.  6.  7.  0.  7.]
 [nan  6.  6.  1.  6.  5.  5.  0.  5.  6.  7.  6.]
 [nan  5.  5.  0.  5.  4.  4.  0.  4.  5.  6.  5.]
 [nan  4.  4.  0.  4.  5.  5.  0.  5.  4.  5.  4.]]

F:
[[  2.  inf  inf  88.  inf 171. 212. 208. 249.  inf  inf  inf]
 [ inf  43.  85.  85. 129. 167. 208. 204. 245. 290.  inf 376.]
 [ inf  39.  81.  81. 125. 151. 192. 198. 229. 286. 332. 372.]
 [ inf  23.  65.  75. 109. 148. 189. 199. 226. 270. 328. 356.]
 [ inf  20.  62.  80. 106. 137. 175. 193. 222. 267. 312. 353.]]

Total cost = 372.0
|y0 = 5, x0 = 6|
|y1 = 5, x1 = 7|
|y2 = 4, x2 = 7|
|y3 = 3, x3 = 7|
|y4 = 3, x4 = 2|
|y5 = 5, x5 = 5|
|y6 = 3, x6 = 7|
|y7 = 4, x7 = 7|
|y8 = 3, x8 = 3|
|y9 = 6, x9 = 5|
|y10 = 4, x10 = 7|
|y11 = 3, x11 = 0|
|y12 = 3|
```

Zadanie 3

- **Jakie modyfikacje zagadnienia można dodać, aby rozszerzyć i bardziej dostosować model problemu do rzeczywistych uwarunkowań produkcyjnych?**

  1. Uwzględnienie kosztów transportu, możliwym podejściem jest dodanie dodatkowego wektora kosztów transportu danej liczby zamówionych produktów – odwoływanie się do wektora za pomocą wartości q dla danego etapu.
  2. Rozszerzenie wektora kosztów do macierzy – dodanie możliwość zmiany kosztów produkcji w różnych miesiącach.
  3. Uwzględnienie możliwości niewypełnienia w całości zapotrzebowania q. W tym wypadku istnieje kilka możliwości:
     a. Opcja, w której po prostu ignorujemy nadmiar.
     b. Opcja, w której niewypełnienia zapotrzebowania powoduje jednorazową karę.
     c. Opcja, w której zobowiązanie przechodzi na następne miesiące zwiększając karę z każdym dodatkowym opóźnieniem

- **Jaka jest złożoność obliczeniowa algorytmu?**

  Przyjmując oznaczenia:
  y – liczba możliwych stanów
  x_max – możliwości produkcyjne fabryki
  le – liczba etapów

  Algorytm działa w dwóch głównych pętlach iterujących po etapach oraz po stanach, następnie dla danego etapu i stanu wybiera decyzję – w pesymistycznej wersji sprawdzając wszystkie możliwe dla danej fabryki decyzje czyli od 0 do możliwości produkcyjnych fabryki. Zatem w pesymistycznym przypadku złożoność wynosi: $O(y \cdot x\_\max \cdot le)$