

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Методы оптимизации и управления»

ОТЧЁТ
к лабораторной работе
на тему:
«МАТРИЧНАЯ ТРАНСПОРТНАЯ ЗАДАЧА»

Выполнил студент группы 053505
Слуцкий Никита Сергеевич

Проверил ассистент
каф.информатики
Туровец Николай Олегович

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Выполнение работы	5
2 Тестирование программного продукта.....	6
Заключение	7
ПРИЛОЖЕНИЕ А Листинг кода.....	8

ВВЕДЕНИЕ

Целью данной лабораторно работы ставится построить решение матричной транспортной задачи, используя метод потенциалов, для нахождения оптимального плана перевозки продукции из m пунктов производства в n пунктов назначения. Решение должно быть построено для работы на ЭВМ с использованием языка программирования Python.

1 ВЫПОЛНЕНИЕ РАБОТЫ

Подробное теоретическое описание и обоснование разработки программного продукта для решения матричной транспортной задачи методом потенциалов опускается, однако ниже будут введены описание проблемы и начало хода решения.

Даны m пунктов производства какой-либо продукции. Также имеется n пунктов сбыта (спроса). Перевозка из i -го пункта производства в j -й пункт сбыта стоит $c[i][j]$. Матрица C также является входным параметром для задачи. Таким образом определены 3 входных параметра для матричной транспортной задачи. Требуется построить план перевозки, чтобы:

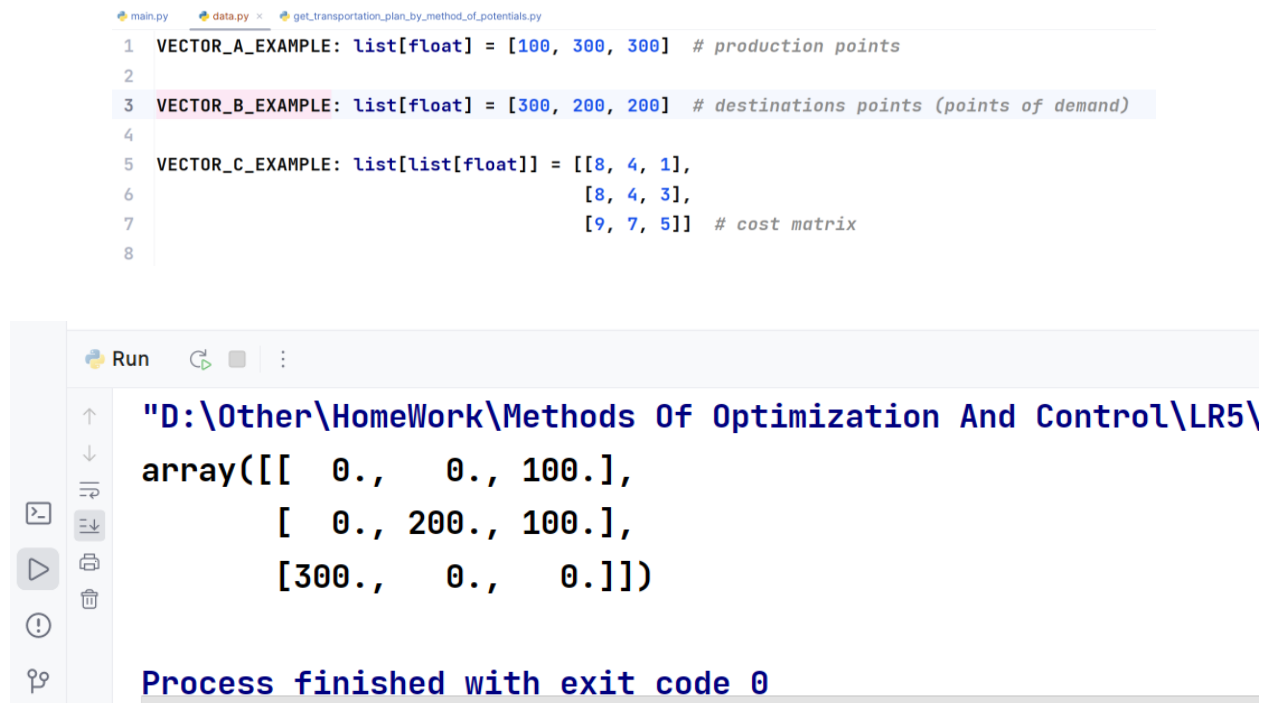
- все продукты из пунктов производства были вывезены;
- все заявки в пунктах сбыта были выполнены;
- общая стоимость перевозок была минимальна;

Для решения данной задачи используется метод потенциалов. Он перебирает базисные планы перевозок по определённым правилам и ищет план с минимальной суммарной стоимостью.

Метод состоит из двух фаз. После “обработки” начального базисного плана итерационно строятся следующие и на каком-то из этапов метод прекращает свою работу, если был определён факт, что план оптимален.

2 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

Для заданного в условии лабораторной работы примера программный продукт после отработки выдаёт корректный ответ.



The image shows a Python IDE with a code editor and a console. The code editor contains the following Python code:

```
1 VECTOR_A_EXAMPLE: list[float] = [100, 300, 300] # production points
2
3 VECTOR_B_EXAMPLE: list[float] = [300, 200, 200] # destinations points (points of demand)
4
5 VECTOR_C_EXAMPLE: list[list[float]] = [[8, 4, 1],
6                                         [8, 4, 3],
7                                         [9, 7, 5]] # cost matrix
8
```

The console shows the output of the program:

```
"D:\Other\HomeWork\Methods Of Optimization And Control\LR5\
array([[ 0.,   0., 100.],
       [ 0., 200., 100.],
       [300.,   0.,   0.]])

Process finished with exit code 0
```

Рисунок 1. Результат вывода программного продукта

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы был реализован алгоритм метода потенциалов для решения матричной транспортной задачи в соответствии с прилагаемым к работе условием и алгоритмом. Программное средство создано на языке программирования Python с использованием библиотеки для математических вычислений NumPy. Версия интерпретатора Python – 3.11.2.

Цели, поставленные во введении настоящего отчёта, можно считать достигнутыми. Лабораторная работа выполнена и готова к сдаче.

ПРИЛОЖЕНИЕ А

Листинг кода

```
import numpy as np
import copy

def check_condition_of_balance(a: list[float], b: list[float]) -> bool:
    return sum(a) == sum(b)

def is_plan_optimal(vector_basis_index: list[tuple[int, int]], vector_res: list[float], matrix_c:
list[list[float]], m: int, n: int) -> tuple[bool, tuple[int, int]]:
    for i in range(m):
        for j in range(n):
            if not (i, j) in vector_basis_index:
                if matrix_c[i][j] < (vector_res[i] + vector_res[j + m]):
                    return False, (i, j)

    return True, (0, 0)

def fill_matrix_x(vector_a: list[float], vector_b: list[float], matrix_x: np.array):
    i: int = 0
    j: int = 0

    vector_basis_index: list[tuple[int, int]] = list()

    while i != len(vector_a) and j != len(vector_b):
        if max(vector_a[i], vector_b[j]) == vector_a[i] and i > j:
            vector_a[i] -= vector_b[j]
            matrix_x[i][j] = vector_b[j]
            vector_basis_index.append((i, j))
            j += 1
        elif max(vector_a[i], vector_b[j]) == vector_b[j] and j <= i:
            vector_b[j] -= vector_a[i]
            matrix_x[i][j] = vector_a[i]
            vector_basis_index.append((i, j))
            i += 1

    return vector_basis_index

def get_transportation_plan_by_method_of_potentials(vector_a: list[float], vector_b: list[float], matrix_c:
list[list[float]]) -> np.array:
    m: int = len(vector_a)
    n: int = len(vector_b)

    if not check_condition_of_balance(vector_a, vector_b):
        raise Exception("Balance condition isn't met")

    matrix_x: np.array = np.zeros((m, n))
    vector_basis_index = fill_matrix_x(vector_a, vector_b, matrix_x)

    while True:
        matrix_u_v: list[list] = list()
        vector_u_v_res: list[float] = list()

        for i, j in vector_basis_index:
            arr: list[float] = [0] * (m + n)
            arr[i] = 1
            arr[j + m] = 1
            matrix_u_v.append(arr)
            vector_u_v_res.append(matrix_c[i][j])

        index: int = matrix_u_v[0].index(1)
        for i in range(len(matrix_u_v)):
            _ = matrix_u_v[i].pop(index)

        vector_res: list = list(np.linalg.solve(np.array(matrix_u_v), np.array(vector_u_v_res)).tolist())
        vector_res.insert(index, 0.0)
        # print(vector_res)

        check, index = is_plan_optimal(vector_basis_index, vector_res, matrix_c, m, n)
        # print("CHANGE ", index)
```

```

if check:
    return matrix_x

i_prev: int = vector_basis_index[0][0]
j_prev: int = vector_basis_index[0][1]

position: int = 0

for i, j in vector_basis_index[1:]:
    position += 1

    if i_prev == i:
        i_prev = i
        j_prev = j
        continue
    if j_prev == j:
        i_prev = i
        j_prev = j
        continue

    vector_basis_index.insert(position, index)
    break

if position == len(vector_basis_index) - 1:
    vector_basis_index.append(index)

# print("vector_basis ", vector_basis_index)
vector_basis_index_copy = copy.deepcopy(vector_basis_index)

for i in range(len(matrix_x)):
    sum_index: int = 0
    copy_i_index: int = 0
    copy_j_index: int = 0

    for i_index, j_index in vector_basis_index_copy:
        if i_index == i:
            sum_index += 1
            copy_i_index = i_index
            copy_j_index = j_index

    if sum_index == 1:
        vector_basis_index_copy.remove((copy_i_index, copy_j_index))

for j in range(len(matrix_x[0])):
    sum_index: int = 0
    copy_i_index: int = 0
    copy_j_index: int = 0

    for i_index, j_index in vector_basis_index_copy:
        if j_index == j:
            sum_index += 1
            copy_i_index = i_index
            copy_j_index = j_index

    if sum_index == 1:
        vector_basis_index_copy.remove((copy_i_index, copy_j_index))
# print("BASIS: ", vector_basis_index)
# print("BASIS COPY: ", vector_basis_index_copy)

vector_calc: list[float] = [0] * len(vector_basis_index_copy)
plus: bool = True
begin_index: int = vector_basis_index_copy.index(index)
# print("INDEX:", begin_index)
begin_index_copy: int = begin_index - 1

while begin_index < len(vector_basis_index_copy):
    i = vector_basis_index_copy[begin_index][0]
    j = vector_basis_index_copy[begin_index][1]

    if vector_basis_index_copy.__contains__((i, j)):
        vector_calc[begin_index] = (plus, matrix_x[i][j])
        plus = not plus
        begin_index += 1

plus = False

```



```

while begin_index_copy >= 0:
    i: int = vector_basis_index_copy[begin_index_copy][0]
    j: int = vector_basis_index_copy[begin_index_copy][1]

    if vector_basis_index_copy.__contains__((i, j)):
        vector_calc[begin_index_copy] = (plus, matrix_x[i][j])
        plus = not plus

    begin_index_copy -= 1

# print(vector_calc)
min_value = min(j for i, j in vector_calc if not i)
# print(min_value)
index_old = []
# print(matrix_x)
counter: int = 0
for i, j in vector_basis_index_copy:
    if vector_basis_index_copy.__contains__((i, j)):
        if vector_calc[counter][0]:
            matrix_x[i][j] += min_value
        else:
            matrix_x[i][j] -= min_value
        if matrix_x[i][j] == 0:
            index_old.append((i, j))

    counter += 1
index_old = sorted(index_old)
vector_basis_index.remove(index_old[0])

```