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](#_Toc133327542)

[1 Выполнение работы 5](#_Toc133327543)

[2 Тестирование программного продукта 6](#_Toc133327544)

[Заключение 7](#_Toc133327545)

[ПРИЛОЖЕНИЕ А Листинг кода 8](#_Toc133327546)

# 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 пуктов 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:

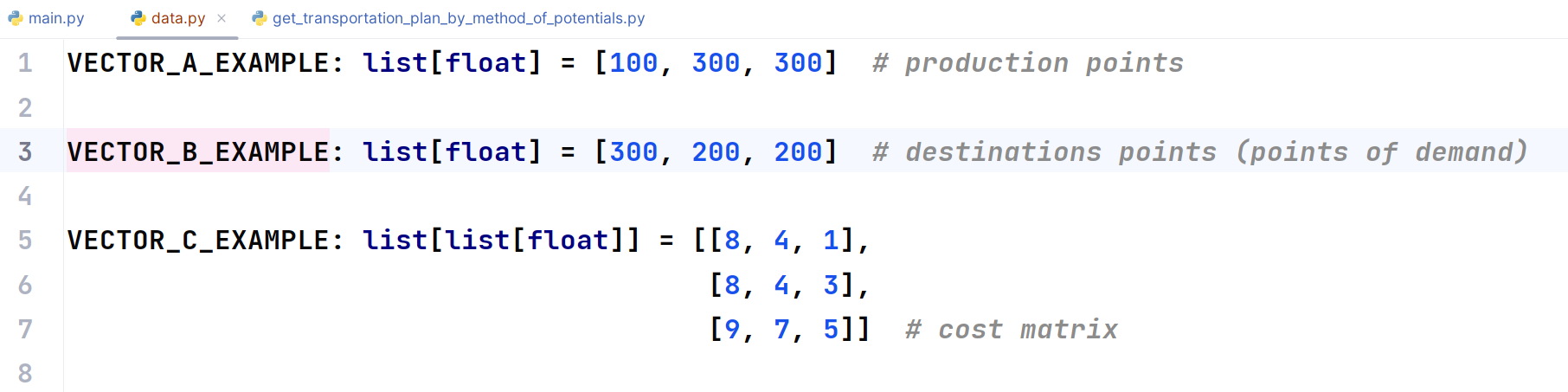
* все продукты из пунктов производства были вывезены;
* все заявки в пунктах сбыта были выполнены;
* общая стоимость перевозок была минимальна;

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.



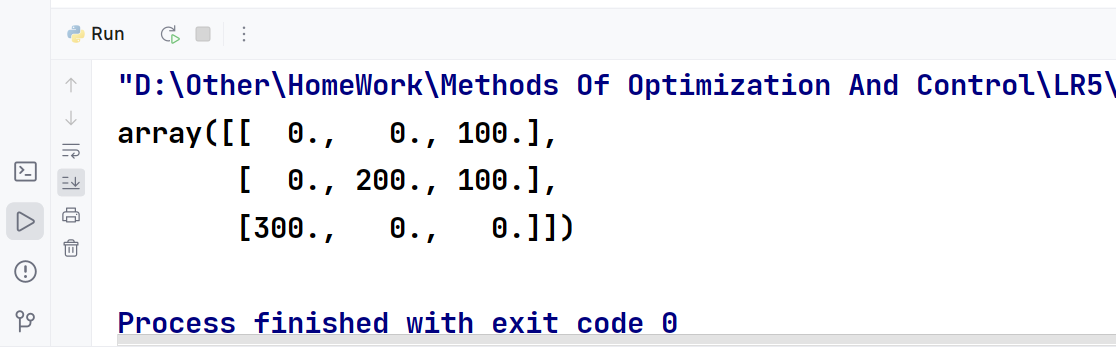


Рисунок 1. Результат вывода программного продукта

# 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])**