|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| Федеральное государственное автономное образовательное учреждение высшего образования | |
| **«Дальневосточный федеральный университет»** (ДВФУ) | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | |
| **Департамент математического и компьютерного моделирования** | |
| **ЛАБОРАТОРНАЯ РАБОТА №3** | |
| По основной образовательной программе подготовки бакалавров  направлению 02.03.01 Математика и компьютерные науки  профиль «Сквозные цифровые технологии» | |
|  | Студент группы  Б9120-02.03.01сцт - Пограничный Кирилл  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |
|  | Преподаватель: Яковлев Анатолий Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| г. Владивосток  2023 | |
|  | |

**Постановка задачи**

Имеем задачу линейной оптимизации. А то есть, задачу оптимизации производства. В данной поставленной задаче, будем решать прямую и двойственную задачу:

**Прямая задача**

Постановка

Сформулируем задачу:

Где: A = ∈

с = ∈

b = ∈

Сформулированная задача, является задачей на максимум дохода, от производства некоторой продукции. Объем производства продукции представлен вектором **x** с неотрицательными элементами. Вектор **с** – это удельный доход от единицы продукции, так же с отрицательными элементами. По условию m = 8, n = 6

Задачу будем решать симплекс-методом, для этого нужно привести задачу к канонической форме, т.е. ограничения типа равенства. Что бы это сделать, введём переменную **z**.

Получаем следующую систему:

Переобозначим переменные:

Получим задачу в канонической форме:

**Опорное решение**

Для прямой задачи мы автоматически имеем опорное решение, или в нашем случае опорное решение, так же будет являться базисным решением. Базисным решением системы называется частное решение, в котором неосновные переменные имеют нулевые значения.

**Двойственная задача**

Постановка

Формулировка задачи:

=

с = ∈

b = ∈

Решением задачи будет вектор **y**, который означает теневую цену ресурсов. Все остальные вектора и матрица удовлетворяют условию на неотрицательность, как и в прямой задаче.

Задачу также будем решать симплекс-методом, для этого нужно привести задачу к канонической форме. Что бы это сделать, введём переменную **z**.

Получаем следующую систему:

Переобозначим переменные:

Получим задачу в канонической форме:

Опорное решение

Это задача не имеет начальной угловой точки (опорного решения).

Для её нахождения будем решать вспомогательную задачу.

Сформулируем вспомогательную задачу, введем переменную **u**. Такая задача будет иметь вид:

Эту задачу решаем симплекс методом и считаем, что базисное решение для этой задачи, является опорным. Базисные решения, координаты которых удовлетворяют условию неотрицательности, являются опорными.

**Симплекс – метод**

**Симплекс-метод –** это метод последовательного перехода от одного базисного решения системы ограничений задачи линейного программирования к другому базисному решению до тех пор, пока функция цели не примет оптимального значения

**Приложение**

import numpy as np  
from tabulate import tabulate  
  
  
class Solution:  
 \_\_m = 8  
 \_\_n = 6  
 \_\_A = np.array([])  
 \_\_b = np.array([])  
 \_\_c = np.array([])  
 \_\_x = np.array([])  
 \_\_support\_matrix = 0  
  
 def \_\_init\_\_(self):  
 np.random.seed(16)  
 self.\_\_c = np.array(np.random.randint(100, size=self.\_\_n))  
 self.\_\_b = np.array(np.random.randint(100, size=self.\_\_m))  
 self.\_\_A = np.array(np.random.randint(100, size=(self.\_\_m, self.\_\_n)))  
  
 print("c:\n", self.\_\_c)  
 print("b:\n", self.\_\_b)  
 print("A:\n", self.\_\_A, "\n")  
  
 self.\_\_support\_matrix = self.\_\_create\_support\_mat()  
  
 def \_\_fun\_for\_print(self, mat, index\_l, index\_c, t):  
 print(f"Шаг алгоритма #{t}")  
 print(tabulate(mat, tablefmt="latex", floatfmt=".2f"))  
 print(f"Индекс разрешающей строки = {index\_l}")  
 print(f"Индекс разрещающего столбца = {index\_c}")  
 print(f"Разрешающий элемент = {mat[index\_l][index\_c]}")  
 answer = []  
  
 for i in range(mat[0].size - 1):  
 if sum(mat[0:, i]) == 1 and mat[0, i] == 0:  
 index\_line = np.where(mat[1:, i] == 1)[0][0] + 1  
 answer.append(mat[index\_line][mat[0].size - 1])  
 else:  
 answer.append(0)  
 print("Промежуточное решение:\n", np.array(answer))  
  
 def \_\_create\_support\_mat(self):  
 main\_mat = np.zeros((7, 21))  
 main\_mat[1:, :8] = self.\_\_A.T  
 main\_mat[1:, 8:14] = -np.eye(6)  
 main\_mat[1:, 14:20] = np.eye(6)  
 main\_mat[1:, 20] = self.\_\_c  
 main\_mat[0, 14:20] = 1  
  
 temp = 0  
 for i in range(1, main\_mat.shape[0]):  
 temp += main\_mat[i]  
 main\_mat[0] -= temp  
  
 return main\_mat  
  
 def \_\_simplex\_method(self, main\_mat, m, n):  
 t = 0  
 index\_min\_line = 0  
 index\_min\_col = 0  
 while np.any(main\_mat[0, :n - 1] < 0):  
 t += 1  
 index\_min\_col = np.where(main\_mat == main\_mat[0, :(n - 1)].min())[1][0]  
 div\_last = []  
 for i in range(1, m):  
 div\_last.append(main\_mat[i][-1] / main\_mat[i][index\_min\_col])  
  
 index\_min\_line = np.where(div\_last == min(filter(lambda x: x > 0, div\_last)))[0][0] + 1  
  
 self.\_\_fun\_for\_print(main\_mat, index\_min\_line, index\_min\_col, t)  
 main\_mat[index\_min\_line] /= main\_mat[index\_min\_line][index\_min\_col]  
 cur\_col = main\_mat[:, index\_min\_col]  
  
 for i in range(len(cur\_col)):  
 if cur\_col[i] != 1:  
 main\_mat[i] -= main\_mat[index\_min\_line] \* cur\_col[i]  
 t += 1  
 print("END OF SIMPLEX!!!: \n")  
 self.\_\_fun\_for\_print(main\_mat, index\_min\_line, index\_min\_col, t)  
  
 return main\_mat  
  
 def \_\_create\_main(self, c, b):  
 main\_mat = np.zeros((9, 15))  
 main\_mat[0][:c.shape[0]] = -c  
 main\_mat[1:, :6] = self.\_\_A  
 main\_mat[1:, 6: 14] = np.eye(8)  
 main\_mat[1:, 14] = b  
 return main\_mat  
  
 def \_\_to\_ready\_double\_problem(self, mat, b):  
 mat[0, :8] = b  
 mat[0, 14] = 0  
 for i in range(15):  
 if sum(mat[1:, i]) == 1 and mat[0, i] != 0:  
 index\_line = np.where(mat[1:, i] == 1)[0][0] + 1  
 mat[0, :15] -= mat[index\_line] \* mat[0][i]  
 mat[0][i] = 0  
  
 return mat  
  
 def execute(self):  
 print("---Прямая---")  
 print(tabulate(self.\_\_simplex\_method(self.\_\_create\_main(self.\_\_c, self.\_\_b), m=9, n=15),  
 tablefmt="latex", floatfmt=".2f"))  
  
 support\_mat = self.\_\_simplex\_method(self.\_\_create\_support\_mat(), m=7, n=21)  
 print("Последний шаг подготовки к решению двойственной задачи:\n")  
 print(tabulate(support\_mat, tablefmt="latex", floatfmt=".2f"))  
  
 double\_mat = np.column\_stack([support\_mat[0:, :14], support\_mat[0:, 20]])  
 double\_mat = self.\_\_to\_ready\_double\_problem(double\_mat, self.\_\_b)  
  
 print("---DOUBLE MATRIX---\n")  
 print(tabulate(double\_mat, tablefmt="latex", floatfmt=".2f"))  
  
 print("---Двойственная---\n")  
 print(tabulate(self.\_\_simplex\_method(double\_mat, m=7, n=15), tablefmt="latex", floatfmt=".2f"))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 Solution().execute()