**МИНОБРНАУКИ РОССИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования

**«Национальный исследовательский университет «МЭИ»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛАБОРАТОРНАЯ РАБОТА №1

По курсу: «Методы решения задач оптимизации»

Тема: «Симплекс-метод»

|  |  |
| --- | --- |
| Выполнил: | Волков М.Л. |
| Вариант: | 1 |
| Группа: | Э-13м-23 |
| Проверил: | Нухулов С.М. |

Москва, 2024 г.

**Предварительный отчет**

**Цель:** получение практических навыков работы с методом решения задач линейного программирования с количеством неизвестных больше 3

**Задание:**

1. Решение задачу линейного программирования табличным симплекс-методом;
2. Написание алгоритма симплекс-метода на языке Python;
3. Решение задачи линейного программирования с помощью написанного алгоритма и сравнение результатов с алгоритмом симплекс-метода, реализованном в библиотеке SciPy;

**Теоретическая справка:**

***Признак бесконечности множества оптимальных планов (альтернативный оптимум):*** *если в индексной строке последней симплексной таблицы (содержащей оптимальный план) имеется хотя бы одна нулевая оценка, соответствующая свободной переменной, то задача линейного программирования имеет бесконечное множество оптимальных планов.*

***Признак неограниченности целевой функции:*** *если в разрешающем столбце нет ни одного положительного элемента, то целевая функция на множестве допустимых планов не ограничена.*

***Признак несовместности системы ограничений:*** *если в оптимальном плане М-задачи не все искусственные переменные равны нулю, то система ограничений исходной задачи несовместна.*

**Расчет:**

**Целевая функция:**

**Ограничения:**

Преобразование (ввод переменных):

Третье равенство не содержит переменных предпочтительного вида, поэтому необходимо использовать искусственный метод. Необходимо ввести дополнительные переменные.

Исходная симплекс-таблица:

| **БП** |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **-4** | **-3** | **2** | **0** | **0** | **M** | **M** |
|  | 0 | 5 | 1 | 3 | 0 | 1 | 0 | 0 | 0 | - |
|  | M | 7 | -1 | 3 | 0 | 0 | 0 | 1 | 0 | - |
|  | M | 5 | -4 | -4 | 3 | 0 | -1 | 0 | 1 | 5/3 |
|  | | 0 | 4 | 3 | -2 | 0 | 0 | 0 | 0 |  |
| 12M | -5M | -M | 3M | 0 | -M | 0 | 0 |

Симплекс-таблица после первой итерации:

| **БП** |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **-4** | **-3** | **2** | **0** | **0** | **M** | **M** |
|  | 0 | 5 | 1 | 3 | 0 | 1 | 0 | 0 | 0 | 5/3 |
|  | M | 7 | -1 | 3 | 0 | 0 | 0 | 1 | 0 | 7/3 |
|  | 2 | 5/3 | -4/3 | -4/3 | 1 | 0 | -1/3 | 0 | 1/3 | -5/4 |
|  | | 10/3 | 4/3 | 1/3 | 0 | 0 | -2/3 | 0 | 2/3 |  |
| 7M | -M | 3M | 0 | 0 | 0 | 0 | -M |

Симплекс-таблица после второй итерации:

| **БП** |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **-4** | **-3** | **2** | **0** | **0** | **M** | **M** |
|  | -3 | 5/3 | 1/3 | 1 | 0 | 1/3 | 0 | 0 | 0 | - |
|  | M | 2 | -2 | 0 | 0 | -1 | 0 | 1 | 0 | - |
|  | 2 | 35/9 | -8/9 | 0 | 1 | 4/9 | -1/3 | 0 | 1/3 | - |
|  | | 25/9 | 11/9 | 0 | 0 | -1/9 | -2/3 | 0 | 2/3 |  |
| 2M | -2M | 0 | 0 | -M | 0 | 0 | -M |

**Вывод:** по признаку несовместности системы ограничений, система ограничений исходной задачи несовместна, так как при оптимальном плане не все искусственные переменные равны 0.

**Отчет**

**Написание алгоритма симплекс-метода:**

Код:

import numpy as np  
  
n\_value = int(input('Введите количество коэффициентов при переменных в целевой функции: '))  
array\_function = [-4, -3, 2, 0, 0, 0, 0]  
array\_function.insert(0, 0)  
print('Коэффициенты при переменных в целевой функции: ', array\_function)  
  
*# Создание матрицы ограничений*n\_constraints = int(input('Введите количество ограничений: '))  
matrix\_constraints = [[1, 3, 0, 1, 0, 0, 0], [-1, 3, 0, 0, 0, 1, 0], [-4, -4, 3, 0, -1, 0, 1]]  
right\_constraints = [5, 7, 5]  
  
for i in range(n\_constraints):  
 matrix\_constraints[i].insert(0, right\_constraints[i])  
print('Исходная симплекс таблица: \n', np.array(matrix\_constraints))  
  
*# Инициализация пустого списка базисов*basis = [0, 0, 0]  
  
*# Инициализация списка дельт*delta = [0] \* (n\_value + 1)  
  
  
def finding\_delta\_list(delta\_arr, matrix, n\_const):  
 for i in range(n\_const + 1):  
 value = 0  
 for j in range(len(basis)):  
 value = value + basis[j] \* matrix[j][i]  
 delta\_arr[i] = value - array\_function[i]  
 print('delta: ', delta\_arr, '\n')  
 return delta\_arr  
  
  
def finding\_resolving\_post(delta\_list):  
 *# Определение разрешающего столбца* value\_j = 0  
 index\_j = 0  
 flag\_j = False  
 for i in range(1, len(delta\_list)):  
 if delta\_list[i] < 0:  
 if abs(delta\_list[i] > abs(value\_j)) or flag\_j == False:  
 value\_j = delta\_list[i]  
 index\_j = i  
 flag\_j = True  
 print('Индекс(номер) разрешающего столбца: ', index\_j, '\n')  
 return value\_j, index\_j, flag\_j  
  
  
def finding\_teta\_arr(n\_const, basis\_arr, right\_const, matrix, index\_j):  
 *# Инициализация столбца тета* teta\_table = [0] \* n\_const  
 for i in range(len(basis\_arr)):  
 if matrix[i][index\_j] == 0:  
 teta\_table[i] = -1  
 else:  
 teta\_table[i] = right\_const[i] / matrix[i][index\_j]  
 print('Тета столбец: ', teta\_table, '\n')  
 return teta\_table  
  
  
def finding\_resolving\_list(teta\_arr):  
 *# Определение разрешающей строки* value\_i = 0  
 index\_i = 0  
 flag\_i = False  
 for i in range(len(teta\_arr)):  
 if teta\_arr[i] > 0:  
 if teta\_arr[i] < value\_i or flag\_i == False:  
 flag\_i = True  
 value\_i = teta\_arr[i]  
 index\_i = i  
 print('Индекс разрешающей строки: ', index\_i, '\n')  
 return value\_i, index\_i, flag\_i  
  
  
def creating\_new\_table(n\_val, n\_const, matrix, index\_i, index\_j, count\_iter):  
 *# Формирование новой таблицы* new\_matrix = []  
 for i in range(n\_const):  
 new\_matrix.append([0] \* (n\_val + 1))  
  
 for i in range(n\_val + 1):  
 for j in range(n\_const):  
 if j != index\_i and i != index\_j:  
 new\_matrix[j][i] = (matrix[j][i] \* matrix[index\_i][index\_j] -  
 matrix[index\_i][i] \* matrix[j][index\_j]) / matrix[index\_i][index\_j]  
 elif j == index\_i and i != index\_j:  
 new\_matrix[j][i] = matrix[j][i] / matrix[index\_i][index\_j]  
 elif j != index\_i and i == index\_j:  
 new\_matrix[j][i] = 0  
 else:  
 new\_matrix[j][i] = 1  
 print('Симплекс таблица после ', count\_iter, ' итерации: \n', np.array(new\_matrix))  
 return new\_matrix  
  
  
def print\_optim(basis\_index\_list, matrix, delta\_list):  
 for i in range(len(basis\_index\_list)):  
 print('X', i + 1, ' = ', matrix[basis\_index\_list.index(i + 1)][0], sep="")  
 print('fun = ', delta\_list[0])  
  
  
flag\_whil = True  
count = 1  
  
*# Ввод списка, хранящего порядок определяемых элементов*basis\_index = [0] \* n\_constraints  
  
while flag\_whil:  
 delta = finding\_delta\_list(delta, matrix\_constraints, n\_value)  
  
 *# Нахождение индекса разрешающего столбца* value\_allow\_j, index\_allow\_j, flag\_post = finding\_resolving\_post(delta)  
 if flag\_post == False:  
 print('План оптимален \n')  
 print\_optim(basis\_index, matrix\_constraints, delta)  
 flag = False  
 break  
  
 *# Определение столбца тета* teta\_table = finding\_teta\_arr(n\_constraints, basis, right\_constraints, matrix\_constraints, index\_allow\_j)  
  
 *# Нахождение индекса разрешающей строки* value\_allow\_i, index\_allow\_i, flag\_list = finding\_resolving\_list(teta\_table)  
 if flag\_list == False:  
 print('Оптимального плана нет!')  
 flag = False  
 break  
  
 basis\_index[index\_allow\_i] = index\_allow\_j  
  
 *# Переопределение базисного столбца* basis[index\_allow\_i] = array\_function[index\_allow\_j]  
  
 *# Определение новой симплекс таблицы* matrix\_constraints = creating\_new\_table(n\_value, n\_constraints, matrix\_constraints, index\_allow\_i, index\_allow\_j,  
 count)  
 count = count + 1

**Проверка работы программы:**

Для проверки работы программы был использован разобранный пример из лекционного курса.

Код:

import numpy as np  
  
n\_value = int(input('Введите количество коэффициентов при переменных в целевой функции: '))  
array\_function = [3, 0, 2, 0, 0, 0]  
array\_function.insert(0, 0)  
  
print('Коэффициенты при переменных в целевой функции: ', array\_function)  
  
*# Создание матрицы ограничений*n\_constraints = int(input('Введите количество ограничений: '))  
matrix\_constraints = [[-2, -1, 5, 1, 0, 0], [1, 0, -2, 0, 1, 0], [0, 2, -1, 0, 0, 1]]  
right\_constraints = [6, 2, 5]  
  
for i in range(n\_constraints):  
 matrix\_constraints[i].insert(0, right\_constraints[i])  
print('Исходная симплекс таблица: \n', np.array(matrix\_constraints))  
  
*# Инициализация пустого списка базисов*basis = [0, 0, 0]  
  
*# Инициализация списка дельт*delta = [0] \* (n\_value + 1)  
  
  
def finding\_delta\_list(delta\_arr, matrix, n\_const):  
 for i in range(n\_const + 1):  
 value = 0  
 for j in range(len(basis)):  
 value = value + basis[j] \* matrix[j][i]  
 delta\_arr[i] = value - array\_function[i]  
 print('delta: ', delta\_arr, '\n')  
 return delta\_arr  
  
  
def finding\_resolving\_post(delta\_list):  
 value\_j, index\_j = 0, 0  
 flag\_j = False  
 for i in range(1, len(delta\_list)):  
 if delta\_list[i] < 0:  
 if abs(delta\_list[i] > abs(value\_j)) or flag\_j == False:  
 value\_j = delta\_list[i]  
 index\_j = i  
 flag\_j = True  
 print('Индекс(номер) разрешающего столбца: ', index\_j, '\n')  
 return value\_j, index\_j, flag\_j  
  
  
def finding\_teta\_arr(n\_const, basis\_arr, right\_const, matrix, index\_j):  
 teta\_table = [0] \* n\_const  
 for i in range(len(basis\_arr)):  
 if matrix[i][index\_j] == 0:  
 teta\_table[i] = -1  
 else:  
 teta\_table[i] = right\_const[i] / matrix[i][index\_j]  
 print('Тета столбец: ', teta\_table, '\n')  
 return teta\_table  
  
  
def finding\_resolving\_list(teta\_arr):  
 value\_i, index\_i = 0, 0  
 flag\_i = False  
 for i in range(len(teta\_arr)):  
 if teta\_arr[i] > 0:  
 if teta\_arr[i] < value\_i or flag\_i == False:  
 flag\_i = True  
 value\_i = teta\_arr[i]  
 index\_i = i  
 print('Индекс разрешающей строки: ', index\_i, '\n')  
 return value\_i, index\_i, flag\_i  
  
  
def creating\_new\_table(n\_val, n\_const, matrix, index\_i, index\_j, count\_iter):  
 new\_matrix = []  
 for i in range(n\_const):  
 new\_matrix.append([0] \* (n\_val + 1))  
  
 for i in range(n\_val + 1):  
 for j in range(n\_const):  
 if j != index\_i and i != index\_j:  
 new\_matrix[j][i] = (matrix[j][i] \* matrix[index\_i][index\_j] -  
 matrix[index\_i][i] \* matrix[j][index\_j]) / matrix[index\_i][index\_j]  
 elif j == index\_i and i != index\_j:  
 new\_matrix[j][i] = matrix[j][i] / matrix[index\_i][index\_j]  
 elif j != index\_i and i == index\_j:  
 new\_matrix[j][i] = 0  
 else:  
 new\_matrix[j][i] = 1  
 print('Симплекс таблица после ', count\_iter, ' итерации: \n', np.array(new\_matrix))  
 return new\_matrix  
  
  
def print\_optim(basis\_index\_list, matrix, delta\_list):  
 for i in range(len(basis\_index\_list)):  
 print('X', i + 1, ' = ', matrix[basis\_index\_list.index(i + 1)][0], sep="")  
 print('fun = ', delta\_list[0])  
  
  
flag\_whil = True  
count = 1  
  
*# Ввод списка, хранящего порядок определяемых элементов*basis\_index = [0] \* n\_constraints  
  
while flag\_whil:  
 delta = finding\_delta\_list(delta, matrix\_constraints, n\_value)  
  
 *# Нахождение индекса разрешающего столбца* value\_allow\_j, index\_allow\_j, flag\_post = finding\_resolving\_post(delta)  
 if flag\_post == False:  
 print('План оптимален \n')  
 print\_optim(basis\_index, matrix\_constraints, delta)  
 flag = False  
 break  
  
 *# Определение столбца тета* teta\_table = finding\_teta\_arr(n\_constraints, basis, right\_constraints, matrix\_constraints, index\_allow\_j)  
  
 *# Нахождение индекса разрешающей строки* value\_allow\_i, index\_allow\_i, flag\_list = finding\_resolving\_list(teta\_table)  
 if flag\_list == False:  
 print('Оптимального плана нет!')  
 flag = False  
 break  
  
 basis\_index[index\_allow\_i] = index\_allow\_j  
  
 *# Переопределение базисного столбца* basis[index\_allow\_i] = array\_function[index\_allow\_j]  
  
 *# Определение новой симплекс таблицы* matrix\_constraints = creating\_new\_table(n\_value, n\_constraints, matrix\_constraints, index\_allow\_i, index\_allow\_j,  
 count)  
 count = count + 1

Результат:

Введите количество коэффициентов при переменных в целевой функции: 6

Коэффициенты при переменных в целевой функции: [0, 3, 0, 2, 0, 0, 0]

Введите количество ограничений: 3

Исходная симплекс таблица:

[[ 6 -2 -1 5 1 0 0]

[ 2 1 0 -2 0 1 0]

[ 5 0 2 -1 0 0 1]]

delta: [0, -3, 0, -2, 0, 0, 0]

Индекс(номер) разрешающего столбца: 1

Тета столбец: [-3.0, 2.0, -1]

Индекс разрешающей строки: 1

Симплекс таблица после 1 итерации:

[[10. 0. -1. 1. 1. 2. 0.]

[ 2. 1. 0. -2. 0. 1. 0.]

[ 5. 0. 2. -1. 0. 0. 1.]]

delta: [6.0, 0, 0.0, -8.0, 0.0, 3.0, 0.0]

Индекс(номер) разрешающего столбца: 3

Тета столбец: [6.0, -1.0, -5.0]

Индекс разрешающей строки: 0

Симплекс таблица после 2 итерации:

[[10. 0. -1. 1. 1. 2. 0.]

[22. 1. -2. 0. 2. 5. 0.]

[15. 0. 1. 0. 1. 2. 1.]]

delta: [86.0, 0.0, -8.0, 0, 8.0, 19.0, 0.0]

Индекс(номер) разрешающего столбца: 2

Тета столбец: [-6.0, -1.0, 5.0]

Индекс разрешающей строки: 2

Симплекс таблица после 3 итерации:

[[25. 0. 0. 1. 2. 4. 1.]

[52. 1. 0. 0. 4. 9. 2.]

[15. 0. 1. 0. 1. 2. 1.]]

delta: [206.0, 0.0, 0, 0.0, 16.0, 35.0, 8.0]

Индекс(номер) разрешающего столбца: 0

План оптимален

X1 = 52.0

X2 = 15.0

X3 = 25.0

fun = 206.0

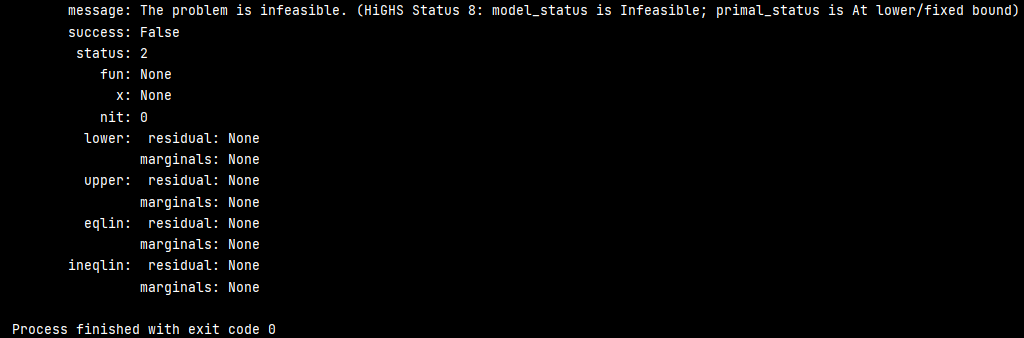
Process finished with exit code 0

**Использование библиотеки SciPy:**

Код:

import scipy.optimize as opt  
  
funcF = [-4, -3, 2]  
Arr\_left\_constraints = [[1, 3, 0], [4, 4, -3]]  
Arr\_right\_constraints = [5, -5]  
  
left\_const = [[-1, 3, 0]]  
right\_const = [7]  
  
bnd = [(0, float("inf")), (0, float("inf")), (0, float("inf"))]  
res = opt.linprog(c=funcF, A\_ub=Arr\_left\_constraints, b\_ub=Arr\_right\_constraints, A\_eq=left\_const, b\_eq=right\_const, bounds=bnd)  
  
print(res)

Результат:



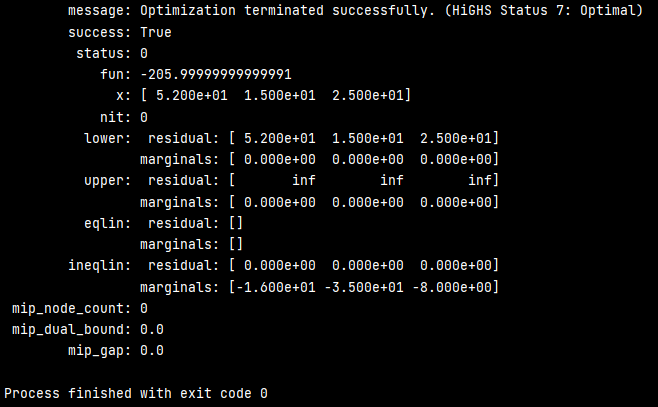
**Проверка работы программы:**

Для проверки работы программы был использован разобранный пример из лекционного курса.

Код:

import scipy.optimize as opt  
  
funcF = [-**3, 0,** -**2**]  
Arr\_left\_constraints = [[-**2,** -**1, 5**]**,** [**1, 0,** -**2**]**,** [**0, 2,** -**1**]]  
Arr\_right\_constraints = [**6, 2, 5**]  
  
bnd = [(**0,** float("inf"))**,** (**0,** float("inf"))**,** (**0,** float("inf"))]  
res = opt.linprog(c=funcF**,** A\_ub=Arr\_left\_constraints**,** b\_ub=Arr\_right\_constraints**,** bounds=bnd)  
  
print(res)

Результат:



**Вывод:** в ходе лабораторной работы, были получены практические навыки использования симплекс-метода; был реализован математический алгоритм симплекс-метода(классического) на языке Python; полученные результаты были сверены с ручным расчетом и с программой, использующей библиотеку SciPy; результаты программной реализации совпали с ручным расчетом.