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 M-задачи не все искусственные переменные равны нулю, то система ограничений исходной задачи несовместна.*

Расчет:

Целевая функция:

$$\max(f) = -4 \cdot x_1 - 3 \cdot x_2 + 2 \cdot x_3$$

Ограничения:

$$\begin{cases} x_1 + 3 \cdot x_2 \leq 5 \\ -x_1 + 3 \cdot x_2 = 7 \\ -4 \cdot x_1 - 4 \cdot x_2 + 3 \cdot x_3 \geq 5 \end{cases}$$

Преобразование (ввод переменных):

$$\begin{cases} x_1 + 3 \cdot x_2 + x_4 = 5 \\ -x_1 + 3 \cdot x_2 = 7 \\ -4 \cdot x_1 - 4 \cdot x_2 + 3 \cdot x_3 - x_5 = 5 \end{cases}$$

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

$$\begin{cases} x_1 + 3 \cdot x_2 + x_4 = 5 \\ -x_1 + 3 \cdot x_2 + \omega_1 = 7 \\ -4 \cdot x_1 - 4 \cdot x_2 + 3 \cdot x_3 - x_5 + \omega_2 = 5 \end{cases}$$

Исходная симплекс-таблица:

БП	$C_б$	A_0	x_1	x_2	x_3	x_4	x_5	ω_1	ω_2	θ
			-4	**-3**	**2**	**0**	**0**	**M**	**M**	
x_4	0	5	1	3	0	1	0	0	0	-
ω_1	M	7	-1	3	0	0	0	1	0	-
ω_2	M	5	-4	-4	3	0	-1	0	1	5/3
$f_i - c_i$		0	4	3	-2	0	0	0	0	
		12M	-5M	-M	3M	0	-M	0	0	

Симплекс-таблица после первой итерации:

БП	$C_б$	A_0	x_1	x_2	x_3	x_4	x_5	ω_1	ω_2	θ
			-4	**-3**	**2**	**0**	**0**	**M**	**M**	
x_4	0	5	1	3	0	1	0	0	0	5/3
ω_1	M	7	-1	3	0	0	0	1	0	7/3
x_3	2	5/3	-4/3	-4/3	1	0	-1/3	0	1/3	-5/4
$f_i - c_i$		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	$C_б$	A_0	x_1	x_2	x_3	x_4	x_5	ω_1	ω_2	θ
			-4	**-3**	**2**	**0**	**0**	**M**	**M**	
x_2	-3	5/3	1/3	1	0	1/3	0	0	0	-
ω_1	M	2	-2	0	0	-1	0	1	0	-
x_3	2	35/9	-8/9	0	1	4/9	-1/3	0	1/3	-
$f_i - c_i$		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:

```
       message: The problem is infeasible. (HiGHS Status 8: model_status is Infeasible; primal_status is At lower/fixed bound)
       success: False
        status: 2
           fun: None
             x: None
           nit: 0
         lower:  residual: None
                marginals: None
         upper:  residual: None
                marginals: None
         eqlin:  residual: None
                marginals: None
       ineqlin:  residual: None
                marginals: None

Process finished with exit code 0
```

Проверка работы программы:

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

Код:

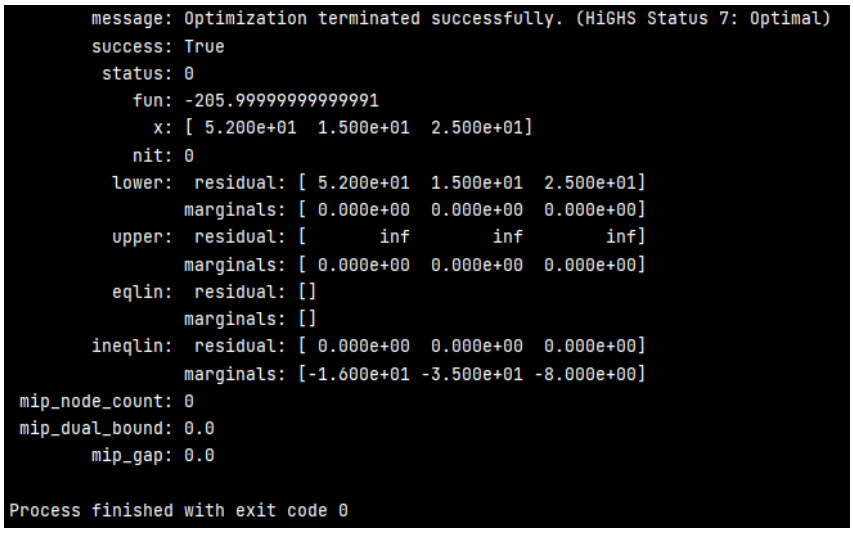
```
import scipy.optimize as opt

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.