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:
«НАЧАЛЬНАЯ ФАЗА СИМПЛЕКС-МЕТОДА»

Выполнил студент группы 053505
Слуцкий Никита Сергеевич

Проверил ассистент
каф.информатики
Туровец Николай Олегович

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пусть имеется задача линейного программирования в канонической форме:

$$c^T x \to max$$
$$Ax = b$$
$$x \geq 0$$

Требуется определить совместна ли задача и, в случае положительного ответа, найти какой-нибудь базисный допустимый план (x, B).

1 ВЫПОЛНЕНИЕ РАБОТЫ

Начальная фаза симплекс-метода

Вход: $c \in R\ n$, $A_0 \in R\ m \times n$ и $b \in R\ m$ — параметры задачи (1)

Выход: (x, B) — базисный допустимый план задачи (1) или сообщение о том, что задача (1) не имеет допустимых планов.

Шаг 1. Необходимо преобразовать задачу (1) таким образом, чтобы вектор правых частей b был неотрицательным. Для этого умножим на −1 все ограничения задачи, правая часть которых отрицательна. А именно, для каждого индекса $i \in \{1, 2, \ldots, m\}$ выполним следующую операцию: если $b_i < 0$, то умножим на −1 компоненту bi и i-ю строку матрицы A;

Шаг 2. Составим вспомогательную задачу линейного программирования

$$c^T x \rightarrow max$$
$$Ax = b$$
$$x \geq 0$$

(2)

где вектор коэффициентов при переменных в целевой функции имеет вид $c^T = (0, 0, \ldots, 0, -1, -1, \ldots, -1) \in R^{n+m}$, вектор переменных — $x = (x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})^T \in R^{n+m}$ (переменные $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ называются искусственными), матрица A получается из матрицы A_0 присоединением к ней справа единичной матрицы порядка m.

Шаг 3. Построим начальный базисный допустимый план (x, B) вспомогательный задачи

Шаг 4. Решим вспомогательную задачу основной фазой симплекс-метода и получим оптимальный план и соответствующее ему множество базисных индексов B.

Шаг 5. Проверим условия совместности: если $x_{n+1} = x_{n+2} = \ldots = x_{n+m} = 0$, то задача (1) совместна; в противном случае, задача (1) не совместна и метод завершает свою работу.

Шаг 6. Формируем допустимый план задачи (1). Для него необходимо подобрать множество базисных индексов. С этой целью скорректируем множество.

Шаг 7. Если $B \subseteq \{1, 2, \ldots, n\}$, то метод завершает свою работу и возвращает базисный допустимый план (x, B).

Шаг 8. Выберем в наборе B максимальный индекс искусственной переменной

Шаг 7. Для каждого индекса $j \in \{1, 2, \ldots, n\} \setminus B$ вычислим вектор

$$\ell(j) = \widetilde{A}_B^{-1} \widetilde{A}_j,$$

где $\widetilde{A}_j$ — это j-ый столбец матрицы $\widetilde{A}$.

Шаг 8. Если найдется индекс $j \in \{1, 2, \ldots, n\} \setminus B$ такой, что $(\ell(j))_k \neq 0$, то заменим в наборе B значение j_k, равное $n + i$, на j.

Шаг 9. Если для любого индекса $j \in \{1, 2, \ldots, n\} \setminus B$ выполняется $(\ell(j))_k = 0$, то i-е основное ограничение задачи (1) линейно выражается через остальные и его необходимо удалить. В этом случае удалим i-ую строку из матрицы A и i-ую компоненту из вектора b. Удалим из B индекс $j_k = n + i$. Кроме этого, удалим i-ую строку из матрицы $\widetilde{A}$. Переходим на Шаг 7.

2 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

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

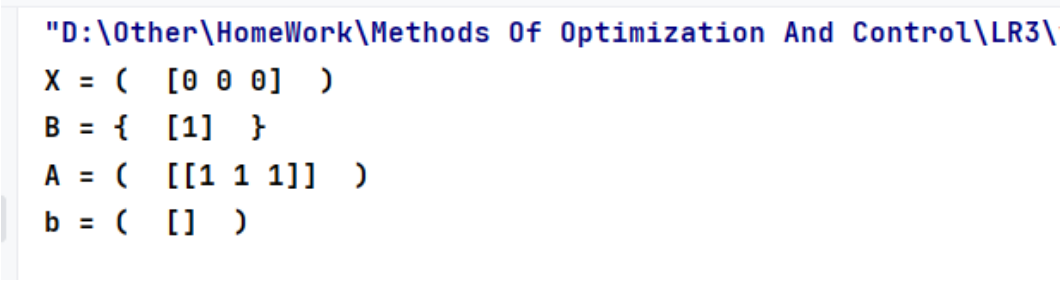

Рисунок 1. Результат вывода программного продукта

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы был реализован алгоритм начальной фазы симплекс-метода в соответствии с шаблоном из методического пособия. Программное средство создано на языке программирования Python с использованием библиотеки для математических вычислений NumPy.

Цели лабораторной работы можно считать достигнутыми. Работа выполнена.

ПРИЛОЖЕНИЕ А

Листинг кода

```
def start_phase_of_simplex_method(matrix_a: np.array, vector_b: np.array) -> (np.array, np.array,
np.array, np.array):
    a: np.array = copy.deepcopy(matrix_a)
    b: np.array = copy.deepcopy(vector_b)

    rows_count: int = matrix_a.shape[0]
    columns_count: int = matrix_a.shape[1]

    # 1) remove negative rows (where b[i] < 0)
    for row_index in range(rows_count):
        if b[row_index] < 0:
            for col_index in range(columns_count):
                a[row_index][col_index] *= -1
            b[row_index] *= -1

    # 2) get auxiliary task of linear programming c_auxiliary
    c_auxiliary: np.array = np.array([0] * columns_count + [-1] * rows_count)
    identity_matrix: np.array = np.identity(rows_count)

    a_auxiliary: np.array = np.append(a, identity_matrix, axis=1)  # glue matrices horizontally

    # 3)
    x_auxiliary_initial_plan: np.array = np.append(np.array([0] * columns_count), b)
    b_auxiliary_initial_plan: np.array = np.array([columns_count + counter + 1 for counter in
range(rows_count)])

    # 4) solve auxiliary task by main(basic) phase of simplex method
    auxiliary_response = basic_phase_of_simplex_method(a_auxiliary, c_auxiliary,
x_auxiliary_initial_plan, b_auxiliary_initial_plan)
    x_basis_plan, b_basis_indexes = auxiliary_response

    # 5) check the conditions of jointness // условия совместности
    for index in range(columns_count, a_auxiliary.shape[0]):
        if x_basis_plan[index] != 0:
            raise Exception('The task is not joint and the method completes its work')

    while max(b_basis_indexes) > columns_count + 1:
        # 6)
        j_k: int = np.max(b_basis_indexes)
        k: int = b_basis_indexes.tolist().index(j_k) + 1
        i: int = j_k - columns_count

        j_vector: list = [i for i in range(1, columns_count + 1) if i <= columns_count + 1 and i
not in b_basis_indexes.tolist()]

        a_base_matrix: np.array = extract_submatrix_by_column_numbers(a_auxiliary,
b_basis_indexes.tolist())
        inverse_matrix: np.array = np.linalg.inv(a_base_matrix)

        for j in j_vector:
            # print('Inverse \n', inverse_matrix)
            # print('submatrix \n', extract_submatrix_by_column_numbers(a_auxiliary, [j]))
            l_j = np.matmul(inverse_matrix, extract_submatrix_by_column_numbers(a_auxiliary,
[j]))
            # print(l_j)

            if l_j[k - 1] != 0:
                b_basis_indexes[k - 1] = j - 1
            else:
                a_auxiliary = np.array(a_auxiliary[:j-1])
```

```
                a = np.array(a[:j-1])

                b_basis_indexes = np.delete(b_basis_indexes, np.argwhere(b_basis_indexes == j_k))
# remove by value
                b = np.delete(b, np.argwhere(b == b[j-1]))

                break

    return x_basis_plan[:columns_count], b_basis_indexes, a, b
```