МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

по дисциплине: Исследование операций

тема: «**Модификации симплекс метода. Методы**

**искусственного базиса и больших штрафов**»

Выполнил: ст. группы ВТ-231

Масленников Даниил

Проверил:

Вирченко Юрий Петрович

Белгород 2025 г.

**Цель работы**:

изучение методов искусственного базиса и больших

штрафов решения задач ЛП в канонической форме, не подготовленных

к работе симплекс-методом в чистом виде.

**Задания для подготовки к работе:**

1. Изучить метод и алгоритм искусственного базиса и составить

программу решения задачи ЛП этим методом.

2. Изучить метод и алгоритм больших штрафов и составить

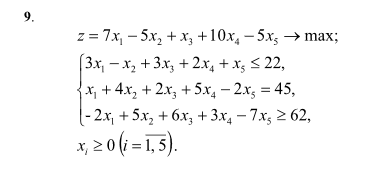
программу решения задачи ЛП этим методом.

3. Запрограммировать изученные алгоритмы и отладить

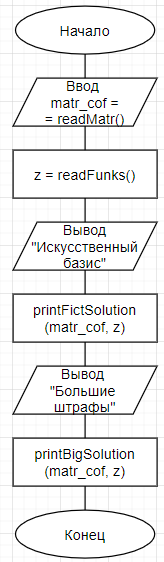
соответствующие программы. В рамках подготовки тестовых

данных решить вручную одну из следующих ниже задач.

# Вариант – 9

****

**Блок-схема программы**

**

**Код программы:**

**from** **math** **import** factorial, fabs

**from** **itertools** **import** \*

*# Функция для ввода матрицы*

**def** readMatr():

    print("Введите количество строк: ")

    count\_string = int(input())

    print("Введите матрицу: ")

    matr = [[float(x) **for** x **in** input().split()] **for** y **in** range(count\_string)]

**return** matr

*# Функция для перевода ведущего столбца в единичный*

**def** basic\_collumn(matr\_cof, number\_string, number\_x):

*# Сохраняем значение элемента в ведущей строке и ведущем столбце*

    cof = matr\_cof[number\_string][number\_x]

*# Делаем ведущий элемент равным 1 путем деления всей строки на этот элемент*

**for** i **in** range(len(matr\_cof[number\_string])):

        matr\_cof[number\_string][i] /= cof

*# Вычитаем ведущую строку из всех остальных строк так, чтобы элементы под*

*# ведущим столбцом стали равными нулю*

**for** i **in** range(len(matr\_cof)):

**if** i != number\_string:

*# Находим коэффициент, на который нужно умножить ведущую строку, чтобы*

*# вычесть ее из текущей строки*

            cof = matr\_cof[i][number\_x]

*# Вычитаем ведущую строку, умноженную на найденный коэффициент*

**for** j **in** range(len(matr\_cof[i])):

                matr\_cof[i][j] -= matr\_cof[number\_string][j] \* cof

*# Возвращаем измененную матрицу коэффициентов*

**return** matr\_cof

*# Функция удаления строк с нулевыми коэффициентами и решения несовместных систем*

**def** delZeroOrNesov(matr\_cof):

**for** i **in** range(len(matr\_cof)):

        flag = **True**

*# Проверяем, все ли коэффициенты в текущей строке равны нулю*

**for** j **in** range(len(matr\_cof[i])):

**if** (matr\_cof[i][j] != 0):

                flag = **False**

*# Если все коэффициенты в строке равны нулю, и свободный член также равен*

*# нулю, удаляем строку*

**if** flag & (matr\_cof[i][-1] == 0):

            matr\_cof.pop(i)

*# Если все коэффициенты в строке равны нулю, но свободный член не равен*

*# нулю, система несовместима*

**elif** flag:

            print("Ваша система несовместима")

            exit(6)

*# Функция для получения списка базисных переменных*

**def** getBasicList(matr\_cof):

    basicList = []  *# Список базисных переменных*

    usage\_string = []  *#Список номеров строк,в которых базисные переменные равны 1*

*# Проходим по столбцам матрицы коэффициентов*

**for** i **in** range(len(matr\_cof[0])):

        count\_one = 0  *# Счетчик единиц в столбце*

*# Счетчик других значений (не нулей и не единиц) в столбце*

        count\_another = 0

        one\_index = 0  *# Индекс строки, в которой нашлась единица*

*# Проходим по строкам столбца*

**for** j **in** range(len(matr\_cof)):

*# Если текущий элемент равен 1, увеличиваем счетчик единиц и*

*# запоминаем индекс строки*

**if** matr\_cof[j][i] == 1:

                count\_one += 1

                one\_index = j

*# Если текущий элемент не равен 0 и не равен 1, увеличиваем счетчик*

*# других значений*

**elif** matr\_cof[j][i] != 0:

                count\_another += 1

*# Если в столбце ровно одна единица и других значений нет, добавляем*

*# столбец в список базисных переменных*

**if** ((count\_one == 1) & (count\_another == 0)):

            basicList.append(i)

            usage\_string.append(one\_index)

*# Возвращаем список базисных переменных и список номеров строк*

**return** [basicList, usage\_string]

*# Функция для проверки равенства базисных переменных*

**def** checkEqual(arr, basic, comb):

*# Проходим по всем парам элементов массива*

**for** i **in** range(len(arr) - 1):

**for** j **in** range(i + 1, len(arr)):

*# Если элементы равны и соответствующие базисные переменные находятся*

*# в заданной комбинации*

**if** (arr[i] == arr[j]) & (basic[i] **in** comb) & (basic[j] **in** comb):

**return** **True**  *# Возвращаем True, если найдены равные базисные*

*# переменные*

**return** **False**  *# Возвращаем False, если равных базисных переменных не найдено*

*# Функция для получения базисной матрицы*

**def** getBasicMatr(matr\_cof, comb):

*# Создаем копию матрицы коэффициентов, чтобы не изменять исходную матрицу*

    new\_matr = [[el **for** el **in** row] **for** row **in** matr\_cof]

*# Получаем списки базисных переменных и соответствующих им строк матрицы*

    basic\_list, basic\_string = getBasicList(new\_matr)

    usage\_strings = []  *# Список использованных строк*

*# Проходим по всем базисным переменным из комбинации*

**for** el **in** comb:

**if** el **not** **in** basic\_list:  *# Если переменная не является базисной*

*# Находим строку, где можно сделать эту переменную базисной*

**for** i **in** range(len(new\_matr)):

**if** (new\_matr[i][el] != 0) & (i **not** **in** usage\_strings):

*# Переводим соответствующий столбец в единичный, чтобы сделать*

*# переменную базисной*

                    basic\_collumn(new\_matr, i, el)

*# Добавляем использованную строку в список*

                    usage\_strings.append(i)

**break**

*# Удаляем строки с нулевыми коэффициентами и проверяем на*

*# несовместность*

            delZeroOrNesov(new\_matr)

*# Обновляем списки базисных переменных*

            basic\_list, basic\_string = getBasicList(new\_matr)

*# Проверяем, не равны ли какие-либо базисные переменные, и возвращаем*

*# пустую матрицу, если равны*

**if** checkEqual(basic\_string, basic\_list, comb):

**return** []

**else**:

*# Если переменная уже является базисной, добавляем соответствующую*

*# строку в список использованных строк*

            usage\_strings.append(basic\_string[basic\_list.index(el)])

**return** new\_matr  *# Возвращаем базисную матрицу*

*# Функция для чтения функции*

**def** readFunck():

    print("Input Z: ")

**return** [float(x) **for** x **in** input().split()]

*# Функция для получения первой таблицы*

**def** getFirstTable(matr\_cof, z):

*# Получаем списки базисных переменных и соответствующих им строк матрицы*

    basic, stringBasic = getBasicList(matr\_cof)

*# Выделяем базисные переменные и соответствующие им строки из списков*

    basic = [basic[i] **for** i **in** range(len(basic) - len(matr\_cof), len(basic))]

    stringBasic = [stringBasic[i] **for** i **in** range(

        len(stringBasic) - len(matr\_cof), len(stringBasic))]

*# Создаем пустую таблицу с нужными размерами и заполняем ее заголовками*

    table = [[0 **for** y **in** range(len(matr\_cof[0]) + 1)]

**for** x **in** range(len(basic) + 2)]

    table[0][0] = "Базисные переменные"

    table[0][1] = "Свободные члены"

**for** i **in** range(2, len(table[0])):

        table[0][i] = 'X' + str(i - 1)

*# Заполняем таблицу базисными переменными и соответствующими им свободными*

*# членами*

**for** x **in** range(len(basic)):

        table[stringBasic[x] + 1][0] = 'X' + str(basic[x] + 1)

**for** i **in** range(1, len(table) - 1):

        table[i][1] = matr\_cof[i - 1][len(matr\_cof[0]) - 1]

*# Заполняем остальные ячейки таблицы коэффициентами из матрицы коэффициентов*

**for** i **in** range(len(matr\_cof)):

**for** j **in** range(len(matr\_cof[0]) - 1):

            table[i + 1][j + 2] = matr\_cof[i][j]

*# Заполняем строку функции Z коэффициентами и свободным членом*

    table[len(table) - 1][0] = 'Z'

    table[len(table) - 1][1] = z[0]

**for** i **in** range(1, len(z)):

        table[len(table) - 1][i + 1] = -z[i]

**return** table

*# Функция для проверки решения*

**def** checkSolution(table):

**for** i **in** range(2, len(table[0])):

**if** (table[len(table) - 1][i] < 0):

**return** **False**

**return** **True**

*# Функция для получения опорного столбца*

**def** getOpSt(table):

    max\_index = 2  *# Индекс начального опорного столбца*

*# Проходим по всем столбцам таблицы, начиная с третьего*

**for** i **in** range(2, len(table[0])):

*# Если элемент последней строки в текущем столбце отрицательный*

**if** table[len(table) - 1][i] < 0:

*# Если текущий элемент меньше элемента с индексом max\_index*

**if** table[len(table) - 1][i] < table[len(table) - 1][max\_index]:

                max\_index = i  *# Обновляем индекс опорного столбца*

**return** max\_index  *# Возвращаем индекс опорного столбца*

*# Функция для выполнения следующего шага в методе искусственного базиса*

**def** nextStep(table):

    indexMin = getOpSt(table)  *# Получаем индекс опорного столбца*

    indexChange = 0  *# Инициализируем индекс строки для изменения*

    min\_val = float('inf')  *# Инициализируем минимальное значение*

*# Находим строку для изменения с минимальным отношением свободного члена к*

*# элементу в опорном столбце*

**for** i **in** range(1, len(table) - 1):

**if** table[i][indexMin] > 0:

            ratio = table[i][1] / table[i][indexMin]  *# Вычисляем отношение*

**if** ratio < min\_val:

                min\_val = ratio

                indexChange = i

*# Проверяем, если indexChange все еще равно 0, значит функция не ограничена*

**if** indexChange == 0:

        print("Функция не ограничена")

        exit(5)

    delit = table[indexChange][indexMin]  *# Получаем делитель*

*# Делим все элементы строки, включая свободный член, на значение делителя*

**for** i **in** range(1, len(table[indexChange])):

        table[indexChange][i] /= delit

    newTable = [[y **for** y **in** x] **for** x **in** table]  *# Создаем новую таблицу*

*# Устанавливаем индекс опорного столбца в строке изменения*

    newTable[indexChange][0] = 'X' + str(indexMin - 1)

*# Проходим по всем строкам кроме строки изменения*

**for** i **in** range(1, len(table)):

**if** i != indexChange:

*# Получаем коэффициент для умножения строки изменения*

            cof = -table[i][indexMin]

*# Добавляем к строке i строку изменения, умноженную на коэффициент cof*

**for** j **in** range(1, len(table[i])):

                newTable[i][j] = table[i][j] + table[indexChange][j] \* cof

**return** newTable  *# Возвращаем обновленную таблицу*

*# Функция для вывода таблицы*

**def** printTable(table):

**for** str **in** table:

**for** el **in** str:

            print(el, end=" ")

        print()

*# Функция для вывода решения*

**def** print\_solution(table):

    print("Решение")

    solution = [0 **for** x **in** range(len(table[0]) - 2)]

**for** i **in** range(1, len(table) - 1):

        print(int(table[i][0][1]))

        solution[int(table[i][0][1]) - 1] = table[i][1]

    print(solution)

    print("Значение в  точке максимума")

    print("Zmax =", table[len(table) - 1][1])

*# Функция для вывода вспомогательной таблицы*

**def** printFict(table, n):

**for** i **in** range(len(table)-1):

**for** j **in** range(len(table[0])):

            print(table[i][j], end=" ")

        print()

*# Функция для получения опорного столбца во вспомогательной таблице*

**def** getOpStlog(table):

    max\_index = 2  *# Индекс начального опорного столбца*

*# Проходим по всем столбцам таблицы, начиная с третьего*

**for** i **in** range(2, len(table[0])):

*# Если элемент предпоследней строки в текущем столбце отрицательный*

**if** table[len(table) - 2][i] < 0:

*# Если текущий элемент меньше элемента с индексом max\_index*

**if** table[len(table) - 2][i] < table[len(table) - 2][max\_index]:

                max\_index = i  *# Обновляем индекс опорного столбца*

**return** max\_index  *# Возвращаем индекс опорного столбца*

*# Функция для выполнения следующего шага в вспомогательной таблице*

**def** nextStepFict(table):

    indexMin = getOpStlog(table)  *# Получаем индекс опорного столбца*

    indexChange = 0  *# Инициализируем индекс строки для изменения*

    min\_val = float('inf')  *# Инициализируем минимальное значение*

*# Находим строку для изменения с минимальным отношением свободного члена к*

*# элементу в опорном столбце*

**for** i **in** range(1, len(table) - 2):

**if** table[i][indexMin] > 0:

            ratio = table[i][1] / table[i][indexMin]  *# Вычисляем отношение*

**if** ratio < min\_val:

                min\_val = ratio

                indexChange = i

*# Проверяем, если indexChange не является строкой Y, выбираем первую строку Y*

**if** table[indexChange][0][0] != 'Y':

**for** i **in** range(1, len(table) - 2):

**if** table[i][0][0] == 'Y':

                indexChange = i

**break**

*# Если indexMin равен нулю или больше или равен длине таблицы - 2,*

*# выбираем первый ненулевой столбец*

**if** (table[indexChange][indexMin] == 0) **or** (indexMin >= len(table[0]) –

len(table) + 2):

**for** i **in** range(2, len(table[0]) - len(table) + 2):

**if** abs(table[indexChange][i]) > 1e-5:

                    indexMin = i

    delit = table[indexChange][indexMin]  *# Получаем делитель*

**for** i **in** range(1, len(table[indexChange])):

*# Делим все элементы строки на значение делителя*

        table[indexChange][i] /= delit

    newTable = [[y **for** y **in** x] **for** x **in** table]  *# Создаем новую таблицу*

*# Устанавливаем индекс опорного столбца в строке изменения*

    newTable[indexChange][0] = 'X' + str(indexMin - 1)

*# Проходим по всем строкам, кроме строки изменения*

**for** i **in** range(1, len(table)):

**if** i != indexChange:

*# Получаем коэффициент для умножения строки изменения*

            cof = -table[i][indexMin]

*# Добавляем к строке i строку изменения, умноженную на коэффициент cof*

**for** j **in** range(1, len(table[i])):

                newTable[i][j] = table[i][j] + table[indexChange][j] \* cof

**return** newTable  *# Возвращаем обновленную таблицу*

*# Функция для добавления фиктивных переменных*

**def** putFict(matr):

*# Создаем новую матрицу, увеличивая количество столбцов на количество строк*

    newMatr = [[0.0 **for** j **in** range(len(matr[0]) + len(matr))]

**for** i **in** range(len(matr))]

*# Заполняем новую матрицу*

**for** i **in** range(len(matr)):

**for** j **in** range(len(matr[0]) - 1):

*# Если свободный член отрицательный, меняем знак элементов матрицы*

**if** matr[i][len(matr[0]) - 1] < 0:

                newMatr[i][j] = -matr[i][j]

**else**:

                newMatr[i][j] = matr[i][j]

*# Добавляем фиктивные переменные в последние столбцы*

**for** i **in** range(len(matr)):

        newMatr[i][len(matr[0]) + i - 1] = 1.0

*# Заполняем последний столбец матрицы значениями свободных членов*

**for** i **in** range(len(matr)):

**if** matr[i][len(matr[0]) - 1] < 0:

            newMatr[i][len(newMatr[0]) - 1] = -matr[i][len(matr[0]) - 1]

**else**:

            newMatr[i][len(newMatr[0]) - 1] = matr[i][len(matr[0]) - 1]

**return** newMatr  *# Возвращаем новую матрицу с добавленными фиктивными*

*# переменными*

*# Функция для вычисления значений Z в вспомогательной таблице*

**def** fictZ(matr):

    zf = [0.0 **for** i **in** range(len(matr[0]))]

**for** j **in** range(len(matr)):

        zf[0] -= matr[j][len(matr[0]) - 1]

**for** i **in** range(1, len(zf) - len(matr)):

**for** j **in** range(len(matr)):

            zf[i] += matr[j][i - 1]

**return** zf

*# Функция для получения таблицы с фиктивным базисом*

**def** getFictBasisTable(matr, z):

*# Добавляем фиктивные переменные к матрице*

    newMatr = putFict(matr)

*# Вычисляем значения функции Z с учетом фиктивных переменных*

    zf = fictZ(newMatr)

*# Получаем первоначальную таблицу с фиктивным базисом*

    table = getFirstTable(newMatr, zf)

*# Добавляем строку для функции Z*

    table.append([0 **for** i **in** range(len(table[0]))])

    table[len(table) - 1][0] = 'Z'

    table[len(table) - 1][1] = z[0]

**for** i **in** range(1, len(z)):

        table[len(table) - 1][i + 1] = -z[i]

*# Преобразуем переменные Y в соответствии с таблицей*

**for** i **in** range(1, len(table) - 2):

        table[i][0] = 'Y' + table[i][0][1]

*# Выводим таблицу на экран перед выполнением искусственного базиса*

    printFict(table, len(matr[0]) - 1)

*# Проводим шаги искусственного базиса*

**for** i **in** range(len(matr)):

        table = nextStepFict(table)

        printFict(table, len(matr[0]) - 1)

*# Удаляем лишнюю строку с искусственной функцией Z*

    newTable = [[0 **for** i **in** range(len(table[0]) - len(matr))]

**for** i **in** range(len(table))]

**for** i **in** range(len(newTable)):

**for** j **in** range(len(newTable[0])):

            newTable[i][j] = table[i][j]

    newTable.pop(len(newTable) - 2)

    newTable[len(newTable) - 1][0] = 'Z'

**return** newTable

*# Функция для вывода решения с фиктивными переменными*

**def** printFictSolution(matr, z):

    table = getFictBasisTable(matr, z)

    printTable(table)

**while** **not** checkSolution(table):

        table = nextStep(table)

        printTable(table)

    print\_solution(table)

*# Функция для вычисления значений Z с большими штрафами*

**def** getMz(matr, z, M):

*# Инициализируем массив значений функции Z*

    zf = [0.0 **for** i **in** range(len(matr[0]))]

*# Добавляем к Z исходные коэффициенты функции*

**for** i **in** range(len(z)):

        zf[i] += z[i]

*# Вычисляем значения с учетом больших штрафов (M)*

**for** j **in** range(len(matr)):

*# Вычитаем из Z значение M \* свободный член*

        zf[0] -= M \* matr[j][len(matr[0]) - 1]

*# Вычисляем остальные значения функции Z с учетом больших штрафов (M)*

**for** i **in** range(1, len(zf) - len(matr)):

**for** j **in** range(len(matr)):

*# Добавляем к Z значение M \* коэффициент*

            zf[i] += M \* matr[j][i - 1]

**return** zf  *# Возвращаем массив значений функции Z*

*# Функция для получения таблицы с большими штрафами*

**def** getBigTable(matr, z):

    M = 1000

    newMatr = putFict(matr)  *# Добавляем фиктивные переменные*

    zf = getMz(newMatr, z, M)  *# Вычисляем значения Z с учетом больших штрафов*

    table = getFirstTable(newMatr, zf)  *# Получаем первоначальную таблицу*

**return** table  *# Возвращаем таблицу*

*# Функция для вывода решения с большими штрафами*

**def** printBigSolution(matr, z):

    table = getBigTable(matr, z)

    printTable(table)

**while** **not** checkSolution(table):

        table = nextStep(table)

        printTable(table)

    print\_solution(table)

*# Основная функция*

**if** \_\_name\_\_ == "\_\_main\_\_":

    matr\_cof = readMatr()

    z = readFunck()

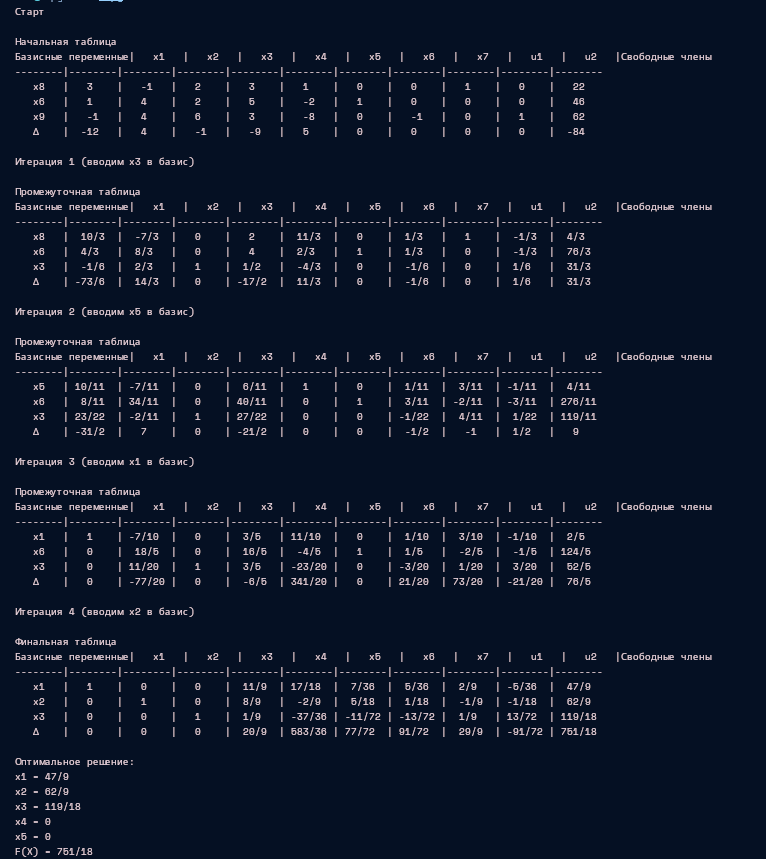
    print("Искусственный базис")

    printFictSolution(matr\_cof, z)

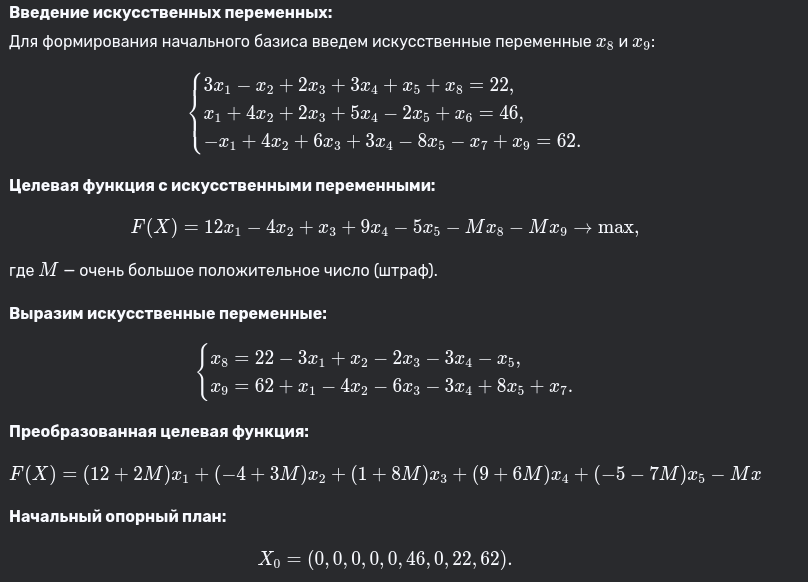
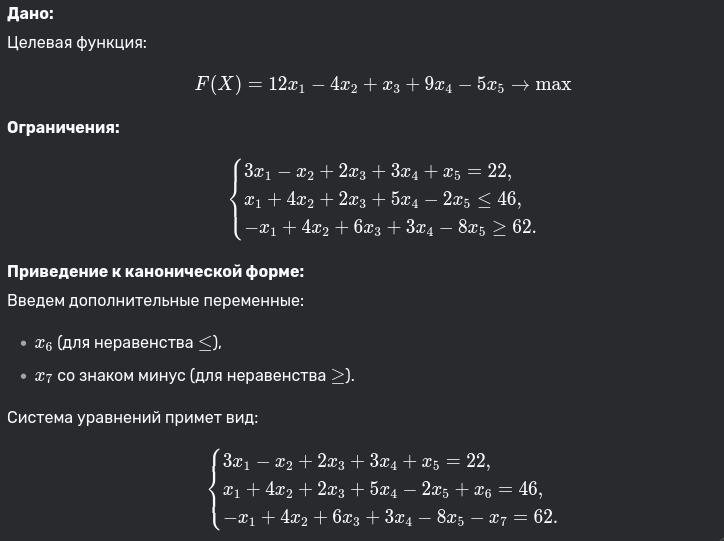
    print("**\n**Большие штрафы")

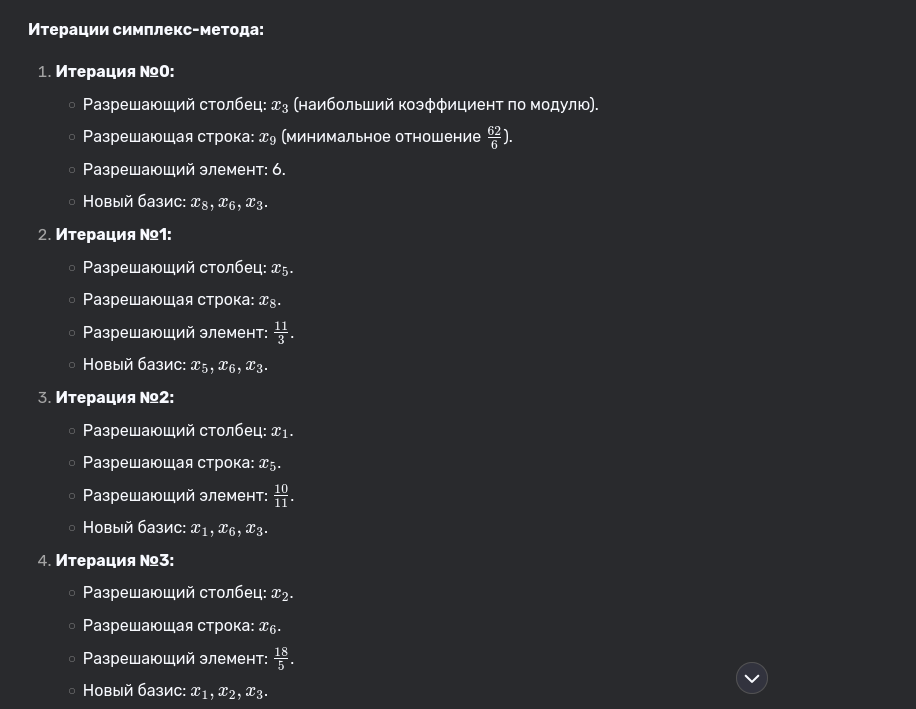
    printBigSolution(matr\_cof, z)

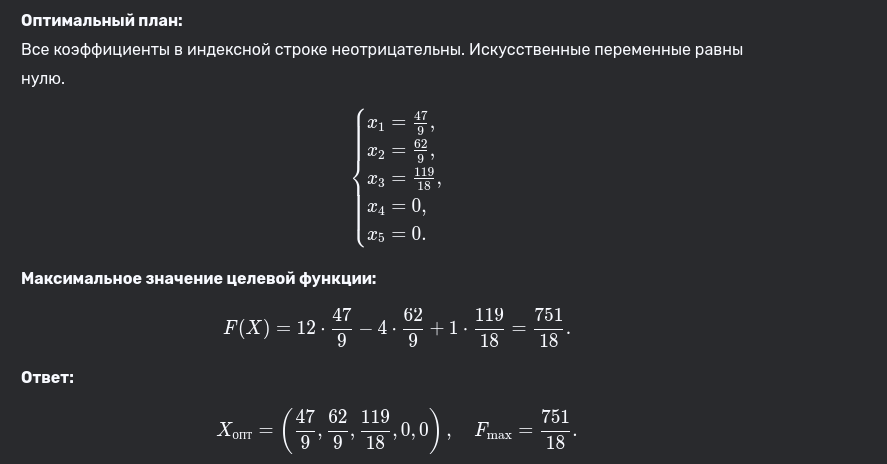
**Результат работы программы**



**Аналитическое решение:**



****

****

**Вывод: в ходе выполнения работы были изучены методы искусственного базиса и больших штрафов решения задач ЛП в канонической форме, не подготовленных к работе симплекс-методом в чистом виде.**