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 №5

по дисциплине: Исследование операций

тема: Двойственный симплекс-метод

Выполнил: ст. группы ПВ-223
Игнатьев Артур
Проверил:
Вирченко Юрий Петрович

Белгород 2024 г.

Цель работы: изучить элементы теории двойственности, двойственный симплекс-метод для пары симметрично двойственных задач, а так же метод последовательного уточнения оценок.

Задания

1. Изучить правило составления двойственных задач, а также формулировки и применения первой, второй и третьей теорем двойственности.

2. Изучить двойственный симплекс-метод для симметрично двойственных задач. Составить и отладить программу решения пары симметрично двойственных задач двойственным симплекс-методом.

3. Изучить понятие псевдоплана, построение симплекс-таблицы, отвечающей псевдоплану. Освоить метод последовательного уточнения оценок. Составить и отладить программу решения задачи ЛП методом последовательного уточнения оценок.

4. Для подготовки тестовых данных решить вручную одну из следующих ниже задач двойственным симплекс-методом для пары симметрично двойственных задач, а также методом последовательного уточнения оценок.

3.

$$z = 2x_1 - 7x_2 - x_3 - 4x_4 \to \max;$$

$$\begin{cases} -3x_1 + x_2 - 4x_3 + x_4 = 20, \\ 2x_1 + 3x_2 + 4x_3 + 2x_4 \geq 32, \\ 4x_1 + 5x_2 + 2x_3 - 3x_4 \geq 26, \end{cases}$$

$$x_i \geq 0 \left(i = \overline{1, 4}\right).$$

Ручной расчет

Решим прямую задачу линейного программирования двойственным симплексным методом, с использованием симплексной таблицы.
Приведем систему ограничений к системе неравенств смысла ≤, умножив соответствующие строки на (-1).

Определим максимальное значение целевой функции $F(X) = 2x_1-7x_2-x_3-4x_4$ при следующих условиях-ограничений.

$-3x_1+x_2-4x_3+x_4=20$

$-2x_1-3x_2-4x_3-2x_4\leq-32$

$-4x_1-5x_2-2x_3+3x_4\leq-26$

Для построения первого опорного плана систему неравенств приведем к системе уравнений путем введения дополнительных переменных (*переход к канонической форме*).

В 2-м неравенстве смысла (≤) вводим базисную переменную x_5. В 3-м неравенстве смысла (≤) вводим базисную переменную x_6.

$-3x_1+x_2-4x_3+x_4 = 20$

$-2x_1-3x_2-4x_3-2x_4+x_5 = -32$

$-4x_1-5x_2-2x_3+3x_4+x_6 = -26$

Введем *искусственные переменные x*: в 1-м равенстве вводим переменную x_7;

$-3x_1+x_2-4x_3+x_4+x_7 = 20$

$-2x_1-3x_2-4x_3-2x_4+x_5 = -32$

$-4x_1-5x_2-2x_3+3x_4+x_6 = -26$

Для постановки задачи на максимум целевую функцию запишем так:
$F(X) = 2x_1-7x_2-1x_3-4x_4 - Mx_7 \rightarrow \max$

За использование искусственных переменных, вводимых в целевую функцию, накладывается так называемый штраф величиной M, очень большое положительное число, которое обычно не задается.

Полученный базис называется искусственным, а метод решения называется методом искусственного базиса.

Причем искусственные переменные не имеют отношения к содержанию поставленной задачи, однако они позволяют построить стартовую точку, а процесс оптимизации вынуждает эти переменные принимать нулевые значения и обеспечить допустимость оптимального решения.

Из уравнений выражаем искусственные переменные: $x_7 = 20+3x_1-x_2+4x_3-x_4$ которые подставим в целевую функцию:

$F(X) = 2x_1-7x_2-x_3-4x_4 - M(20+3x_1-x_2+4x_3-x_4) \rightarrow max$

или

$F(X) = (2-3M)x_1+(-7+M)x_2+(-1-4M)x_3+(-4+M)x_4+(-20M) \rightarrow max$

Матрица коэффициентов A = a(ij) этой системы уравнений имеет вид:

-3	1	-4	1	0	0	1
-2	-3	-4	-2	1	0	0
-4	-5	-2	3	0	1	0

Базисные переменные это переменные, которые входят только в одно уравнение системы ограничений и притом с единичным коэффициентом.

Экономический смысл дополнительных переменных: дополнительные переменные задачи ЛП обозначают излишки сырья, времени, других ресурсов, остающихся в производстве данного оптимального плана.

Решим систему уравнений относительно базисных переменных: x_7, x_5, x_6

Полагая, что **свободные переменные** равны 0, получим первый опорный план:

X0 = (0,0,0,0,-32,-26,20)

Базисное решение называется допустимым, если оно неотрицательно.

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	20	-3	1	-4	1	0	0	1
x_5	-32	-2	-3	-4	-2	1	0	0
x_6	-26	-4	-5	-2	3	0	1	0
F(X0)	-20M	-2+3M	7-M	1+4M	4-M	0	0	0

1. Проверка критерия оптимальности.

План 0 в симплексной таблице **является псевдопланом**, поэтому определяем ведущие строку и столбец.

2. Определение новой свободной переменной.

Среди отрицательных значений базисных переменных выбираем наибольший по модулю.

Ведущей будет 2-ая строка, а переменную x_5 следует вывести из базиса.

3. Определение новой базисной переменной.

Минимальное значение θ соответствует 4-му столбцу, т.е. переменную x_4 необходимо ввести в базис.

На пересечении ведущих строки и столбца находится разрешающий элемент (РЭ), равный (-2).

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	20	-3	1	-4	1	0	0	1
x_5	-32	-2	-3	-4	-2	1	0	0
x_6	-26	-4	-5	-2	3	0	1	0
F(X0)	-20M	-2+3M	7-M	1+4M	4-M	0	0	0
θ		-2+3M : (-2)	7-M : (-3)	1+4M : (-4)	4-M : (-2)	-	-	-

4. Пересчет симплекс-таблицы.

Выполняем преобразования симплексной таблицы методом Жордано-Гаусса.

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	4	-4	-1/2	-6	0	1/2	0	1
x_4	16	1	3/2	2	1	-1/2	0	0
x_6	-74	-7	-19/2	-8	0	3/2	1	0
F(X0)	-64-4M	-6+4M	1+M	-7+6M	0	2-M	0	0

Представим расчет каждого элемента в виде таблицы:

B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
20-(-32*1):-2	-3-(-2*1):-2	1-(-3*1):-2	-4-(-4*1):-2	1-(-2*1):-2	0-(1*1):-2	0-(0*1):-2	1-(0*1):-2
-32 : -2	-2 : -2	-3 : -2	-4 : -2	-2 : -2	1 : -2	0 : -2	0 : -2
-26-(-32*3):-2	-4-(-2*3):-2	-5-(-3*3):-2	-2-(-4*3):-2	3-(-2*3):-2	0-(1*3):-2	1-(0*3):-2	0-(0*3):-2
(0)-(-32*(0)):-2	(-6+4M)-(-2*(0)):-2	(1+M)-(-3*(0)):-2	(-7+6M)-(-4*(0)):-2	(0)-(-2*(0)):-2	(2-M)-(1*(0)):-2	(0)-(0*(0)):-2	(0)-(0*(0)):-2

1. Проверка критерия оптимальности.

План 1 в симплексной таблице **является псевдопланом**, поэтому определяем ведущие строку и столбец.

2. Определение новой свободной переменной.

Среди отрицательных значений базисных переменных выбираем наибольший по модулю.

Ведущей будет 3-ая строка, а переменную x_6 следует вывести из базиса.

3. Определение новой базисной переменной.

Минимальное значение θ соответствует 2-му столбцу, т.е. переменную x_2 необходимо ввести в базис.

На пересечении ведущих строки и столбца находится разрешающий элемент (РЭ), равный ($^{-19}/_{2}$).

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	4	-4	-1/2	-6	0	1/2	0	1
x_4	16	1	3/2	2	1	-1/2	0	0
x_6	-74	-7	-19/2	-8	0	3/2	1	0
F(X0)	-64-4M	-6+4M	1+M	-7+6M	0	2-M	0	0
θ		-6+4M : (-7)	1+M : (-$^{19}/_{2}$)	-7+6M : (-8)	-	-	-	-

4. Пересчет симплекс-таблицы.

Выполняем преобразования симплексной таблицы методом Жордано-Гаусса.

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_7	150/19	-69/19	0	-106/19	0	8/19	-1/19	1
x_4	82/19	-2/19	0	14/19	1	-5/19	3/19	0
x_2	148/19	14/19	1	16/19	0	-3/19	-2/19	0
F(X1)	$^{-1364}/_{19}$-$^{150}/_{19}$M	-$^{128}/_{19}$+$^{69}/_{19}$M	0	-$^{149}/_{19}$+$^{106}/_{19}$M	0	$^{41}/_{19}$-$^{8}/_{19}$M	$^{2}/_{19}$+M	0

Представим расчет каждого элемента в виде таблицы:

B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
4-(-74*$^{-1}/_2$):$^{-19}/_2$	-4-(-7*$^{-1}/_2$):$^{-19}/_2$	$^{-1}/_2$-($^{-19}/_2$*$^{-1}/_2$):$^{-19}/_2$	-6-(-8*$^{-1}/_2$):$^{-19}/_2$	0-(0*$^{-1}/_2$):$^{-19}/_2$	$^{1}/_2$-($^{3}/_2$*$^{-1}/_2$):$^{-19}/_2$	0-(1*$^{-1}/_2$):$^{-19}/_2$	1-(0*$^{-1}/_2$):$^{-19}/_2$
16-(-74*$^{3}/_2$):$^{-19}/_2$	1-(-7*$^{3}/_2$):$^{-19}/_2$	$^{3}/_2$-($^{-19}/_2$*$^{3}/_2$):$^{-19}/_2$	2-(-8*$^{3}/_2$):$^{-19}/_2$	1-(0*$^{3}/_2$):$^{-19}/_2$	$^{-1}/_2$-($^{3}/_2$*$^{3}/_2$):$^{-19}/_2$	0-(1*$^{3}/_2$):$^{-19}/_2$	0-(0*$^{3}/_2$):$^{-19}/_2$
-74 : $^{-19}/_2$	-7 : $^{-19}/_2$	$^{-19}/_2$: $^{-19}/_2$	-8 : $^{-19}/_2$	0 : $^{-19}/_2$	$^{3}/_2$: $^{-19}/_2$	1 : $^{-19}/_2$	0 : $^{-19}/_2$
(0)-(-74*(0)):$^{-19}/_2$	($^{-128}/_{19}$+$^{69}/_{19}$M)-(-7*(0)):$^{-19}/_2$	(0)-($^{-19}/_2$*(0)):$^{-19}/_2$	($^{-149}/_{19}$+$^{106}/_{19}$M)-(-8*(0)):$^{-19}/_2$	(0)-(0*(0)):$^{-19}/_2$	($^{41}/_{19}$-$^{8}/_{19}$M)-($^{3}/_2$*(0)):$^{-19}/_2$	($^{2}/_{19}$+M)-(1*(0)):$^{-19}/_2$	(0)-(0*(0)):$^{-19}/_2$

В базисном столбце все элементы положительные.

Переходим к основному алгоритму симплекс-метода.

Итерация №0.

1. Проверка критерия оптимальности.

Текущий опорный план неоптимален, так как в индексной строке находятся отрицательные коэффициенты.

2. Определение новой базисной переменной.

В качестве ведущего выберем столбец, соответствующий переменной x_5, так как это наибольший коэффициент по модулю.

3. Определение новой свободной переменной.

Вычислим значения D_i по строкам как частное от деления: b_i / a_{i5}
и из них выберем наименьшее:

min ($^{150}/_{19}$: $^{8}/_{19}$, - , -) = $^{75}/_{4}$

Следовательно, 1-ая строка является ведущей.

Разрешающий элемент равен ($^{8}/_{19}$) и находится на пересечении ведущего столбца и ведущей строки.

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7	min
x_7	150/19	-69/19	0	-106/19	0	8/19	-1/19	1	75/4
x_4	82/19	-2/19	0	14/19	1	-5/19	3/19	0	-
x_2	148/19	14/19	1	16/19	0	-3/19	-2/19	0	-
F(X1)	$^{-1364}/_{19}$-$^{150}/_{19}$M	-$^{128}/_{19}$+$^{69}/_{19}$M	0	-$^{149}/_{19}$+$^{106}/_{19}$M	0	$^{41}/_{19}$-$^{8}/_{19}$M	$^{2}/_{19}$+M	0	

4. Пересчет симплекс-таблицы.

Формируем следующую часть симплексной таблицы. Вместо переменной x_7 в план 1 войдет переменная x_5.

Строка, соответствующая переменной x_5 в плане 1, получена в результате деления всех элементов строки x_7 плана 0 на разрешающий элемент РЭ=$^8/_{19}$. На месте разрешающего элемента получаем 1. В остальных клетках столбца x_5 записываем нули.

Таким образом, в новом плане 1 заполнены строка x_5 и столбец x_5. Все остальные элементы нового плана 1, включая элементы индексной строки, определяются по правилу прямоугольника.

Для этого выбираем из старого плана четыре числа, которые расположены в вершинах прямоугольника и всегда включают разрешающий элемент РЭ.

НЭ = СЭ - (А*В)/РЭ

СТЭ - элемент старого плана, РЭ - разрешающий элемент ($^8/_{19}$), А и В - элементы старого плана, образующие прямоугольник с элементами СТЭ и РЭ.

Представим расчет каждого элемента в виде таблицы:

B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
$^{150}/_{19}$: $^8/_{19}$	- $^{69}/_{19}$: $^8/_{19}$	0 : $^8/_{19}$	- $^{106}/_{19}$: $^8/_{19}$	0 : $^8/_{19}$	$^8/_{19}$: $^8/_{19}$	- $^1/_{19}$: $^8/_{19}$	1 : $^8/_{19}$
$^{82}/_{19}$-($^{150}/_{19}$*-$^5/_{19}$):$^8/_{19}$	$^{-2}/_{19}$-(-$^{69}/_{19}$*-$^5/_{19}$):$^8/_{19}$	0-(0*-$^5/_{19}$):$^8/_{19}$	$^{14}/_{19}$-(-$^{106}/_{19}$*-$^5/_{19}$):$^8/_{19}$	1-(0*-$^5/_{19}$):$^8/_{19}$	$^{-5}/_{19}$-($^8/_{19}$*-$^5/_{19}$):$^8/_{19}$	$^3/_{19}$-(-$^1/_{19}$*-$^5/_{19}$):$^8/_{19}$	0-(1*-$^5/_{19}$):$^8/_{19}$

$^{148}/_{19}$-($^{150}/_{19}$*-$^{3}/_{19}$):$^{8}/_{19}$	$^{14}/_{19}$-(-$^{69}/_{19}$*-$^{3}/_{19}$):$^{8}/_{19}$	1-(0*-$^{3}/_{19}$):$^{8}/_{19}$	$^{16}/_{19}$-(-$^{106}/_{19}$*-$^{3}/_{19}$):$^{8}/_{19}$	0-(0*-$^{3}/_{19}$):$^{8}/_{19}$	-$^{3}/_{19}$-($^{8}/_{19}$*-$^{3}/_{19}$):$^{8}/_{19}$	-$^{2}/_{19}$-(-$^{1}/_{19}$*-$^{3}/_{19}$):$^{8}/_{19}$	0-(1*-$^{3}/_{19}$):$^{8}/_{19}$
(0)-($^{150}/_{19}$*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	(-$^{128}/_{19}$+$^{69}/_{19}$M)-(-$^{69}/_{19}$*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	(0)-(0*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	(-$^{149}/_{19}$+$^{106}/_{19}$M)-(-$^{106}/_{19}$*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	(0)-(0*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	($^{41}/_{19}$-$^{8}/_{19}$M)-($^{8}/_{19}$*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	($^{2}/_{19}$+M)-(-$^{1}/_{19}$*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$	(0)-(1*($^{41}/_{19}$-$^{8}/_{19}$M)):$^{8}/_{19}$

Получаем новую симплекс-таблицу:

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_5	75/4	-69/8	0	-53/4	0	1	-1/8	19/8
x_4	37/4	-19/8	0	-11/4	1	0	1/8	5/8
x_2	43/4	-5/8	1	-5/4	0	0	-1/8	3/8
F(X1)	$^{-449}/_{4}$	$^{95}/_{8}$	0	$^{83}/_{4}$	0	0	$^{3}/_{8}$	$^{-41}/_{8}$+M

1. Проверка критерия оптимальности.

Среди значений индексной строки нет отрицательных. Поэтому эта таблица определяет оптимальный план задачи.

Окончательный вариант симплекс-таблицы:

Базис	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_5	75/4	-69/8	0	-53/4	0	1	-1/8	19/8
x_4	37/4	-19/8	0	-11/4	1	0	1/8	5/8
x_2	43/4	-5/8	1	-5/4	0	0	-1/8	3/8
F(X2)	$^{-449}/_4$	$^{95}/_8$	0	$^{83}/_4$	0	0	$^3/_8$	$^{-41}/_8$+M

Так как в оптимальном решении отсутствуют искусственные переменные (они равны нулю), то данное решение является допустимым.

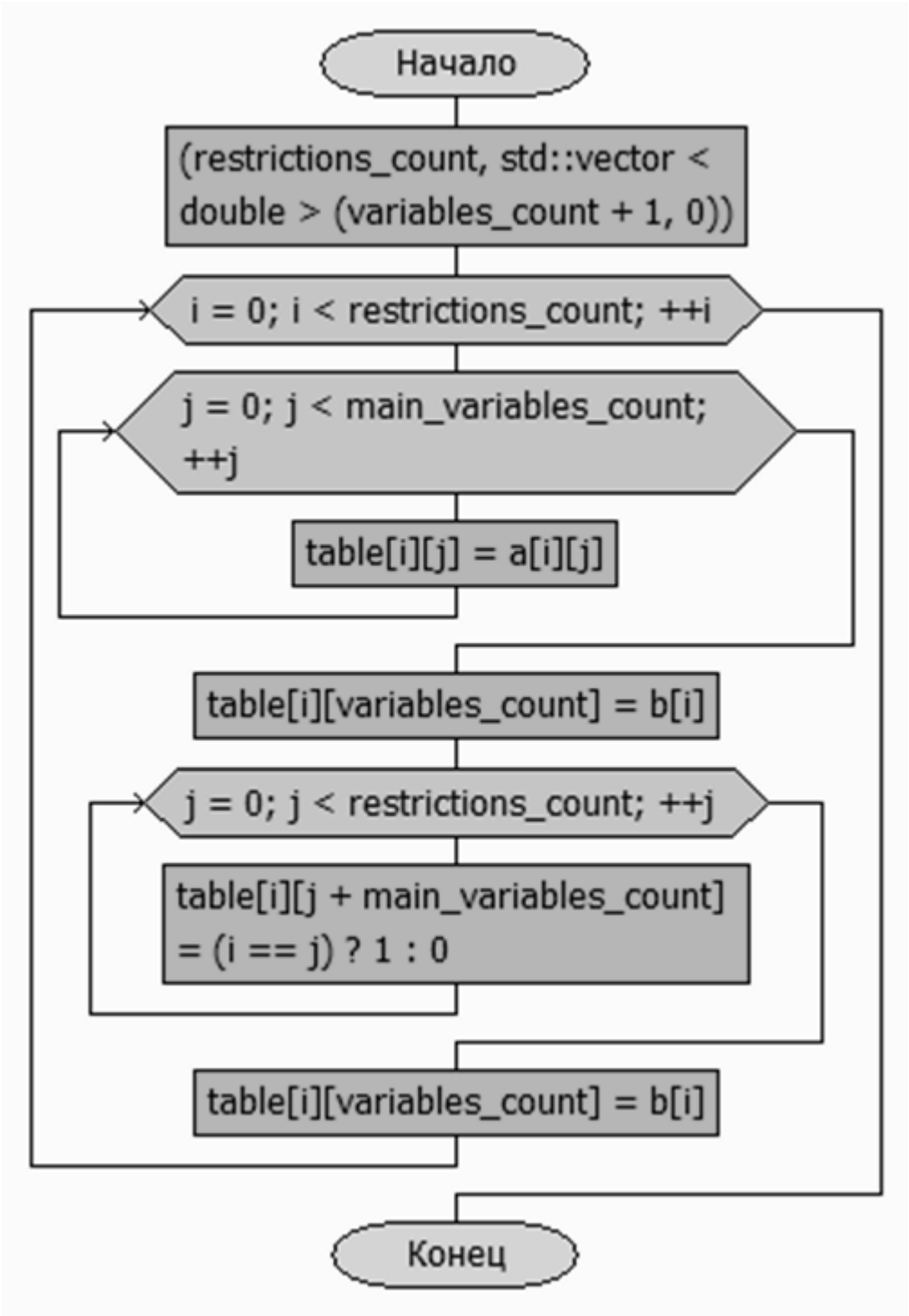
Оптимальный план можно записать так:

$x_1 = 0$, $x_2 = {}^{43}/_4$, $x_3 = 0$, $x_4 = {}^{37}/_4$

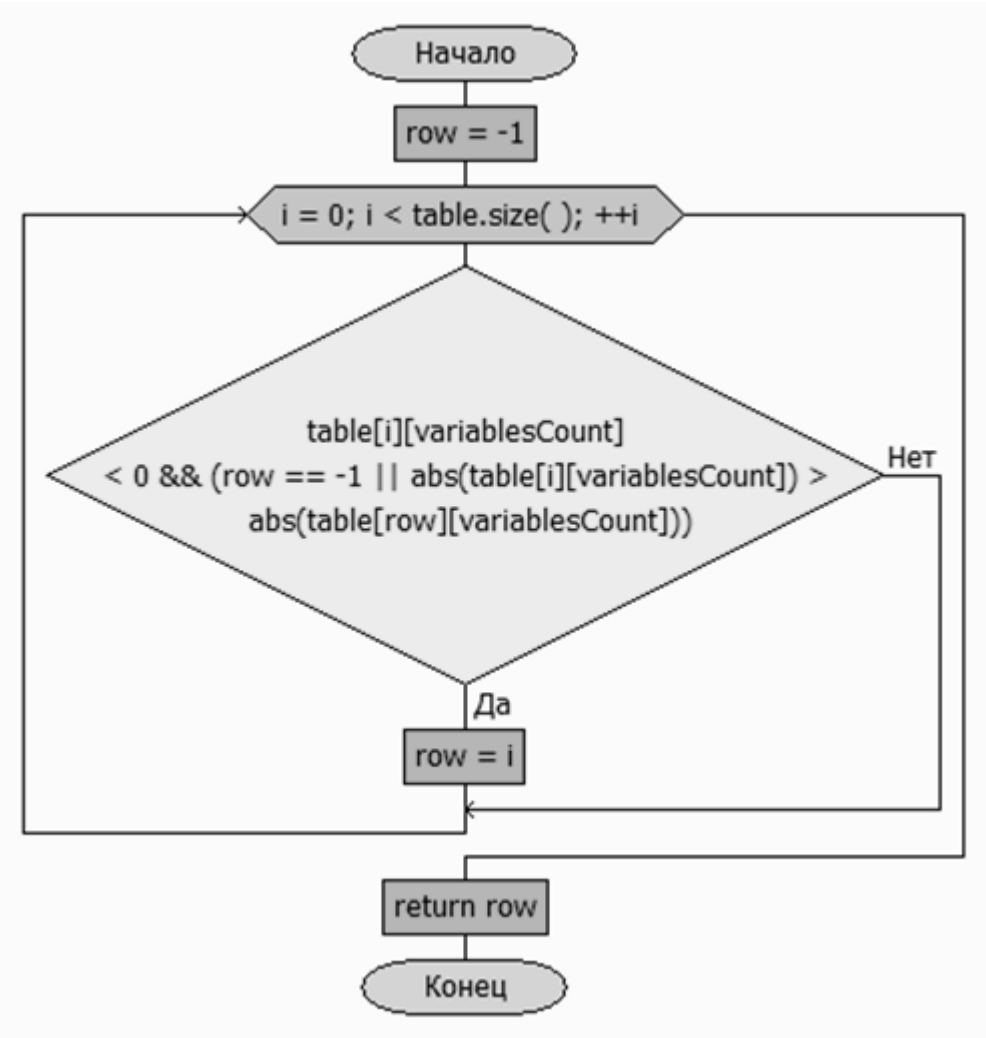
$F(X) = 2*0 -7*{}^{43}/_4 -1*0 -4*{}^{37}/_4 = {}^{-449}/_4$

Блок-схемы:

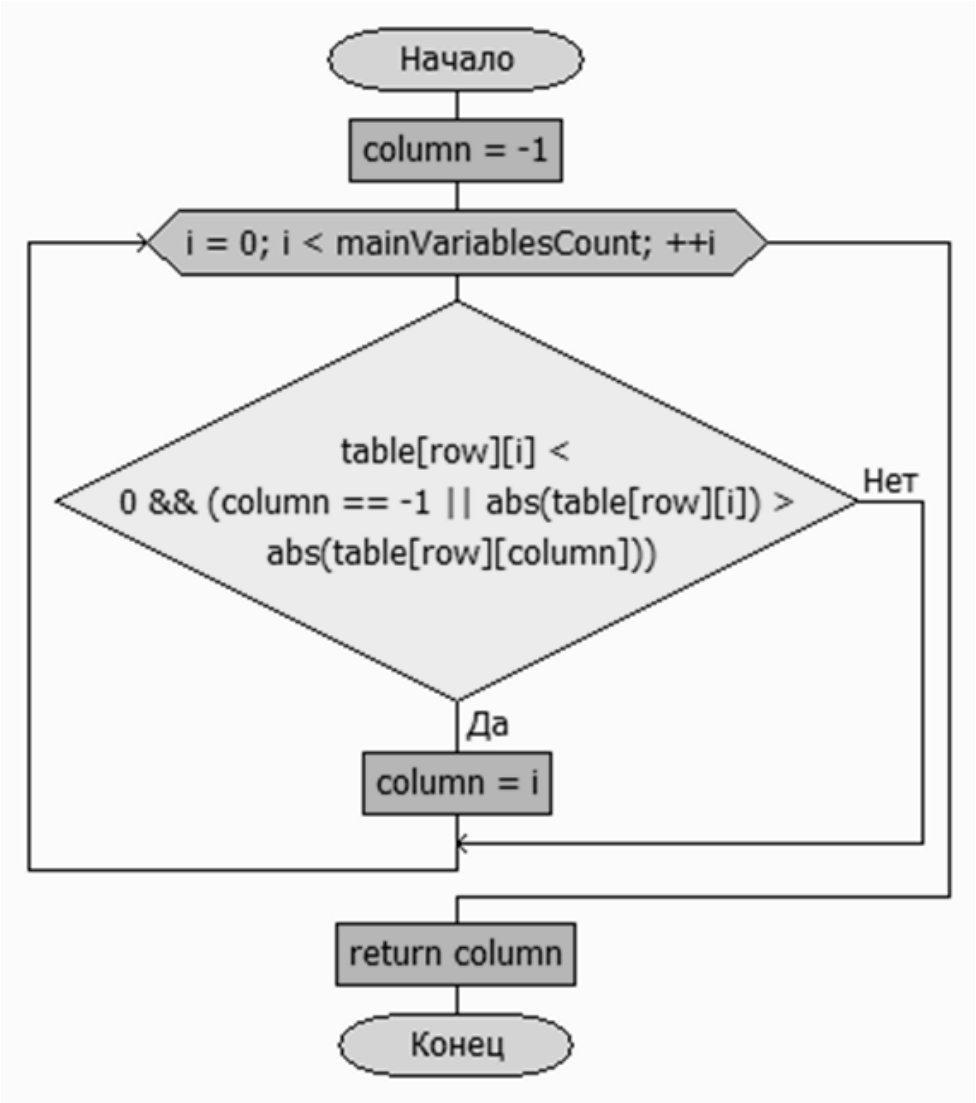
Функция InitTable:



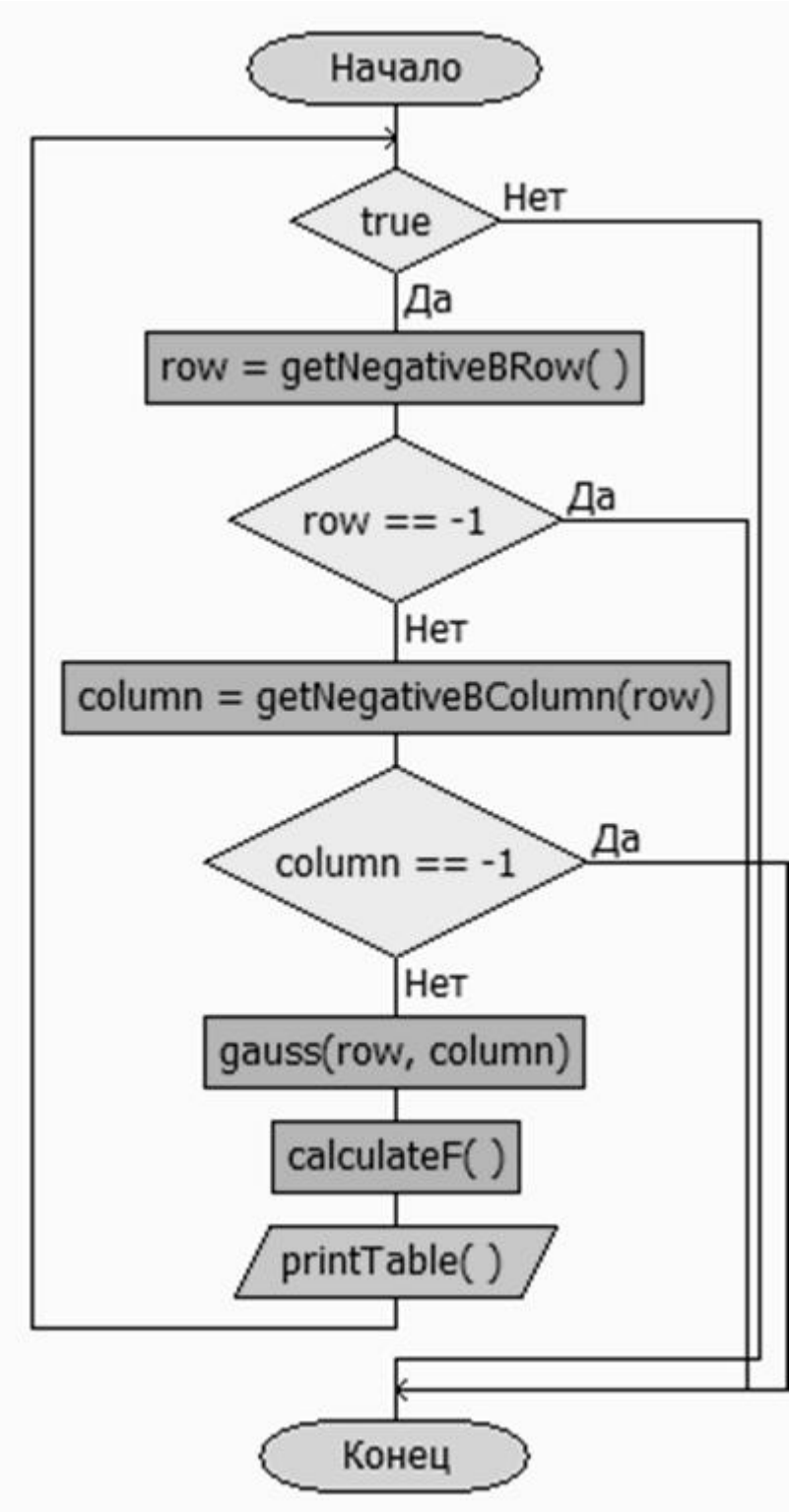
Функция getNegativeBRow



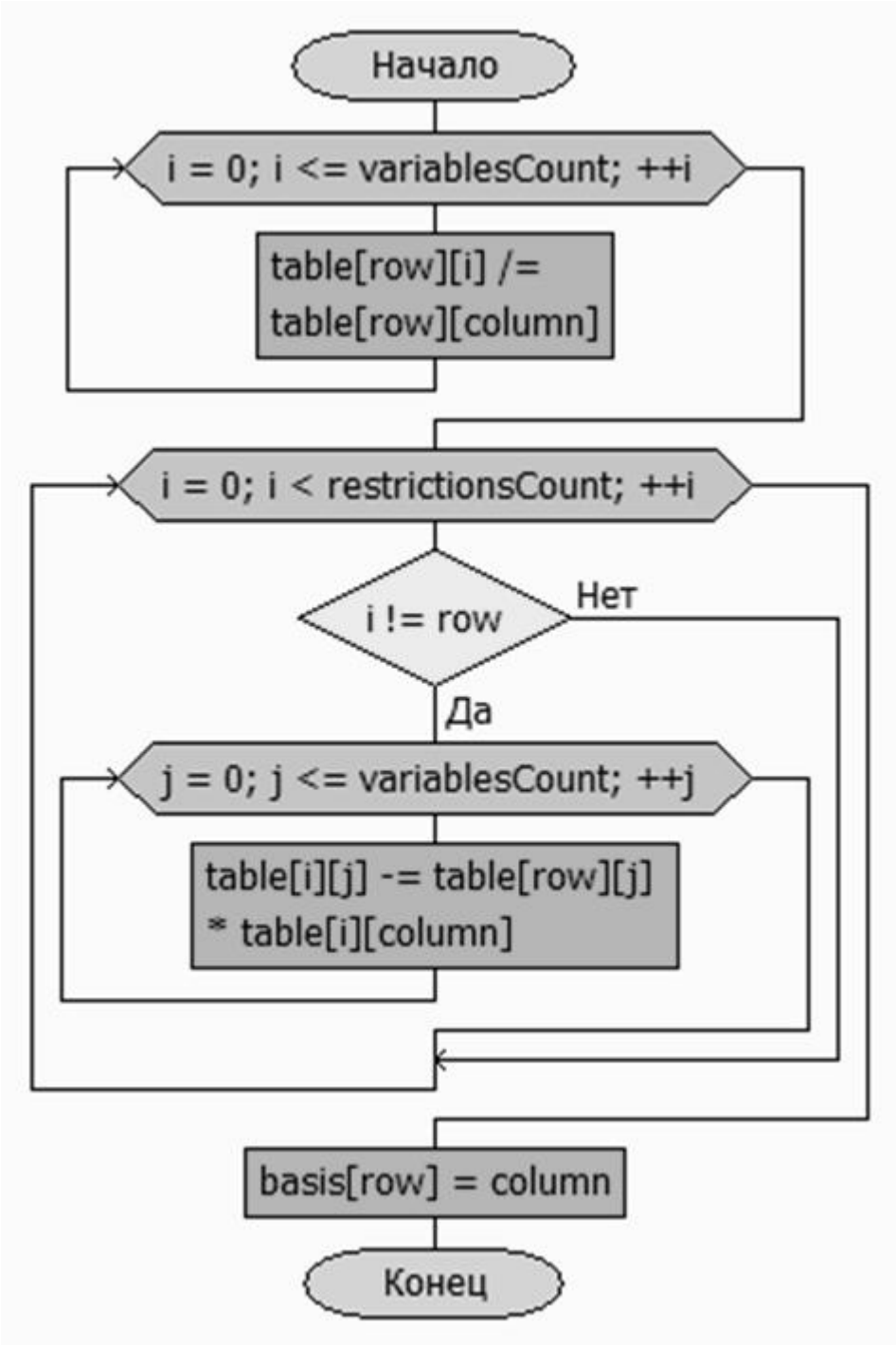
Функция getNegativeBColumn



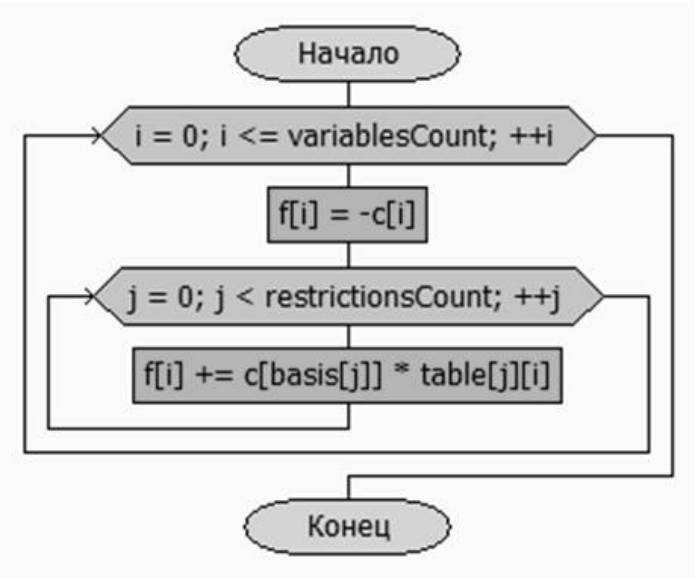
Функция removeNegativeB



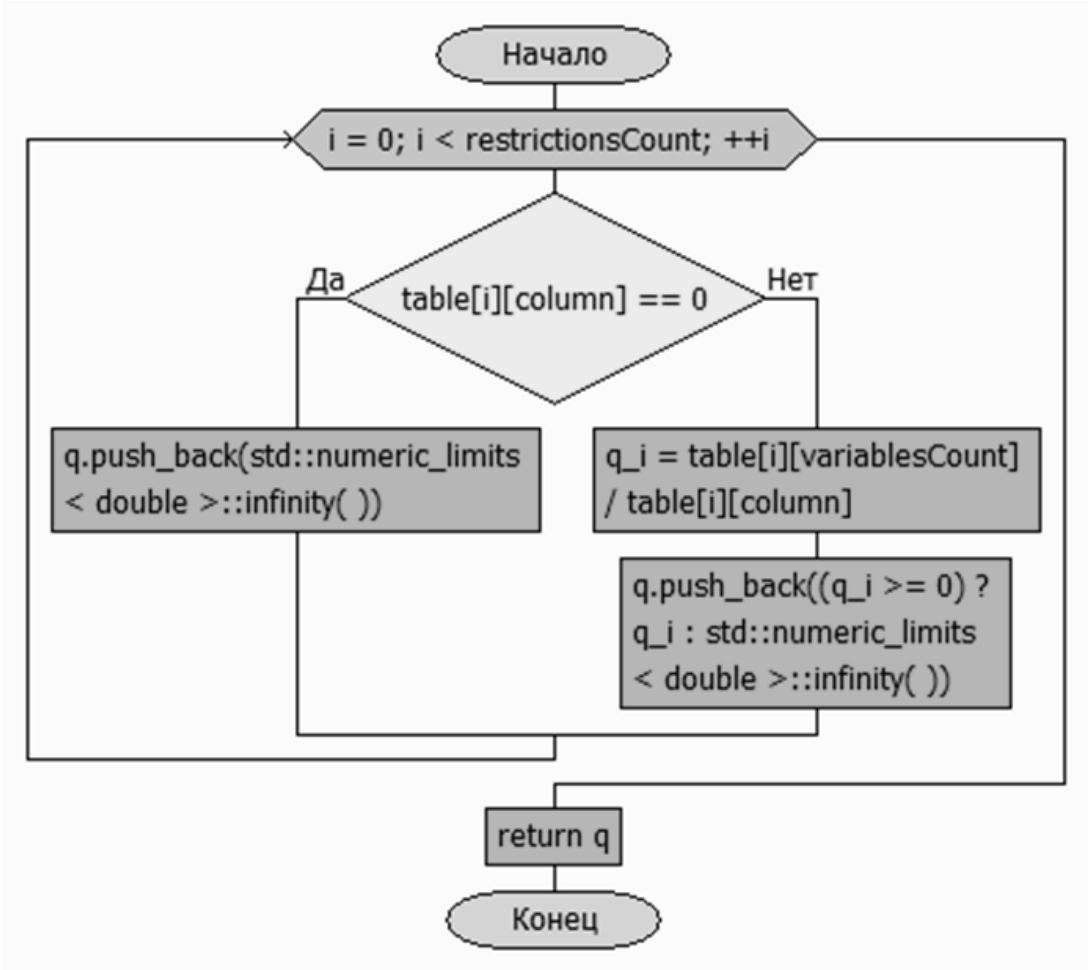
Функция gauss



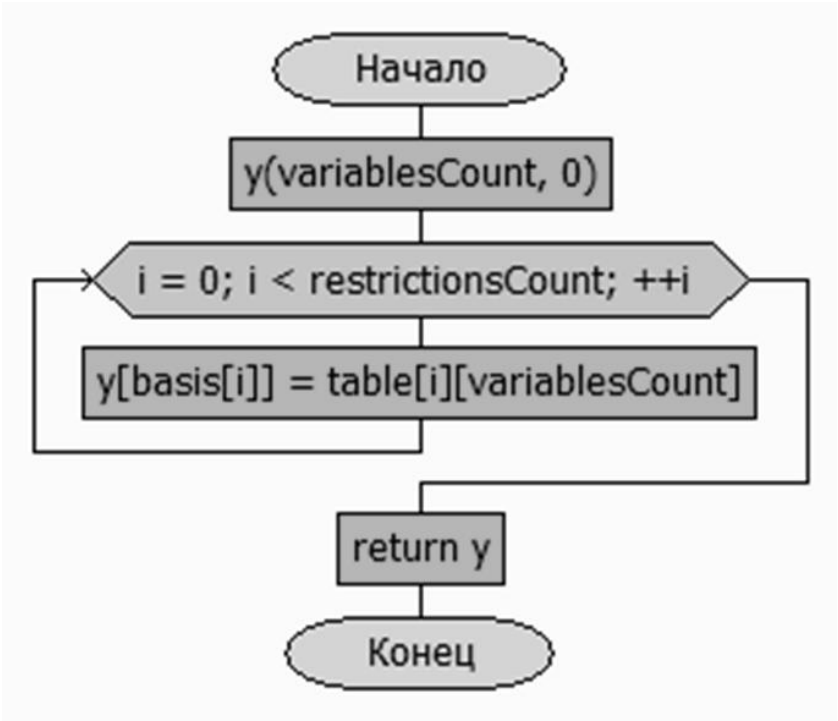
Функция calculateF



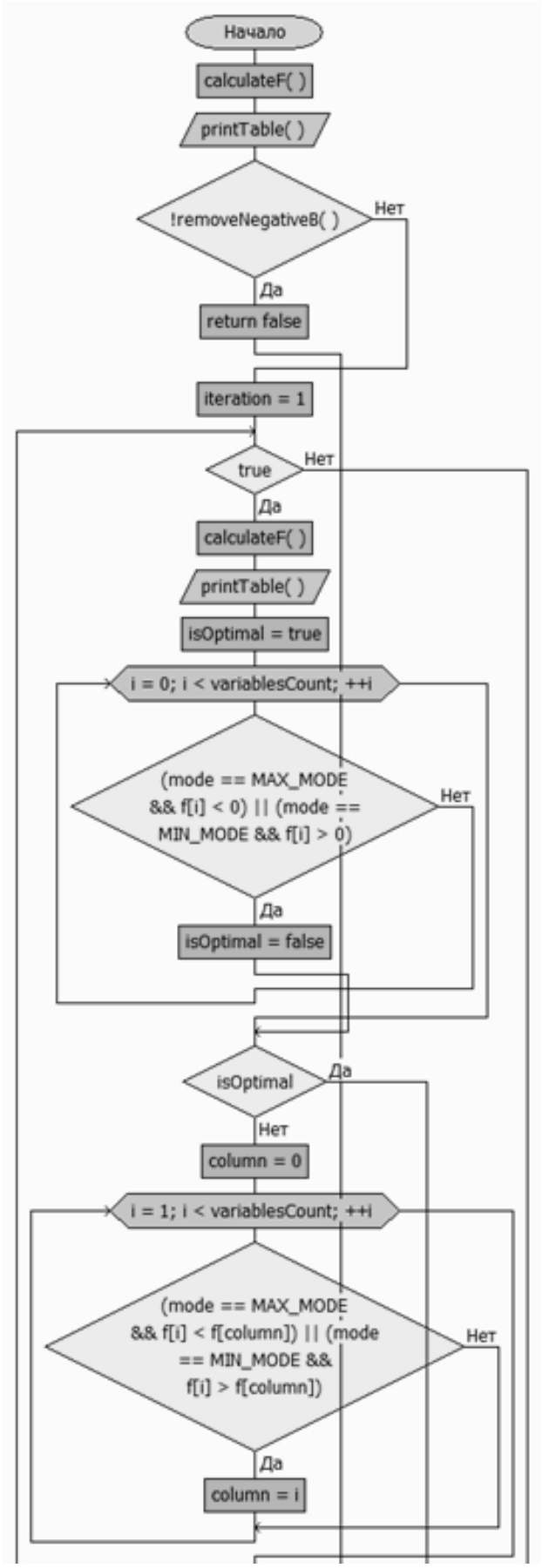
Функция getRelations

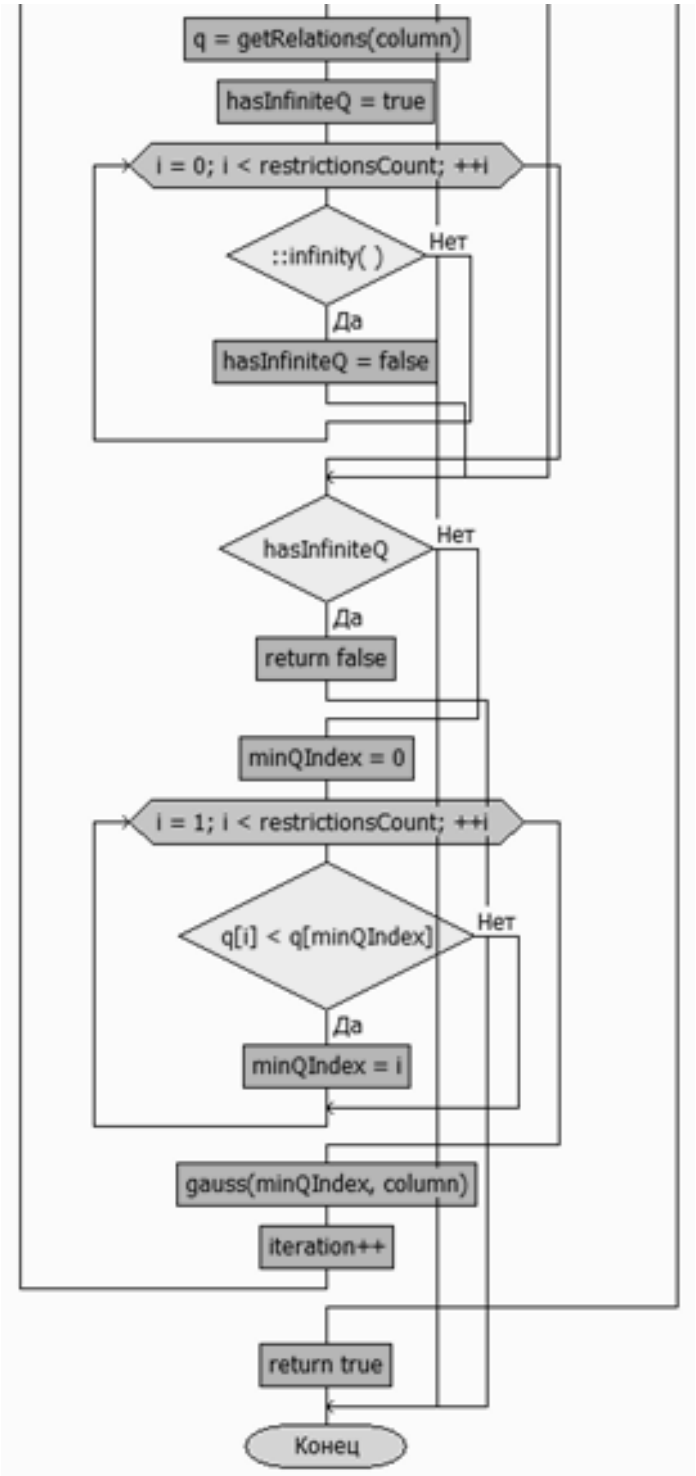


Функция getSolve

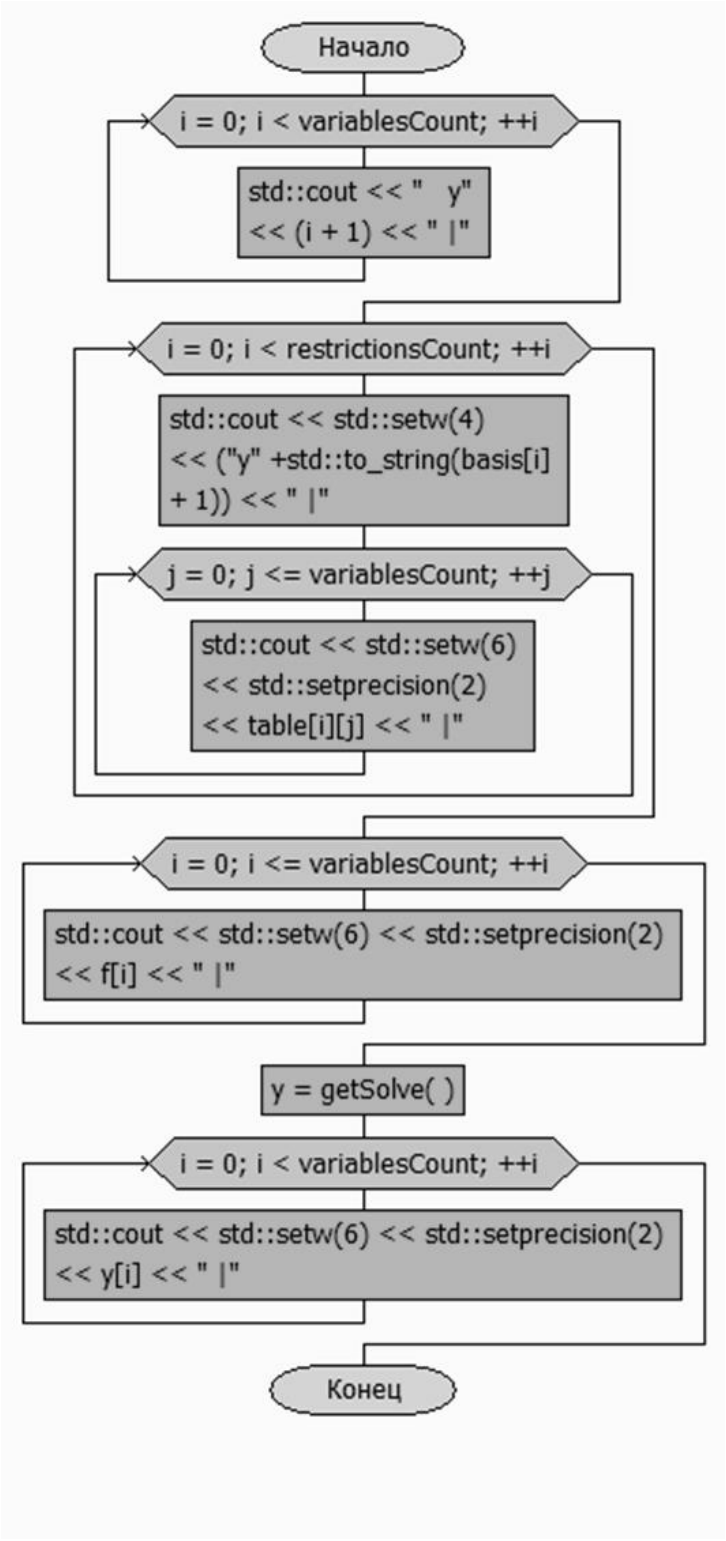


Функция solve

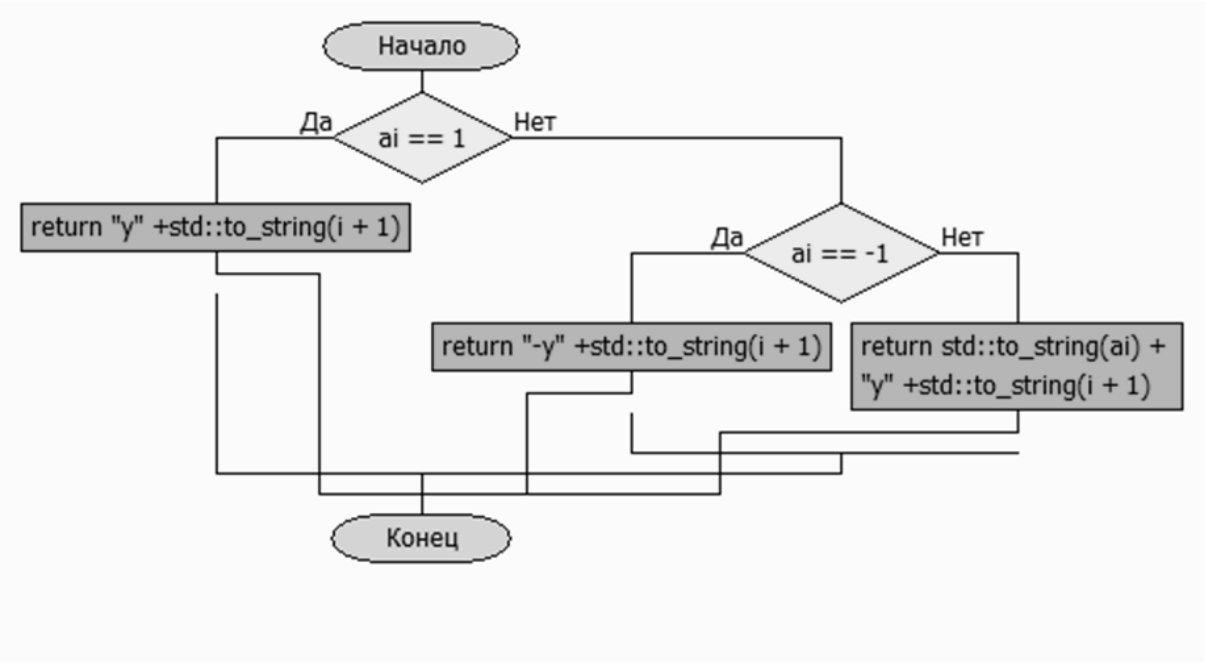



q = getRelations(column)
hasInfiniteQ = true
i = 0; i < restrictionsCount; ++i
::infinity( )
Нет
Да
hasInfiniteQ = false
hasInfiniteQ
Нет
Да
return false
minQIndex = 0
i = 1; i < restrictionsCount; ++i
q[i] < q[minQIndex]
Нет
Да
minQIndex = i
gauss(minQIndex, column)
iteration++
return true
Конец

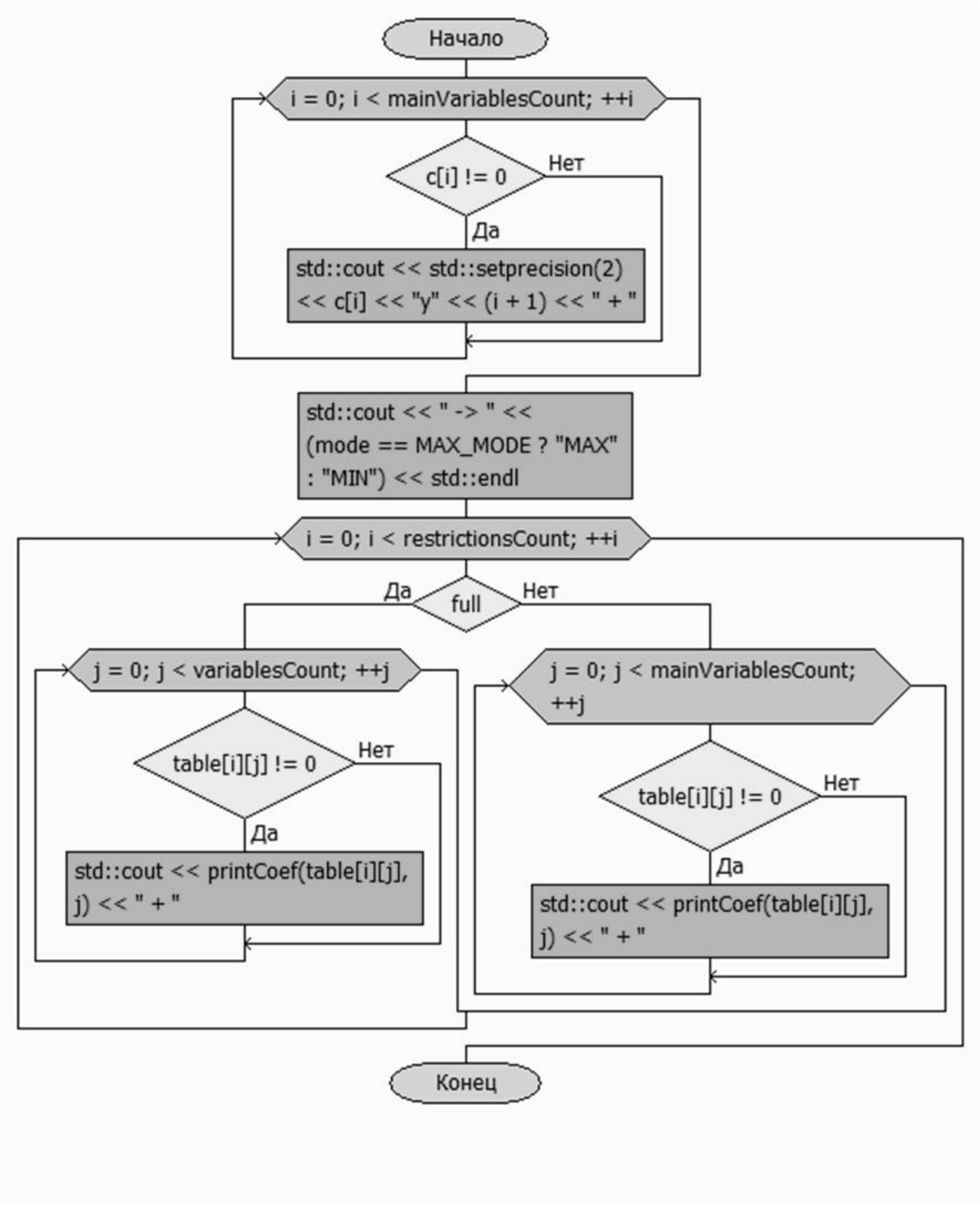
Функция printTable



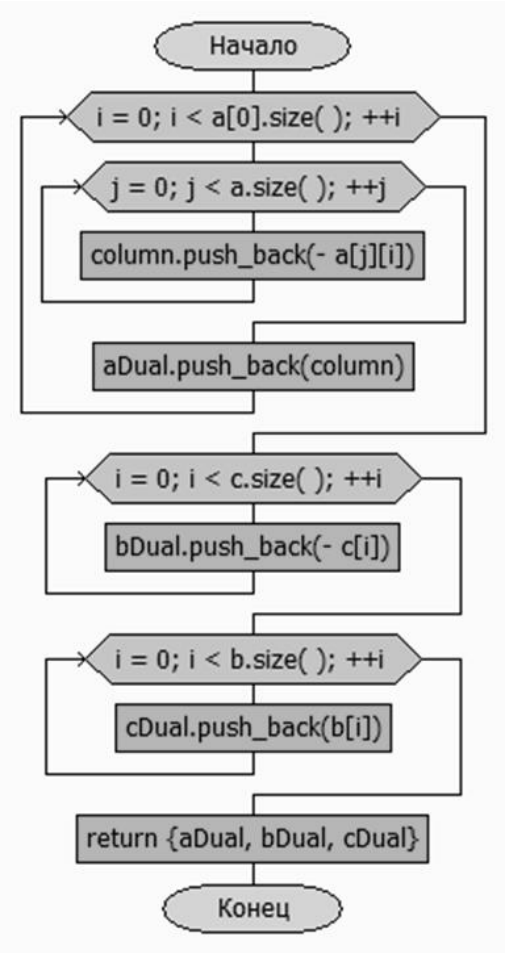
Функция printCoef



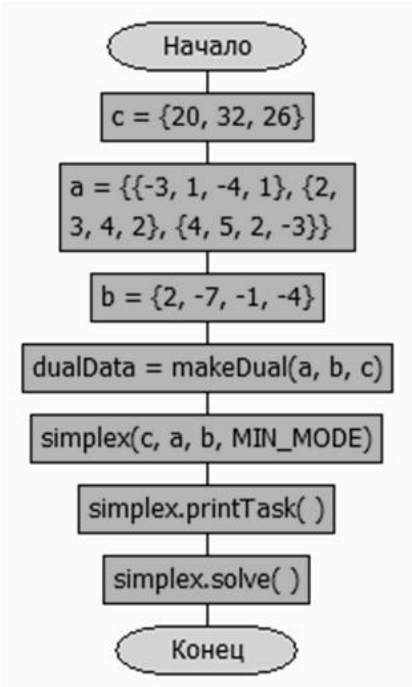
Функция PrintTask



Функция makeDual



Функция main



Код программы:

```
import numpy as np  # Импортируем библиотеку numpy и используем сокращение
np для удобства

MAX_MODE = 'MAX'  # Устанавливаем режим 'MAX' в качестве константы для
максимизации
MIN_MODE = 'MIN'  # Устанавливаем режим 'MIN' в качестве константы для
минимизации


class SimplexMethod:  # Создаем класс SimplexMethod для реализации
симплекс-метода
    def __init__(self, c, a, b, mode):  # Определяем конструктор класса с
параметрами c, a, b, mode
        self.main_variables_count = a.shape[1]  # Определяем количество
основных переменных
        self.restrictions_count = a.shape[0]  # Определяем количество
ограничений
        self.variables_count = self.main_variables_count +
self.restrictions_count  # Определяем общее количество переменных
        self.mode = mode  # Запоминаем режим работы (максимизация или
минимизация)

        self.c = np.concatenate([c, np.zeros((self.restrictions_count +
1))])  # Дополняем вектор c нулями
        self.f = np.zeros((self.variables_count))  # Инициализируем вектор
значений функции F
        self.basis = [i + self.main_variables_count for i in
range(self.restrictions_count)]  # Определяем индексы базисных переменных

        self.init_table(a, b)  # Инициализируем таблицу

    # инициализация таблицы
    def init_table(self, a, b):  # Метод для инициализации таблицы
        self.table = np.zeros((self.restrictions_count,
self.variables_count + 1))  # Создаем таблицу коэффициентов

        for i in range(self.restrictions_count):  # Проходим по всем
ограничениям
            for j in range(self.main_variables_count):  # Проходим по всем
основным переменным
                self.table[i][j] = a[i][j]  # Заполняем соответствующие
значения из матрицы a

            self.table[i][self.variables_count] = b[i]  # Заполняем
значения b

            for j in range(self.restrictions_count):  # Проходим по всем
ограничениям
                self.table[i][j + self.main_variables_count] = int(i == j)
# Заполняем значения для базисных переменных

            self.table[i][-1] = b[i]  # Заполняем значения b еще раз

    # получение строки с максимальным по модулю отрицательным значением b
    def get_negative_b_row(self):  # Метод для получения строки с
максимальным по модулю отрицательным значением b
        row = -1  # Инициализируем переменную row значением -1

        for i, a_row in enumerate(self.table):  # Проходим по всем строкам
```

```
таблицы
            if a_row[-1] < 0 and (row == -1 or abs(a_row[-1]) >
abs(self.table[row][-1])):  # Если значение b отрицательное и больше
предыдущего
                row = i  # Присваиваем переменной row текущее значение

        return row  # Возвращаем найденную строку

    # получение столбца с максимальным по модулю элементом в строке
    def get_negative_b_column(self, row):  # Метод для получения столбца с
максимальным по модулю элементом в строке
        column = -1  # Инициализируем переменную column значением -1

        for i, aij in enumerate(self.table[row][:-1]):  # Проходим по всем
элементам выбранной строки, кроме последнего
            if aij < 0 and (column == -1 or abs(aij) >
abs(self.table[row][column])):  # Если элемент отрицательный и больше
предыдущего
                column = i  # Присваиваем переменной column текущее
значение

        return column  # Возвращаем найденный столбец

    # удаление отрицательных свободных коэффициентов
    def remove_negative_b(self):  # Метод для удаления отрицательных
свободных коэффициентов
        while True:  # Запускаем цикл
            row = self.get_negative_b_row()  # Получаем строку с
отрицательным b

            if row == -1:  # Если такая строка не найдена
                return True  # Возвращаем True

            column = self.get_negative_b_column(row)  # Получаем столбец
для исключения

            if column == -1:  # Если столбец не найден
                return False  # Возвращаем False

            self.gauss(row, column)  # Выполняем исключение Гаусса
            self.calculate_f()  # Пересчитываем значения F
            print('\nNegative b has been removed in row', row + 1)  #
Выводим информацию о удалении отрицательного b
            self.print_table()  # Выводим таблицу

    # выполнение шага метода гаусса
    def gauss(self, row, column):  # Метод для выполнения шага метода
Гаусса
        self.table[row] /= self.table[row][column]  # Делим строку на
значение разрешающего элемента

        for i in range(self.restrictions_count):  # Проходим по всем
ограничениям
            if i != row:  # Если это не выбранная строка
                self.table[i] -= self.table[row] * self.table[i][column]  #
Выполняем исключение Гаусса для других строк

        self.basis[row] = column  # Делаем переменную базисной

    # расчёт значений F
    def calculate_f(self):  # Метод для расчета значений F
```

```
        for i in range(self.variables_count + 1):  # Проходим по всем
переменным
            self.f[i] = -self.c[i]  # Инициализируем значения F

            for j in range(self.restrictions_count):  # Проходим по всем
ограничениям
                self.f[i] += self.c[self.basis[j]] * self.table[j][i]  #
Выполняем расчет значений F

    # расчёт симплекс-отношений для столбца column
    def get_relations(self, column):  # Метод для расчета симплекс-
отношений для столбца column
        q = []  # Инициализируем список q

        for i in range(self.restrictions_count):  # Проходим по всем
ограничениям
            if self.table[i][column] == 0:  # Если элемент равен 0
                q.append(np.inf)  # Добавляем бесконечность в список
            else:
                q_i = self.table[i][-1] / self.table[i][column]  #
Вычисляем симплекс-отношение
                q.append(q_i if q_i >= 0 else np.inf)  # Добавляем значение
в список, если оно неотрицательное, иначе добавляем бесконечность

        return q  # Возвращаем список симплекс-отношений

    # получение решения
    def get_solve(self):  # Метод для получения решения
        y = np.zeros((self.variables_count))  # Инициализируем вектор
решения

        for i in range(self.restrictions_count):  # Проходим по всем
ограничениям
            y[self.basis[i]] = self.table[i][-1]  # Запол

        return y  # Возвращаем вектор решения

    # решение
    def solve(self):  # Метод для решения задачи симплекс-методом
        print('\nIteration 0')  # Выводим информацию о начале итерации
        self.calculate_f()  # Вычисляем значения F
        self.print_table()  # Выводим таблицу

        if not self.remove_negative_b():  # Если не удалось удалить
отрицательные b
            print('Solve does not exist')  # Выводим сообщение о том, что
решения не существует
            return False  # Возвращаем False

        iteration = 1  # Инициализируем переменную итерации значением 1

        while True:  # Запускаем цикл
            self.calculate_f()  # Вычисляем значения F
            print('\nIteration', iteration)  # Выводим информацию о текущей
итерации
            self.print_table()  # Выводим таблицу

            if all(fi >= 0 if self.mode == MAX_MODE else fi <= 0 for fi in
self.f[:-1]):  # Если все значения F неотрицательные (для MAX_MODE) или не
положительные (для MIN_MODE)
                break  # Завершаем работу цикла
```

```
            column = (np.argmin if self.mode == MAX_MODE else
np.argmax)(self.f[:-1])  # Получаем разрешающий столбец
            q = self.get_relations(column)  # Получаем симплекс-отношения
для найденного столбца

            if all(qi == np.inf for qi in q):  # Если не удалось найти
разрешающую строку
                print('Solve does not exist')  # Выводим сообщение о том,
что решения нет
                return False  # Возвращаем False

            self.gauss(np.argmin(q), column)  # Выполняем исключение Гаусса
            iteration += 1  # Увеличиваем значение итерации

        return True  # Возвращаем True, если решение найдено

    # вывод симплекс-таблицы
    def print_table(self):  # Метод для вывода симплекс-таблицы
        print('     |' + ''.join(['   y%-3d |' % (i + 1) for i in
range(self.variables_count)]) + '    b   |')  # Форматированный вывод
заголовков таблицы

        for i in range(self.restrictions_count):  # Проходим по всем
ограничениям
            print('%4s |' % ('y' + str(self.basis[i] + 1)) + ''.join(['
%6.2f |' % aij for j, aij in enumerate(self.table[i])]))  # Выводим
значения из таблицы

        print('   F |' + ''.join([' %6.2f |' % aij for aij in self.f]))  #
Выводим значения F
        print('   y |' + ''.join([' %6.2f |' % xi for xi in
self.get_solve()]))  # Выводим значения переменных

    # вывод коэффициента
    def print_coef(self, ai, i):  # Метод для вывода коэффициента
        if ai == 1:  # Если коэффициент равен 1
            return 'y%d' % (i + 1)  # Возвращаем строку вида 'y1'

        if ai == -1:  # Если коэффициент равен -1
            return '-y%d' % (i + 1)  # Возвращаем строку вида '-y1'

        return '%.2fy%d' % (ai, i + 1)  # Возвращаем строку вида '2.00y1'

    # вывод задачи
    def print_task(self, full=False):  # Метод для вывода задачи
        print(' + '.join(['%.2fy%d' % (ci, i + 1) for i, ci in
enumerate(self.c[:self.main_variables_count]) if ci != 0]), '-> ',
self.mode)  # Выводим целевую функцию и режим

        for row in self.table:  # Проходим по всем строкам таблицы
            if full:  # Если параметр full равен True
                print(' + '.join([self.print_coef(ai, i) for i, ai in
enumerate(row[:self.variables_count]) if ai != 0]), '=', row[-1])  #
Выводим полное описание ограничений
            else:
                print(' + '.join([self.print_coef(ai, i) for i, ai in
enumerate(row[:self.main_variables_count]) if ai != 0]), '<=', row[-1])  #
Выводим краткое описание ограничений

# перевод в двойственную задачу
```

```
def make_dual(a, b, c):  # Функция для перевода задачи в двойственную
    return -a.T, -c, b  # Возвращаем транспонированную матрицу a,
отрицательный вектор c и вектор b

def main():  # Основная функция
    c = np.array([20, 32, 26])  # Определяем вектор c
    a = np.array([  # Определяем матрицу a
        [-3, 1, -4, 1],
        [2, 3, 4, 2],
        [4, 5, 2, -3]
    ])

    b = np.array([2, -7, -1, -4])  # Определяем вектор b

    a, b, c = make_dual(a, b, c)  # Преобразуем задачу в двойственную
    simplex = SimplexMethod(c, a, b, MIN_MODE)  # Создаем объект класса
SimplexMethod с заданными параметрами

    print("Dual task:")  # Выводим информацию о двойственной задаче
    simplex.print_task()  # Выводим описание задачи
    simplex.solve()  # Решаем задачу симплекс-методом

if __name__ == '__main__':  # Проверяем, что скрипт запущен напрямую, а не
импортирован как модуль
    main()  # Вызываем основную функцию
```

Результат работы программы:

Оптимальный план:

	B	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_5	75/4	-69/8	0	-53/4	0	1	-1/8	19/8
x_4	37/4	-19/8	0	-11/4	1	0	1/8	5/8
x_2	43/4	-5/8	1	-5/4	0	0	-1/8	3/8
F	$^{-449}/_4$	$^{95}/_8$	0	$^{83}/_4$	0	0	$^3/_8$	$^{-41}/_8$+M

Значение целевой функции: -112,25

Вывод: Проведено изучение основ теории двойственности, двойственного симплекс-метода для пары симметрично двойственных задач и метода последовательного уточнения оценок. В процессе стало ясно, что двойственность - важный инструмент в линейном программировании, который помогает использовать данные о прямой задаче для решения её двойственной версии. Двойственный симплекс-метод - это изменённый симплекс-метод, который применяется для симметрично двойственных задач. Метод последовательного уточнения оценок помогает улучшить точность решения задачи линейного программирования путём поочередного уточнения и улучшения оценок переменных.