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.

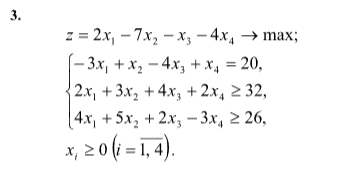
**Задания**

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.



**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) = 2x1-7x2-x3-4x4 при следующих условиях-ограничений.  
-3x1+x2-4x3+x4=20  
-2x1-3x2-4x3-2x4≤-32  
-4x1-5x2-2x3+3x4≤-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 x5. В 3-м неравенстве смысла (≤) вводим базисную переменную x6.  
-3x1+x2-4x3+x4 = 20  
-2x1-3x2-4x3-2x4+x5 = -32  
-4x1-5x2-2x3+3x4+x6 = -26

Введем *искусственные переменные x*: в 1-м равенстве вводим переменную x7;  
-3x1+x2-4x3+x4+x7 = 20  
-2x1-3x2-4x3-2x4+x5 = -32  
-4x1-5x2-2x3+3x4+x6 = -26

Для постановки задачи на максимум целевую функцию запишем так:  
F(X) = 2x1-7x2-1x3-4x4 - Mx7 → max

За использование искусственных переменных, вводимых в целевую функцию, накладывается так называемый штраф величиной М, 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: x7 = 20+3x1-x2+4x3-x4 которые подставим в целевую функцию:  
F(X) = 2x1-7x2-x3-4x4 - M(20+3x1-x2+4x3-x4) → max  
или  
F(X) = (2-3M)x1+(-7+M)x2+(-1-4M)x3+(-4+M)x4+(-20M) → 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: x7, x5, x6

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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x7 | 20 | -3 | 1 | -4 | 1 | 0 | 0 | 1 |
| x5 | -32 | -2 | -3 | -4 | -2 | 1 | 0 | 0 |
| x6 | -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 x5 следует вывести из базиса.

**3. Определение новой базисной переменной**.  
Минимальное значение θ соответствует 4-му столбцу, т.е. переменную x4 необходимо ввести в базис.  
На пересечении ведущих строки и столбца находится разрешающий элемент (РЭ), равный (-2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x7 | 20 | -3 | 1 | -4 | 1 | 0 | 0 | 1 |
| x5 | -32 | -2 | -3 | -4 | -2 | 1 | 0 | 0 |
| x6 | -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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x7 | 4 | -4 | -1/2 | -6 | 0 | 1/2 | 0 | 1 |
| x4 | 16 | 1 | 3/2 | 2 | 1 | -1/2 | 0 | 0 |
| x6 | -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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| 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 x6 следует вывести из базиса.

**3. Определение новой базисной переменной**.  
Минимальное значение θ соответствует 2-му столбцу, т.е. переменную x2 необходимо ввести в базис.  
На пересечении ведущих строки и столбца находится разрешающий элемент (РЭ), равный (-19/2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x7 | 4 | -4 | -1/2 | -6 | 0 | 1/2 | 0 | 1 |
| x4 | 16 | 1 | 3/2 | 2 | 1 | -1/2 | 0 | 0 |
| x6 | -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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x7 | 150/19 | -69/19 | 0 | -106/19 | 0 | 8/19 | -1/19 | 1 |
| x4 | 82/19 | -2/19 | 0 | 14/19 | 1 | -5/19 | 3/19 | 0 |
| x2 | 148/19 | 14/19 | 1 | 16/19 | 0 | -3/19 | -2/19 | 0 |
| F(X1) | -1364/19-150/19M | -128/19+69/19M | 0 | -149/19+106/19M | 0 | 41/19-8/19M | 2/19+M | 0 |

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| B | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| 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+69/19M)-(-7\*(0)):-19/2 | (0)-(-19/2\*(0)):-19/2 | (-149/19+106/19M)-(-8\*(0)):-19/2 | (0)-(0\*(0)):-19/2 | (41/19-8/19M)-(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 x5, так как это наибольший коэффициент по модулю.

**3. Определение новой свободной переменной**.  
Вычислим значения Di по строкам как частное от деления: bi / ai5  
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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | min |
| x7 | 150/19 | -69/19 | 0 | -106/19 | 0 | 8/19 | -1/19 | 1 | 75/4 |
| x4 | 82/19 | -2/19 | 0 | 14/19 | 1 | -5/19 | 3/19 | 0 | - |
| x2 | 148/19 | 14/19 | 1 | 16/19 | 0 | -3/19 | -2/19 | 0 | - |
| F(X1) | -1364/19-150/19M | -128/19+69/19M | 0 | -149/19+106/19M | 0 | 41/19-8/19M | 2/19+M | 0 |  |

**4. Пересчет симплекс-таблицы**.

Формируем следующую часть симплексной таблицы. Вместо переменной x7 в план 1 войдет переменная x5.  
Строка, соответствующая переменной x5 в плане 1, получена в результате деления всех элементов строки x7 плана 0 на разрешающий элемент РЭ=8/19. На месте разрешающего элемента получаем 1. В остальных клетках столбца x5 записываем нули.

Таким образом, в новом плане 1 заполнены строка x5 и столбец x5. 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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| 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\*(41/19-8/19M)):8/19 | (-128/19+69/19M)-(-69/19\*(41/19-8/19M)):8/19 | (0)-(0\*(41/19-8/19M)):8/19 | (-149/19+106/19M)-(-106/19\*(41/19-8/19M)):8/19 | (0)-(0\*(41/19-8/19M)):8/19 | (41/19-8/19M)-(8/19\*(41/19-8/19M)):8/19 | (2/19+M)-(-1/19\*(41/19-8/19M)):8/19 | (0)-(1\*(41/19-8/19M)):8/19 |

Получаем новую симплекс-таблицу:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Базис | B | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x5 | 75/4 | -69/8 | 0 | -53/4 | 0 | 1 | -1/8 | 19/8 |
| x4 | 37/4 | -19/8 | 0 | -11/4 | 1 | 0 | 1/8 | 5/8 |
| x2 | 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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x5 | 75/4 | -69/8 | 0 | -53/4 | 0 | 1 | -1/8 | 19/8 |
| x4 | 37/4 | -19/8 | 0 | -11/4 | 1 | 0 | 1/8 | 5/8 |
| x2 | 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:  
x1 = 0, x2 = 43/4, x3 = 0, x4 = 37/4  
F(X) = 2\*0 -7\*43/4 -1\*0 -4\*37/4 = -449/4

**Блок-схемы:**

Функция InitTable:

Изображение выглядит как текст, снимок экрана, Шрифт, Параллельный

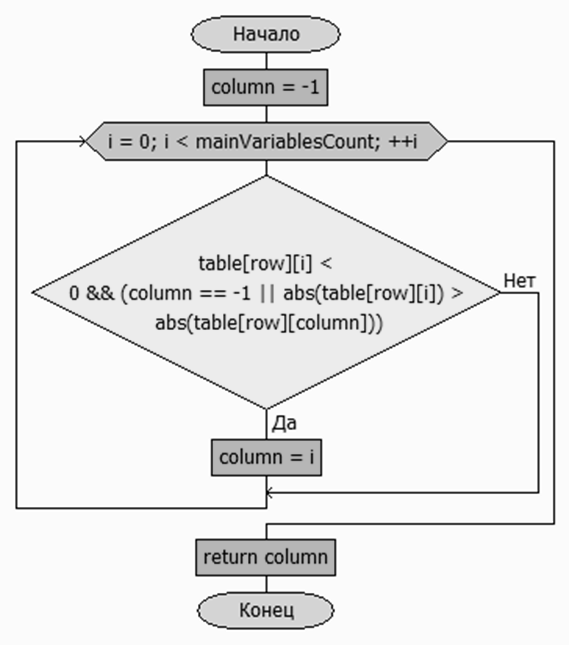
Автоматически созданное описание

Функция getNegativeBRow

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Функция getNegativeBColumn



Функция removeNegativeB

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Функция gauss

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Функция calculateF

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Функция getRelations

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Функция getSolve

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Функция solve

Изображение выглядит как зарисовка, диаграмма, рисунок, текст

Автоматически созданное описание

Изображение выглядит как зарисовка, диаграмма, рисунок, Технический чертеж

Автоматически созданное описание

Функция printTable

Изображение выглядит как текст, Параллельный, диаграмма, зарисовка

Автоматически созданное описание

Функция printCoef

Изображение выглядит как диаграмма, текст, План, Технический чертеж

Автоматически созданное описание

Функция PrintTask

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Автоматически созданное описание

Функция makeDual

Изображение выглядит как текст, снимок экрана, Шрифт, Параллельный

Автоматически созданное описание

Функция main

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

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 | x1 | x2 | x3 | x4 | x5 | x6 | x7 |
| x5 | 75/4 | -69/8 | 0 | -53/4 | 0 | 1 | -1/8 | 19/8 |
| x4 | 37/4 | -19/8 | 0 | -11/4 | 1 | 0 | 1/8 | 5/8 |
| x2 | 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.