Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Курсовая работа по дисциплине  
 Алгоритмы и вычислительные методы оптимизации**

**Симплекс-метод**

Вариант 1

Выполнил:

студент гр. ИП-014 Обухов А.И.

ФИО студента

«27» мая 2023 г.

Проверил

Новожилов Д. И.

ФИО преподавателя

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2023 г.

**Оглавление**

[1. Постановка задачи 3](#_Toc135151225)

[2. Алгоритм метода 4](#_Toc135151226)

[3. Описание основных функций программы 5](#_Toc135151227)

[4. Результат выполнения 6](#_Toc135151228)

[5. Список литературы 7](#_Toc135151229)

[6. Листинг программы 8](#_Toc135151230)

# Постановка задачи

Написать программу, решающую задачу линейного программирования в канонической симплекс-методом форме одним из перечисленных способов:

* симплекс-методом, используя в качестве начальной угловой точки опорное решение, найденное методом Жордана-Гаусса (1);
* методом искусственного базиса (2);
* двойственным симплекс-методом (3).

Требования на оценку хорошо:

− программа работает с классом простых дробей;

− программа находит решение ЗЛП методом по варианту;

− в качестве входных данных подаются матрицы, удовлетворяющие требованиям метода;

− программа должна обрабатывать возможное отсутствие решений;

− если решений бесконечно много, то программа находит конечное число решений (больше одного).

Номер варианта V выбирается по формуле V = (N mod 3) + 1, где N – номер в таблице. V = (16 mod 3) + 1 = 2

# Алгоритм метода

1. Составление первого опорного плана. Переход к канонической форме задачи линейного программирования путем введения неотрицательных дополнительных балансовых переменных.
2. Проверка плана на оптимальность. Если найдется хотя бы один коэффициент индексной строки меньше нуля, то план не оптимальный, и его необходимо улучшить.
3. Определение ведущих столбца и строки. Из отрицательных коэффициентов индексной строки выбирается наибольший по абсолютной величине. Затем элементы столбца свободных членов симплексной таблицы делит на элементы того же знака ведущего столбца.
4. Построение нового опорного плана. Переход к новому плану осуществляется в результате пересчета симплексной таблицы методом Жордана—Гаусса.

# Описание основных функций программы

Класс простых дробей Fraction, организует работу с дробями в данной программе (все математические операции, вплоть до сокращения.)

Функция print\_simplex\_table(A, B, Z, basis\_variables) совершает вывод симплекс таблица

Функция print\_solution(A, B, basis\_variables, n) выводит решения

Функция jordan\_gauss(A, B) реализует метод Жордано-Гаусса

Функция find\_basic\_solution(A, B, basis\_variables, n) осуществляет поиск опорного решения

Функция find\_simple\_division(A, B, s) нужна для вычисления СО

Функция simplex\_table(A, B, Z, basis\_variables) выполняет пересчет симплекс таблицы.

В функции main происходит вызов всех вспомогательных функций и пересчёт результатов промежуточных симплекс таблиц Жордановскими преобразованиями.

# Результат выполнения

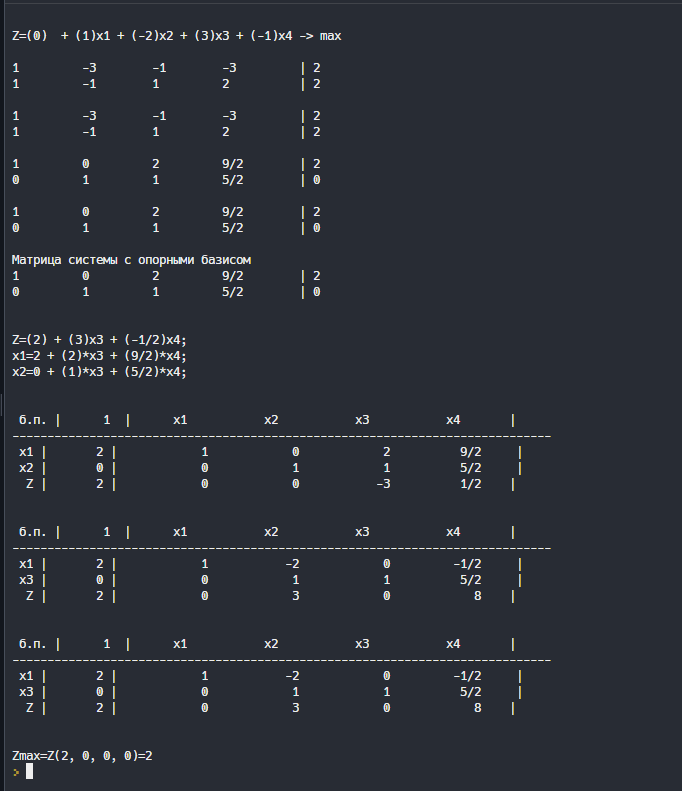


Рисунок 1 — ЗЛП с единственным решением

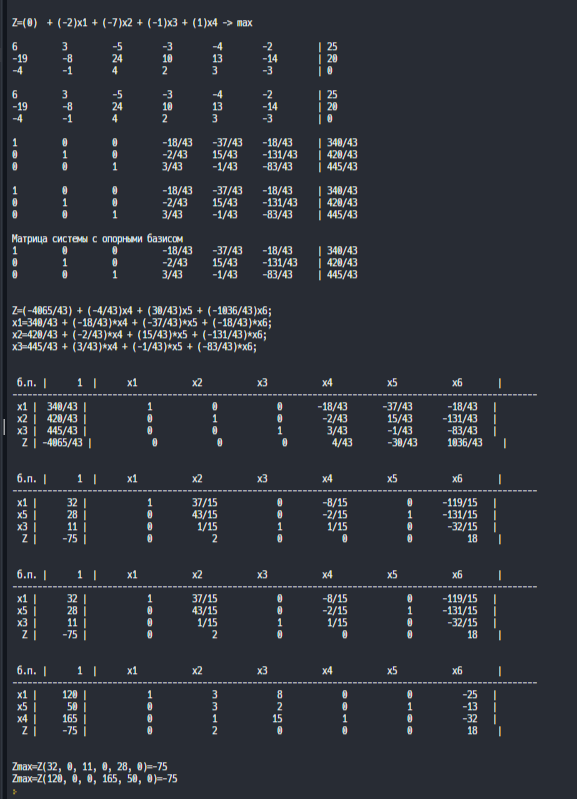


Рисунок 2 — ЗЛП имеет несколько решений

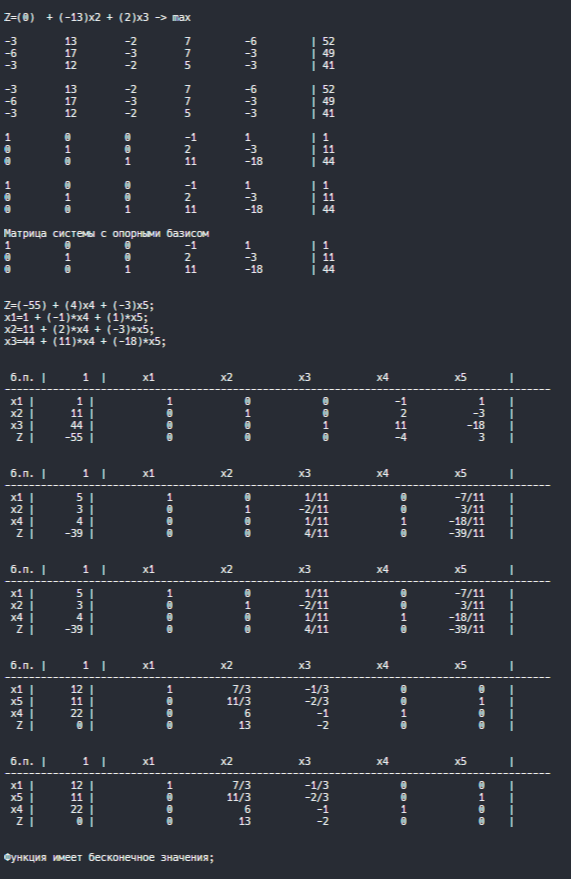


Рисунок 3 — ЗЛП не имеет решений

# Список литературы

1. *Ольга Андреева, О.И. Ремизова* Основы алгоритмизации и программирования на языке Python. СПБ.:Питер, 2022. – 150с.
2. *Галкина М.Ю.* Алгоритмы и вычислительные методы оптимизации: учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск: СибГУТИ, 2021. 136 с.

# Листинг программы

def simplex\_table(A, B, Z, basis\_variables):

pot = []

basis\_variables\_clone = []

n = len(A[0])

k = 0

s = 0

X = [[]]

flagneg = False

zerocount = 0

while True:

print\_simplex\_table(A, B, Z, basis\_variables)

flagneg = False

zerocount = 0

pot.clear()

for j in range(n):

if (check\_in(j, basis\_variables)):

continue

if -Z[j].\_numerator < 0:

if flagneg:

if Z[pot[0]] < Z[j]:

pot[0] = j

else:

pot.append(j)

flagneg = True

elif Z[j].\_numerator == 0:

zerocount += 1

if flagneg:

s = pot[0]

k = find\_simple\_division(A, B, s)

if (k == -1):

return

calculate\_simplex\_table(A, B, Z, k, s)

basis\_variables[k] = s

print\_simplex\_table(A, B, Z, basis\_variables)

else:

X = [Fraction(0) for i in range(zerocount + 1)]

X[0] = create\_solution(B, basis\_variables, 0, basis\_variables[0], n)

for i in range(len(basis\_variables)):

basis\_variables\_clone.append(basis\_variables[i])

count = 1

for j in range(n):

if not check\_in(j, basis\_variables\_clone) and Z[j] == Fraction(0):

k = find\_simple\_division(A, B, j)

calculate\_simplex\_table(A, B, Z, k, j)

basis\_variables[k] = j

print\_simplex\_table(A, B, Z, basis\_variables)

X[count] = create\_solution(B, basis\_variables, k, j, n)

count += 1

for j in range(zerocount + 1):

print\_simplex\_solution(X[j], Z[n])

break

def main():

sys.stdin = open('input.txt', 'r')

n, m = map(int, input().split())

basis\_variables = []

A = []

B = []

Z = []

for i in range(m):

temp = input().split()

tempv = []

for j in range(n):

tempv.append(Fraction.from\_str(temp[j]))

A.append(tempv)

B.append(Fraction.from\_str(temp[n]))

tempv = input().split()

for j in range(n):

Z.append(Fraction.from\_str(tempv[j]))

Z.append(Fraction(0))

print("\nZ=({})".format(Z[n]), end=' ')

for i in range(n):

if Z[i] != Fraction(0):

print(" + ({})x{}".format(Z[i], i + 1), end="")

print(" -> max \n")

print\_matrix(A, B)

r = find\_basic\_solution(A, B, basis\_variables, n)

if r == -1:

return 0

for i in range(r):

for j in range(n):

if not check\_in(j, basis\_variables):

Z[j] = Z[j] - Z[basis\_variables[i]] \* A[i][j]

Z[n] = Z[n] + Z[basis\_variables[i]] \* B[i]

Z[basis\_variables[i]] = Fraction(0)

print("\nZ=({})".format(Z[n]), end='')

for i in range(n):

if not check\_in(i, basis\_variables):

print(" + ({})x{}".format(Z[i], i + 1), end='')

print(';')

print\_solution(A, B, basis\_variables, n)

print()

simplex\_table(A, B, Z, basis\_variables)

if \_\_name\_\_ == '\_\_main\_\_':

main()