Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

Лабораторная работа № 2

**«Нахождение всех базисных решений системы линейных уравнений»**

по дисциплине «Алгоритмы и вычислительные методы оптимизации»

Бригада № 1

Выполнил: студент 3 курса группы ИП-811 Мироненко К. А

Проверил: ассистент кафедры ПМиК Новожилов Д.И.

**Оглавление**

[1. Постановка задачи 3](#_Toc51530076)

[2. Примеры работы программы 4](#_Toc51530077)

[*Приложение* Листинг 8](#_Toc51530078)

# Постановка задачи

Написать программу, находящую все базисные решения системы линейных уравнений методом Жордана-Гаусса.

Программа должна выводить промежуточные матрицы после каждого шага исключений и все найденные базисные решения.

Программа должна работать для различных тестов: система имеет единственное решение, система имеет бесконечно много решений, система не имеет решения.

Должна иметься возможность быстро ввести входные данные для различного количества переменных и уравнений. Начальную работу программу необходимо продемонстрировать на предложенной ниже системе (система выбирается по номеру бригады).

Для получения максимальной оценки необходимо, чтобы все вычисления выполнялись в простых дробях. Для этого использовать класс простых дробей, реализованный в лабораторной 1.

# Примеры работы программы

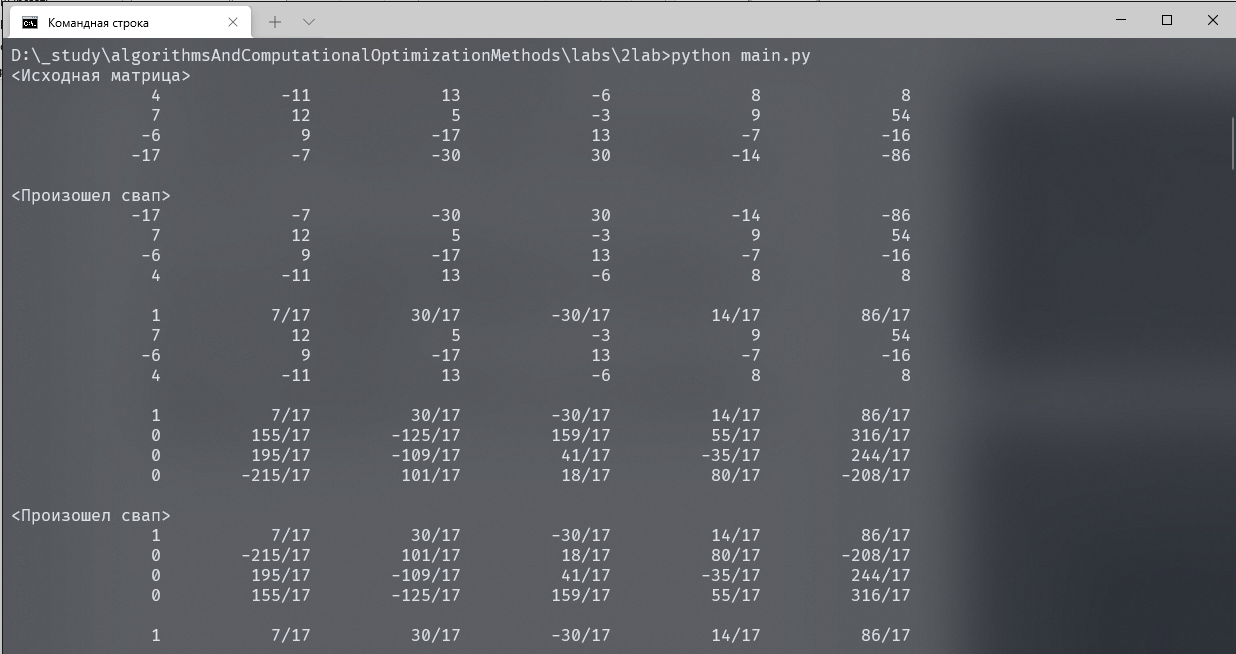


Рис.1.1 Решение системы по варианту

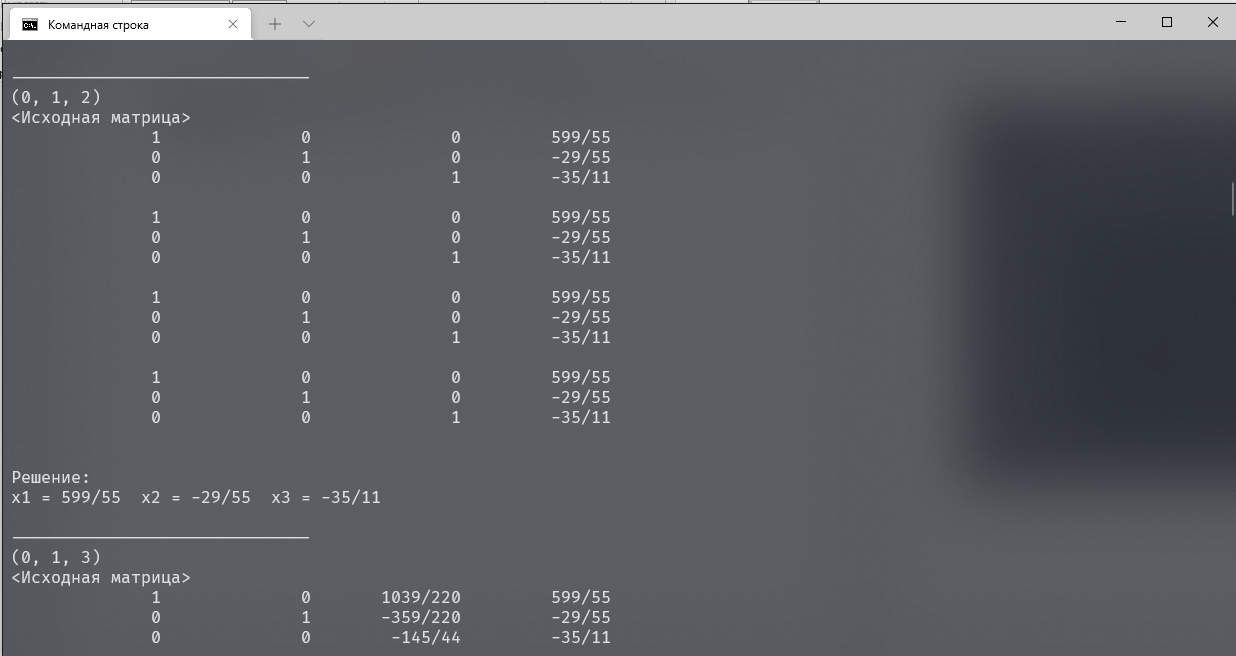


Рис.1.2 Решение системы по варианту

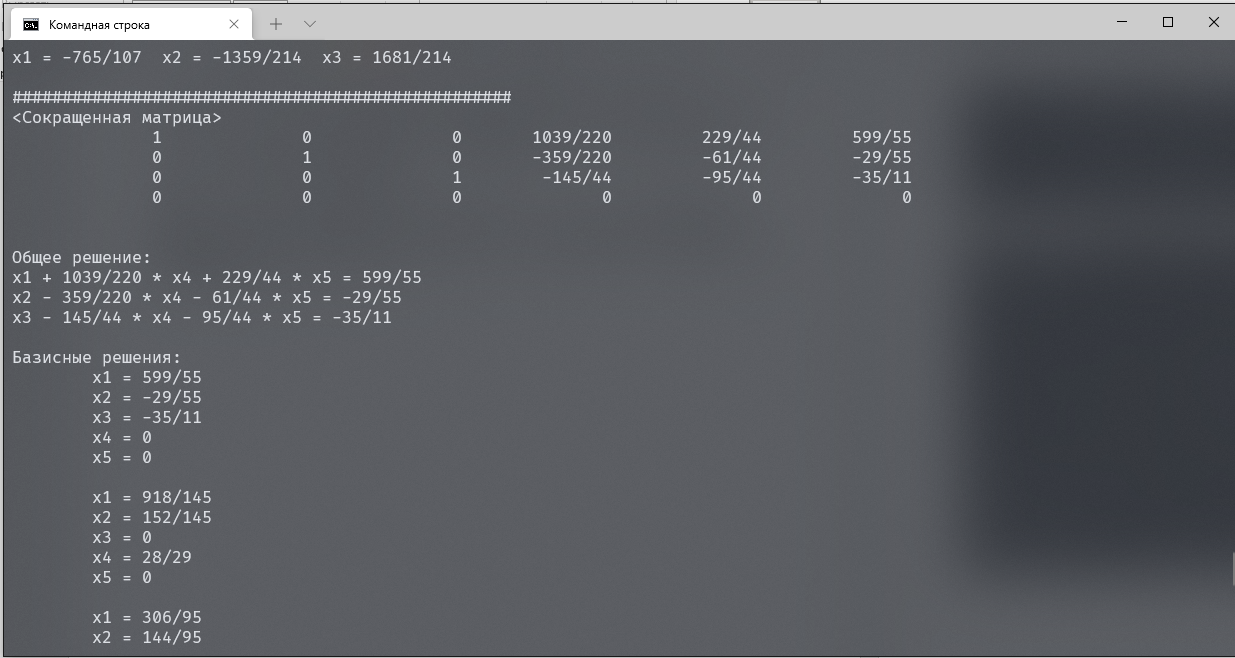


Рис.1.3 Решение системы по варианту

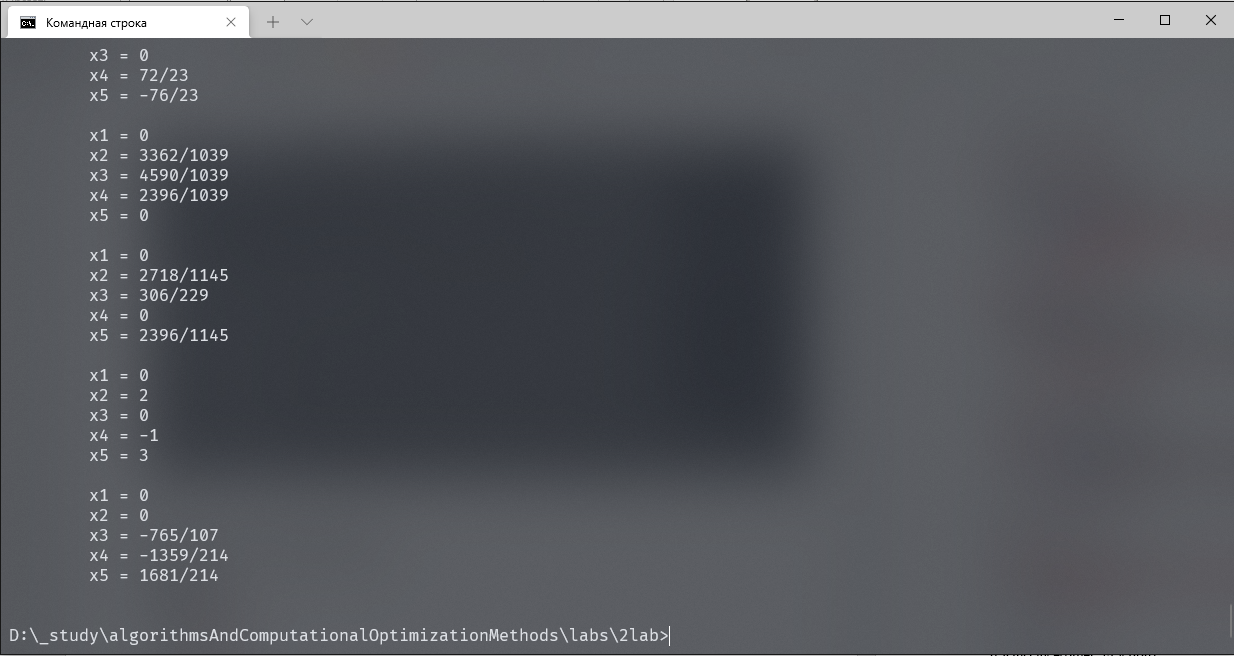


Рис.1.4 Решение системы по варианту

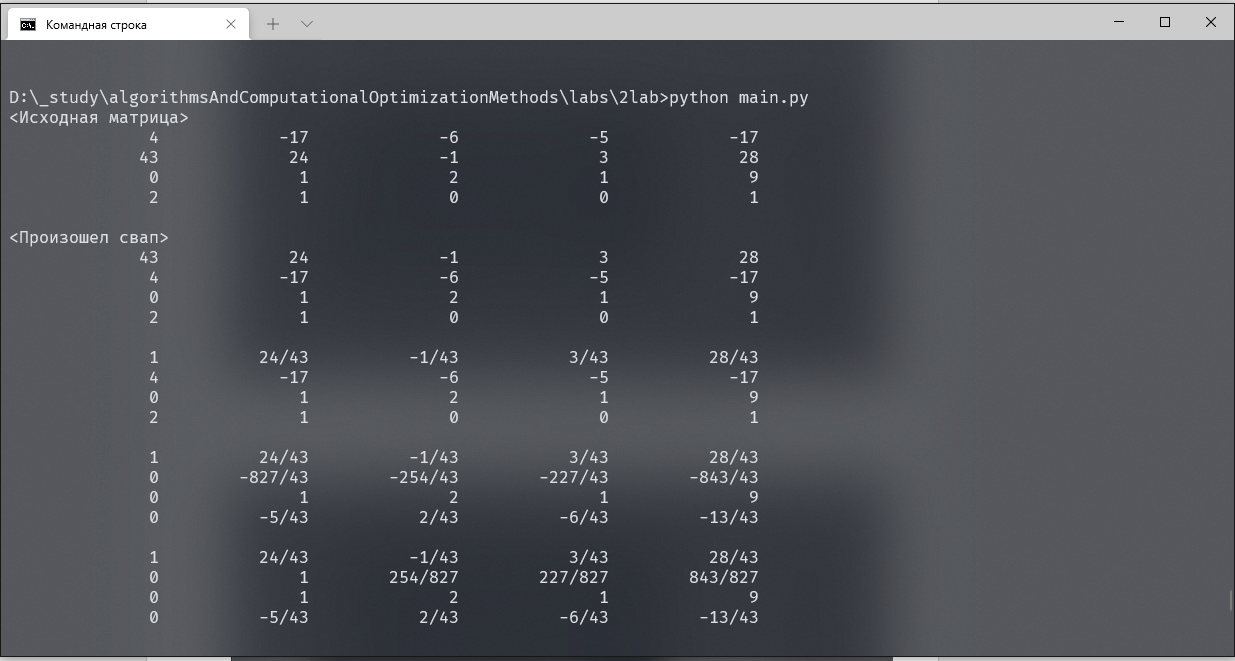


Рис.2.1 Пример из практики с единственным решением

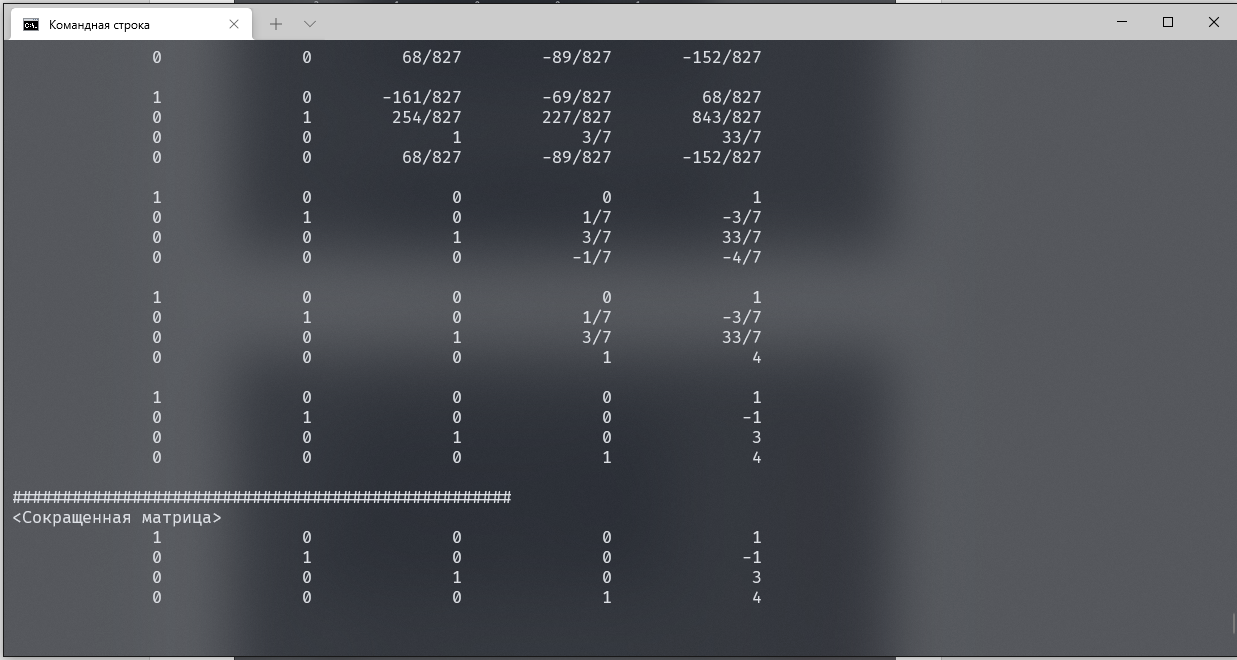


Рис.2.2 Пример из практики с единственным решением

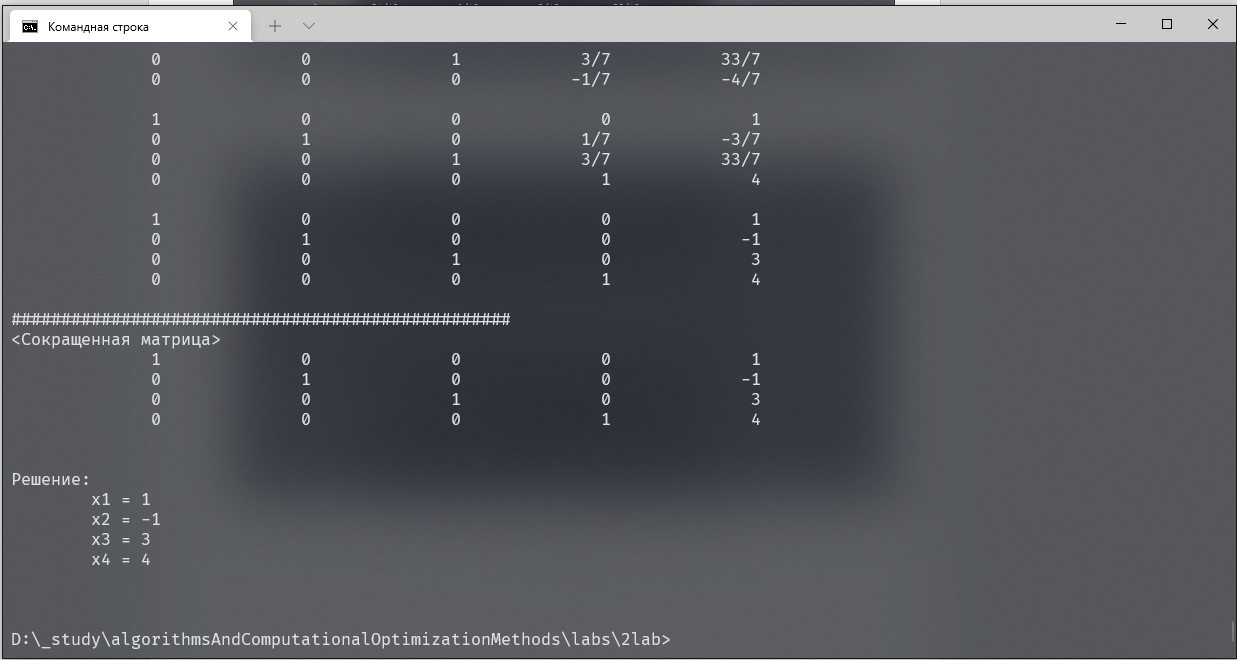


Рис.2.3 Пример из практики с единственным решением

# *Приложение* Листинг

import sys

import math

import itertools

class Fraction:

"""

Класс, реализующий дроби

"""

\_\_slots\_\_ = ('\_numerator', '\_denominator')

def \_\_init\_\_(self, numerator=0, denominator=1):

if type(numerator) is not int or type(denominator) is not int:

raise TypeError(

'Fraction(%s, %s) - the numerator and denominator values must be integers' % (numerator, denominator))

if denominator == 0:

raise ZeroDivisionError('Fraction(%s, 0)' % numerator)

g = math.gcd(numerator, denominator)

if denominator < 0:

g = -g

numerator //= g

denominator //= g

self.\_numerator = numerator

self.\_denominator = denominator

def \_\_add\_\_(self, other):

"""Сумма 2-х дробей"""

if isinstance(other, Fraction):

return Fraction(self.\_numerator \* other.\_denominator + other.\_numerator \* self.\_denominator,

self.\_denominator \* other.\_denominator)

return NotImplemented

def \_\_sub\_\_(self, other):

"""Разность 2-х дробей"""

if isinstance(other, Fraction):

return Fraction(self.\_numerator \* other.\_denominator - other.\_numerator \* self.\_denominator,

self.\_denominator \* other.\_denominator)

return NotImplemented

def \_\_mul\_\_(self, other):

"""Произведение 2-х дробей"""

if isinstance(other, Fraction):

return Fraction(self.\_numerator \* other.\_numerator,

self.\_denominator \* other.\_denominator)

return NotImplemented

def \_\_truediv\_\_(self, other):

"""Частное 2-х дробей"""

if isinstance(other, Fraction):

return Fraction(self.\_numerator \* other.\_denominator,

self.\_denominator \* other.\_numerator)

return NotImplemented

def \_\_lt\_\_(self, other):

"""x < y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator < other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_le\_\_(self, other):

"""x <= y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator <= other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_eq\_\_(self, other):

"""x == y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator == other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_ne\_\_(self, other):

"""x != y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator != other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_gt\_\_(self, other):

"""x > y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator > other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_ge\_\_(self, other):

"""x >= y"""

if isinstance(other, Fraction):

return self.\_numerator \* other.\_denominator >= other.\_numerator \* self.\_denominator

return NotImplemented

def \_\_repr\_\_(self):

if self.\_denominator == 1:

return 'Fraction(%s)' % self.\_numerator

else:

return 'Fraction(%s, %s)' % (self.\_numerator, self.\_denominator)

def \_\_str\_\_(self):

if self.\_denominator == 1:

return str(self.\_numerator)

else:

return '%s/%s' % (self.\_numerator, self.\_denominator)

def abs(self):

return Fraction(abs(self.\_numerator), abs(self.\_denominator))

def print\_matrix(matrix):

for i in matrix:

for j in i:

print('%15s' % j, end='')

print()

print()

def jordan\_gauss\_method(matrix, flag=True):

print("<Исходная матрица>")

print\_matrix(matrix)

# По столбцам

for c in range(len(matrix)):

index = c

# По элементам столбца

for i in range(c + 1, len(matrix)):

if matrix[index][c].abs() < matrix[i][c].abs():

index = i

if index != c:

matrix[index], matrix[c] = matrix[c], matrix[index]

print("<Произошел свап>")

print\_matrix(matrix)

if matrix[c][c] == Fraction(0):

continue

if matrix[c][c] != Fraction(1):

# Сокращение строки

matrix[c] = [i / matrix[c][c] for i in matrix[c]]

print\_matrix(matrix)

# По всем строкам для их "зануления"

for i in range(len(matrix)):

if matrix[i][c] == Fraction(0) or i == c:

continue

coeff = matrix[i][c] \* Fraction(-1)

# По элементам строк, начиная с c-ого

for j in range(c, len(matrix[0])):

matrix[i][j] = matrix[i][j] + matrix[c][j] \* coeff

print\_matrix(matrix)

count\_no\_null\_str = 0

for i in matrix:

null\_sum\_flag = True

for j in i[:-1]:

if j != Fraction(0):

null\_sum\_flag = False

break

# Если элементы строки нулевые и значение не нулевое

if null\_sum\_flag and i[-1] != Fraction(0):

count\_no\_null\_str = 0

break

# Если строка нулевая

elif null\_sum\_flag and i[-1] == Fraction(0):

continue

count\_no\_null\_str += 1

if not count\_no\_null\_str:

return None

elif count\_no\_null\_str == len(matrix) and count\_no\_null\_str == len(matrix[0]) - 1:

return [[matrix[i][-1] for i in range(len(matrix))]]

else:

res = [[], []]

# Общее решение

for i in matrix:

tmp\_sum = Fraction(0)

for j in i:

tmp\_sum += j.abs()

if tmp\_sum != Fraction(0):

res[0].append(i)

matrix = res[0]

# Базисное решение

if flag:

for i in itertools.combinations([i for i in range(len(matrix[0])-1)], count\_no\_null\_str):

print("-" \* 30)

print(i)

tmp\_res = [0 for i in range(len(matrix[0]) - 1)]

tmp\_matrix = []

# Получение необходимых столбов, как строк

for j in i:

tmp\_matrix.append([x[j] for x in matrix])

tmp\_matrix.append([x[-1] for x in matrix])

# Транспонирование

tmp\_matrix = [list(j) for j in zip(\*tmp\_matrix)]

res\_r = jordan\_gauss\_method(tmp\_matrix, False)

if not res\_r:

print("\nНет решения")

elif len(res\_r) == 1:

print("\nРешение:")

ans = res\_r[0]

for j in range(len(ans)):

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

t = 0

for j in i:

tmp\_res[j] = ans[t]

t += 1

print("\n")

res[1].append(tmp\_res)

elif len(res\_r) == 2:

print("\nСЛАУ имеет множество решений")

return res

def main():

matrix = []

f = open('input.txt', 'r')

# TODO: возможность ввода дробей

for line in f:

a = list(map(int, line.strip().split(' ')))

matrix.append(list(map(Fraction, a)))

res = jordan\_gauss\_method(matrix)

print("#" \* 50)

print("<Сокращенная матрица>")

print\_matrix(matrix)

if not res:

print("\nНет решения")

elif len(res) == 1:

print("\nРешение:")

ans = res[0]

for i in range(len(ans)):

print("\tx{} = {}".format(i + 1, ans[i]))

print("\n")

elif len(res) == 2:

o\_ans = res[0]

ans = res[1]

print("\nОбщее решение:")

for i in o\_ans:

tmp\_str = ""

for j in range(len(i) - 1):

tmp = i[j]

if tmp != Fraction(0):

if tmp.abs() != Fraction(1):

if tmp < Fraction(0):

tmp\_str += " - "

else:

tmp\_str += " + "

tmp\_str += str(tmp.abs()) + " \* "

tmp\_str += "x" + str(j + 1)

tmp\_str += " = " + str(i[-1])

print(tmp\_str)

print("\nБазисные решения:")

for i in ans:

for j in range(len(i)):

print("\tx{} = {}".format(j + 1, i[j]))

print()

else:

print("Вышла какая-то ошибочка :с")

return

def test():

a = Fraction(-1, 5)

b = Fraction(4, 10)

print(a + b)

print(a < b)

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

main()

# test()