Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

Лабораторная работа № 1

**«Решение систем линейных уравнений методом Жордана-Гаусса»**

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 5](#_Toc51530078)

# Постановка задачи

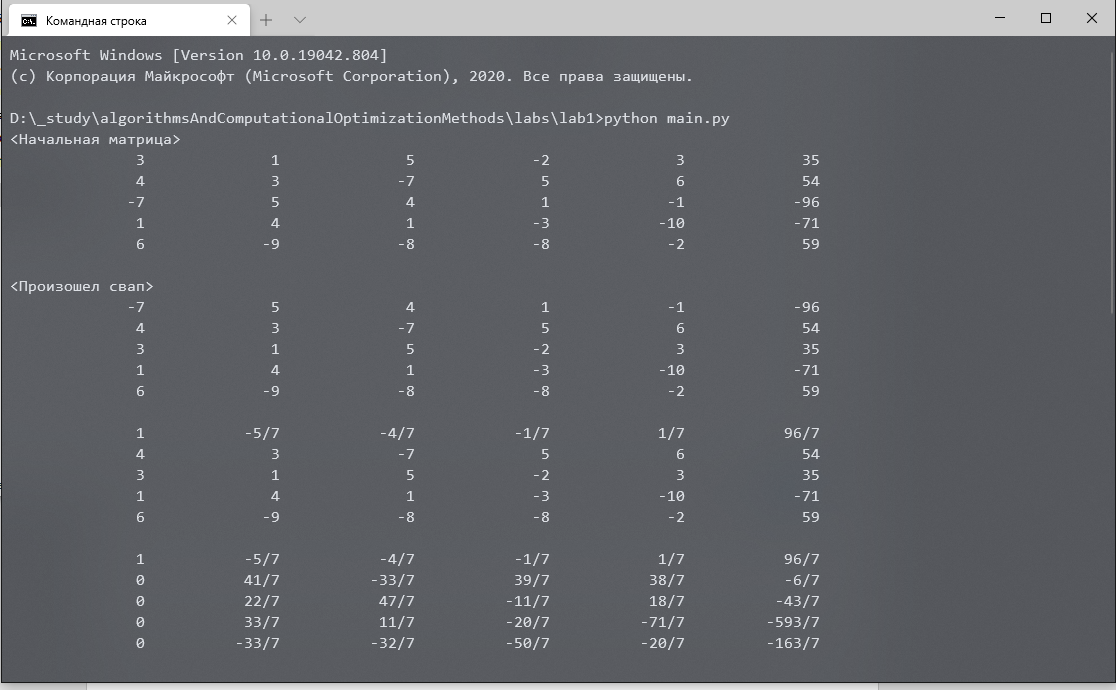
Написать программу, находящую решение системы линейных уравнений методом Жордана-Гаусса.

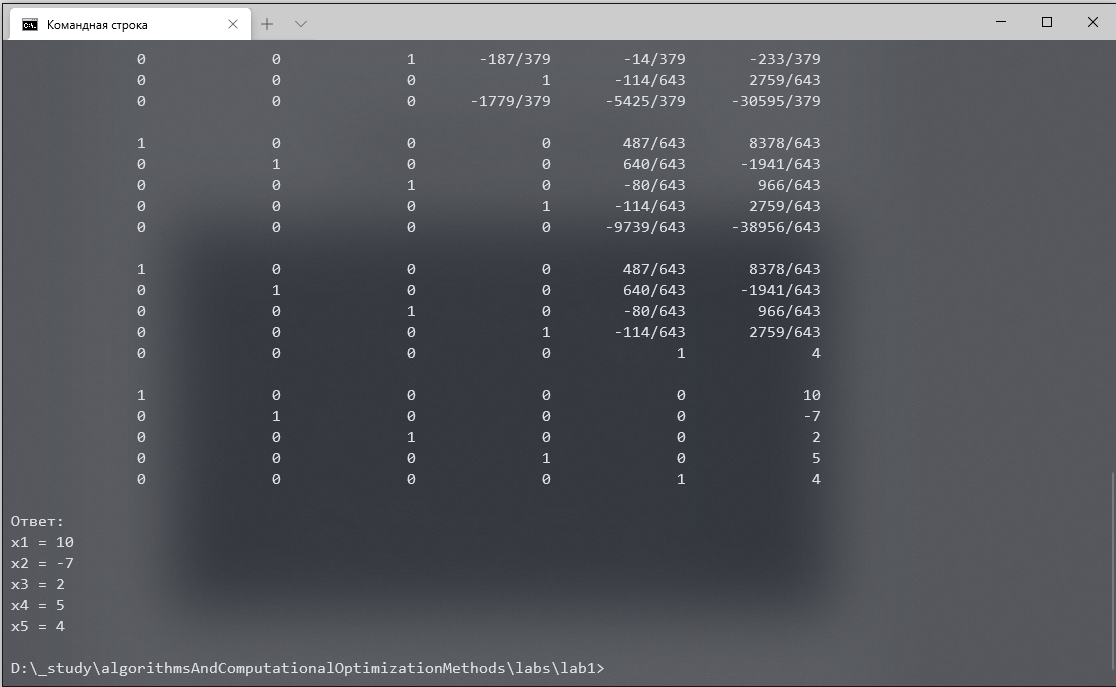
Программа должна выводить промежуточные матрицы после каждого шага исключений и решение системы. Программа должна работать для различных тестов: система имеет единственное решение, система имеет бесконечно много решений, система не имеет решения.

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 реализовать класс простых дробей. Реализованный класс можно будет использовать в лабораторной №2 и курсовой работе.

# Примеры работы программы





# *Приложение* Листинг

import sys

import math

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 get\_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):

print("<Начальная матрица>")

print\_matrix(matrix)

for c in range(len(matrix)):

# По столбцам

index = c

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

# По строкам

if matrix[index][c].get\_abs() < matrix[i][c].get\_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):

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

print\_matrix(matrix)

for i in range(0, len(matrix)):

# По строкам, но по всем

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

continue

if i == c:

continue

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

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

# По элементам строк

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

# print(\*matrix, sep="\n", end="\n\n")

print\_matrix(matrix)

no\_null\_str = 0

for i in matrix:

flag = True

for j in i[:-1]:

if j != Fraction(0):

flag = False

break

if flag and i[-1] != Fraction(0):

no\_null\_str = 0

break

elif flag and i[-1] == Fraction(0):

continue

no\_null\_str += 1

print("Ответ:")

if not no\_null\_str:

print("нет решения")

elif no\_null\_str == len(matrix) and no\_null\_str == len(matrix[0]) - 1:

tmp = list()

for i in range(len(matrix)):

tmp.append("x" + str(i + 1) + " = " + str(matrix[i][-1]))

print(\*tmp, sep='\n')

else:

tmp = list()

for i in range(len(matrix)):

str\_tmp = ""

if matrix[i][i] == Fraction(1):

str\_tmp += "x" + str(i + 1) + " = " + str(matrix[i][-1])

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

if j == i:

continue

if matrix[i][j] == Fraction(0):

continue

str\_tmp += " + "

str\_tmp += "(" + str(matrix[i][j] \* Fraction(-1)) + ")" + "x" + str(j + 1)

else:

flag = True

for j in matrix[i][:-1]:

if j != Fraction(0):

flag = False

break

if flag and matrix[i][-1] == Fraction(0):

continue

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

if matrix[i][j] == Fraction(0):

continue

str\_tmp += " + "

str\_tmp += "(" + str(matrix[i][j] \* Fraction(-1)) + ")" + "x" + str(j + 1)

str\_tmp += " = " + str(matrix[i][-1])

if str\_tmp != "":

tmp.append(str\_tmp)

print(\*tmp, sep='\n')

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)))

jordan\_gauss\_method(matrix)

return

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

main()