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.

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи

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

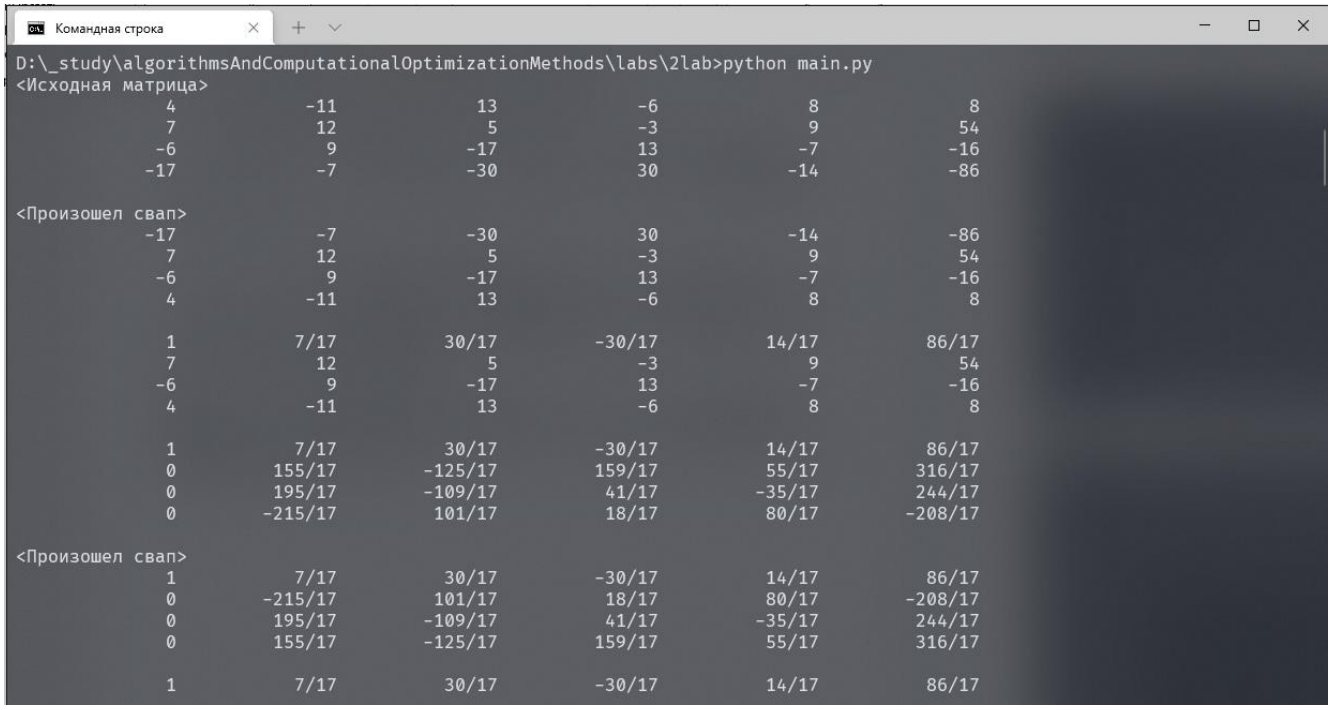
Программа должна выводить промежуточные матрицы после каждого шага исключений и все найденные базисные решения.

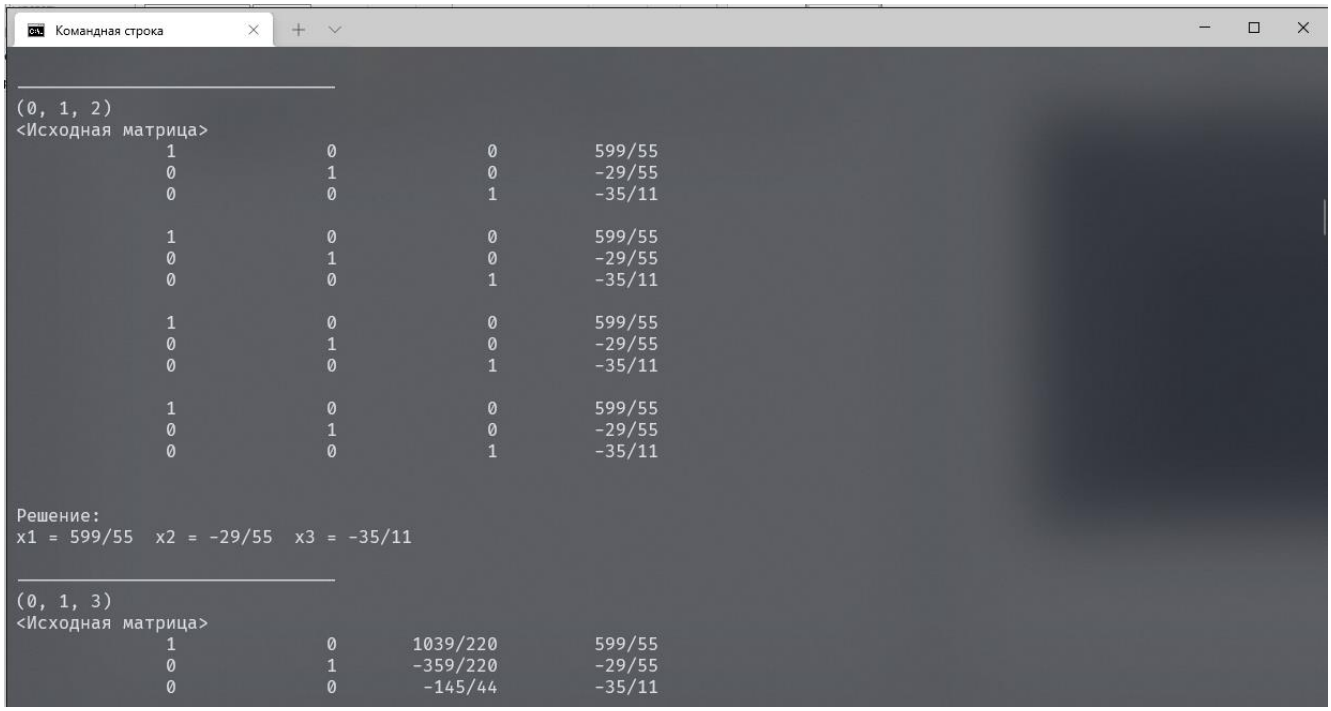
Программа должна работать для различных тестов: система имеет единственное решение, система имеет бесконечно много решений, система не имеет решения.

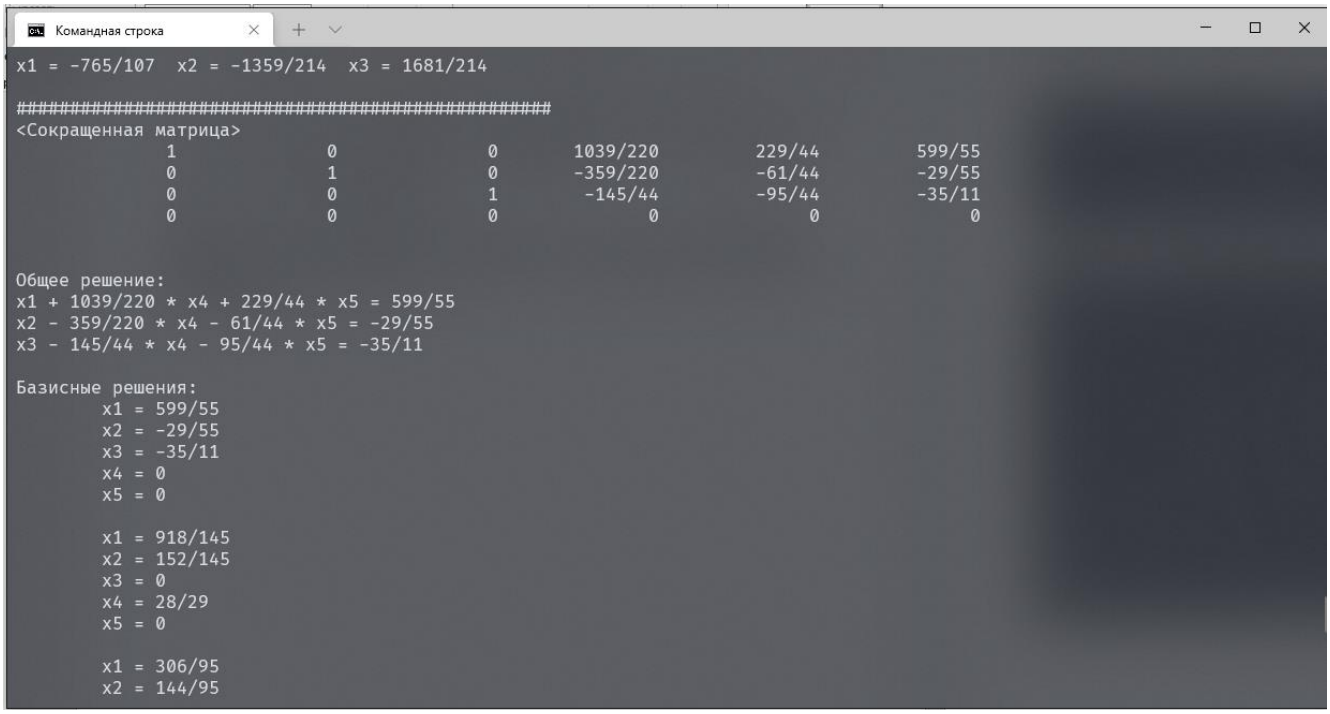
Должна иметься возможность быстро ввести входные данные для различного количества переменных и уравнений. Начальную работу программу необходимо продемонстрировать на предложенной ниже системе (система выбирается по номеру бригады).

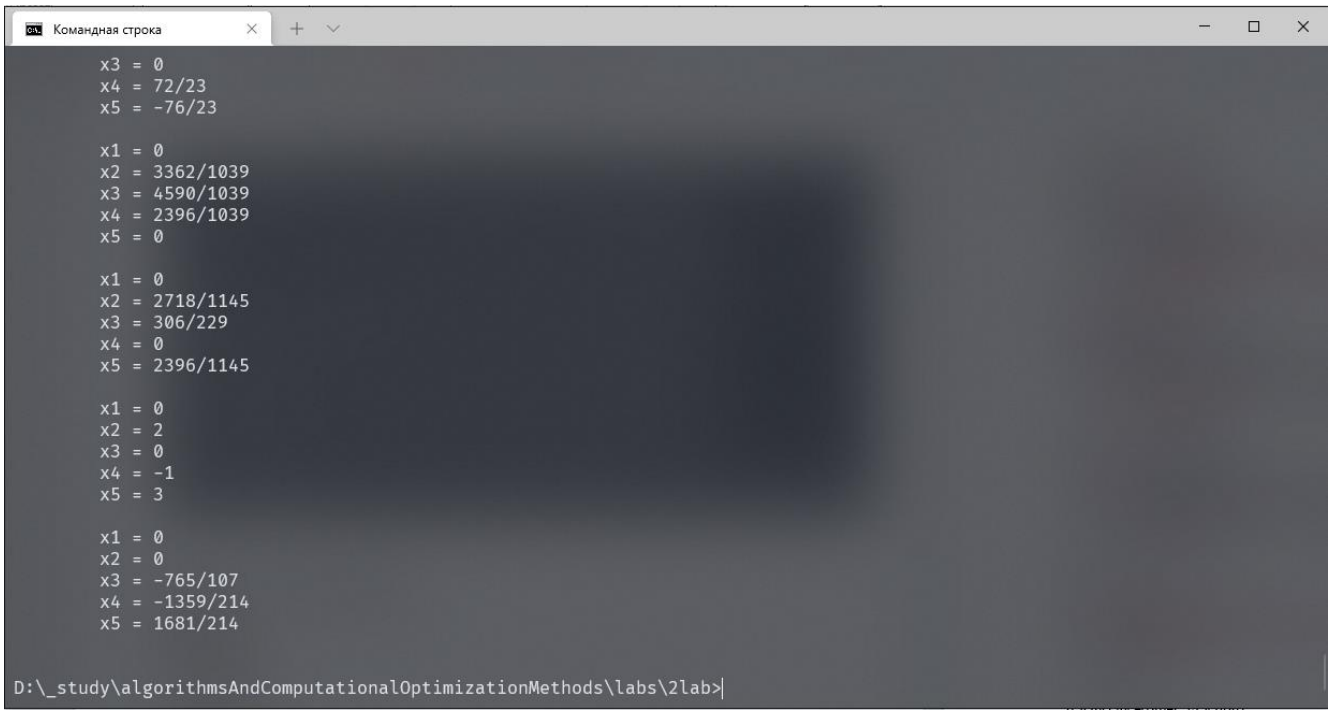
Для получения максимальной оценки необходимо, чтобы все вычисления выполнялись в простых дробях. Для этого использовать класс простых дробей, реализованный в лабораторной 1.

2. Примеры работы программы






Командная строка
x1 = -765/107 x2 = -1359/214 x3 = 1681/214
##
<Сокращенная матрица>
1 0 0 1039/220 229/44 599/55
0 1 0 -359/220 -61/44 -29/55
0 0 1 -145/44 -95/44 -35/11
0 0 0 0 0 0
Общее решение:
x1 + 1039/220 * x4 + 229/44 * x5 = 599/55
x2 - 359/220 * x4 - 61/44 * x5 = -29/55
x3 - 145/44 * x4 - 95/44 * x5 = -35/11
Базисные решения:
x1 = 599/55
x2 = -29/55
x3 = -35/11
x4 = 0
x5 = 0
x1 = 918/145
x2 = 152/145
x3 = 0
x4 = 28/29
x5 = 0
x1 = 306/95
x2 = 144/95


Командная строка
x3 = 0
x4 = 72/23
x5 = -76/23
x1 = 0
x2 = 3362/1039
x3 = 4590/1039
x4 = 2396/1039
x5 = 0
x1 = 0
x2 = 2718/1145
x3 = 306/229
x4 = 0
x5 = 2396/1145
x1 = 0
x2 = 2
x3 = 0
x4 = -1
x5 = 3
x1 = 0
x2 = 0
x3 = -765/107
x4 = -1359/214
x5 = 1681/214
D:_study\algorithmsAndComputationalOptimizationMethods\labs\2lab>

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 с-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 :c")
    return


def test():
    a = Fraction(-1, 5)
    b = Fraction(4, 10)
    print(a + b)
    print(a < b)


if __name__ == '__main__':
    main()
    # test()
```