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

Лабораторная работа № 2
по дисциплине
«Алгоритмы и вычислительные методы оптимизации»

Бригада 4

Выполнил:

студент гр. ИП- 111 Корнилов А.А.
ФИО студента

«12» февраля 2024 г.

Новосибирск 2024 г.

Содержание

1. Постановка задачи 3

2. Результат работы программы 4

2.1. Предложенная система 4

2.2. Система для тестирования 8

Приложение. Листинг 10

Класс простых дробей 10

Основная программа 12

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 найденные базисные решения системы.

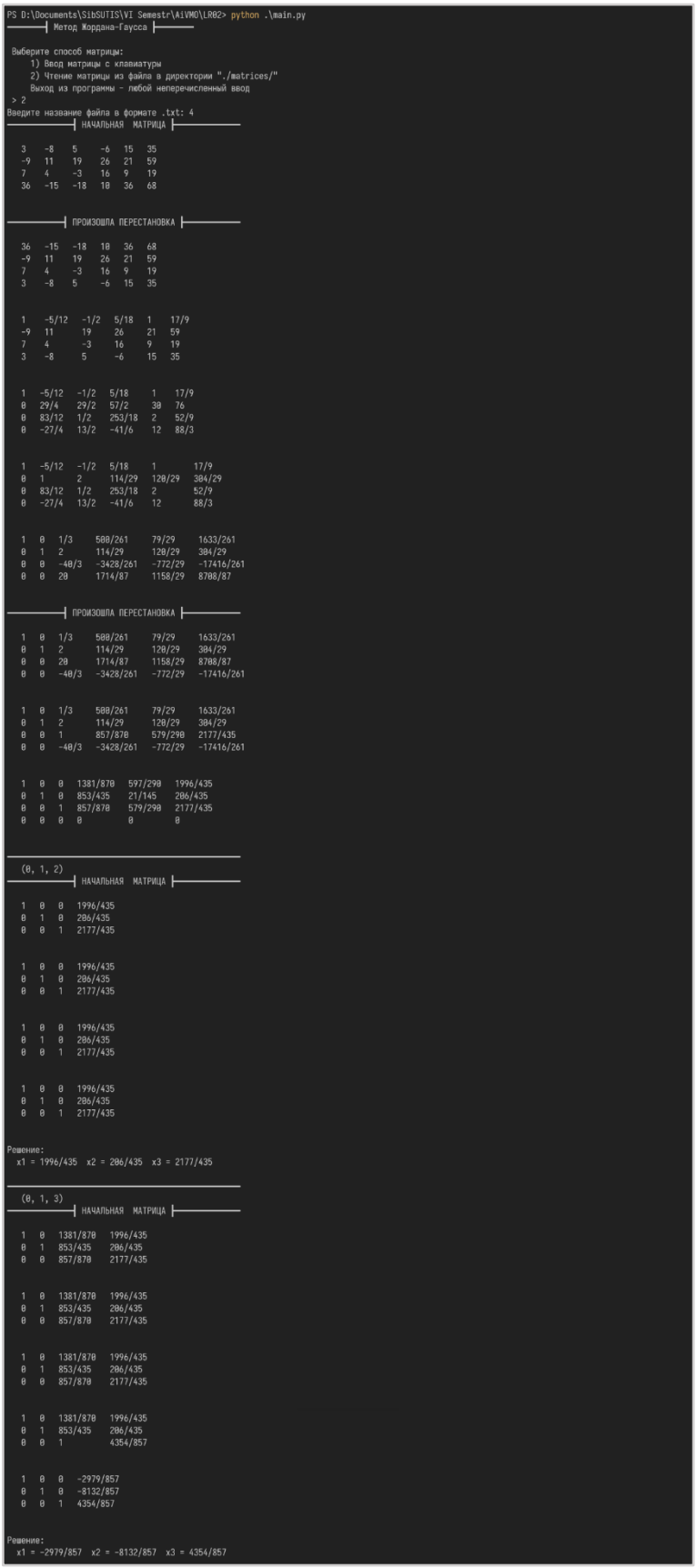
Должна иметься возможность быстро ввести входные данные для различного количества переменных и уравнений. Начальную работу программу необходимо продемонстрировать на предложенной ниже системе:

4. $$\begin{cases} 3x_1 - 8x_2 + 5x_3 - 6x_4 + 15x_5 = 35 \\ -9x_1 + 11x_2 + 19x_3 + 26x_4 + 21x_5 = 59 \\ 7x_1 + 4x_2 - 3x_3 + 16x_4 + 9x_5 = 19 \\ 36x_1 - 15x_2 - 18x_3 + 10x_4 + 36x_5 = 68 \end{cases}$$

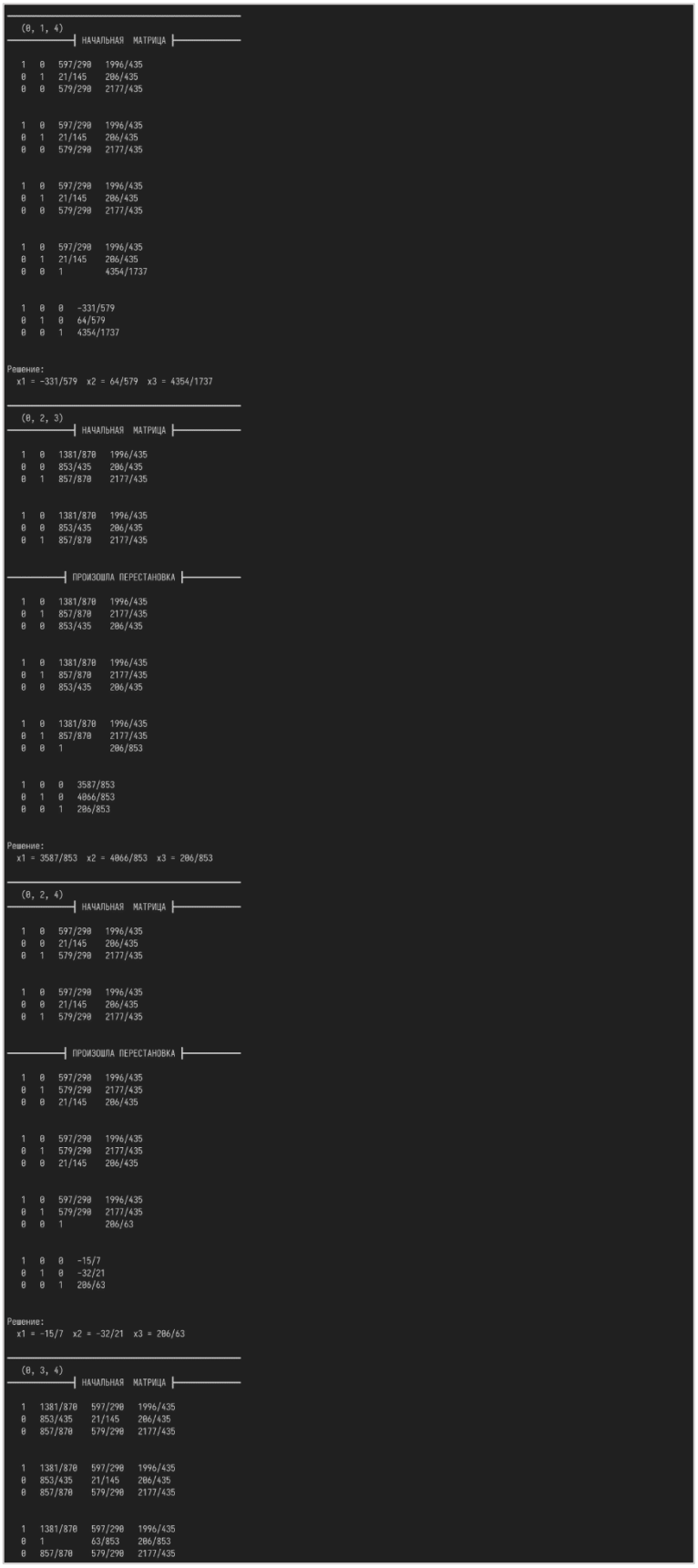
Для получения максимальной оценки необходимо, чтобы все вычисления выполнялись в простых дробях. Для этого использовать класс простых дробей, реализованный в *лабораторной работе №1*.

2. Результат работы программы

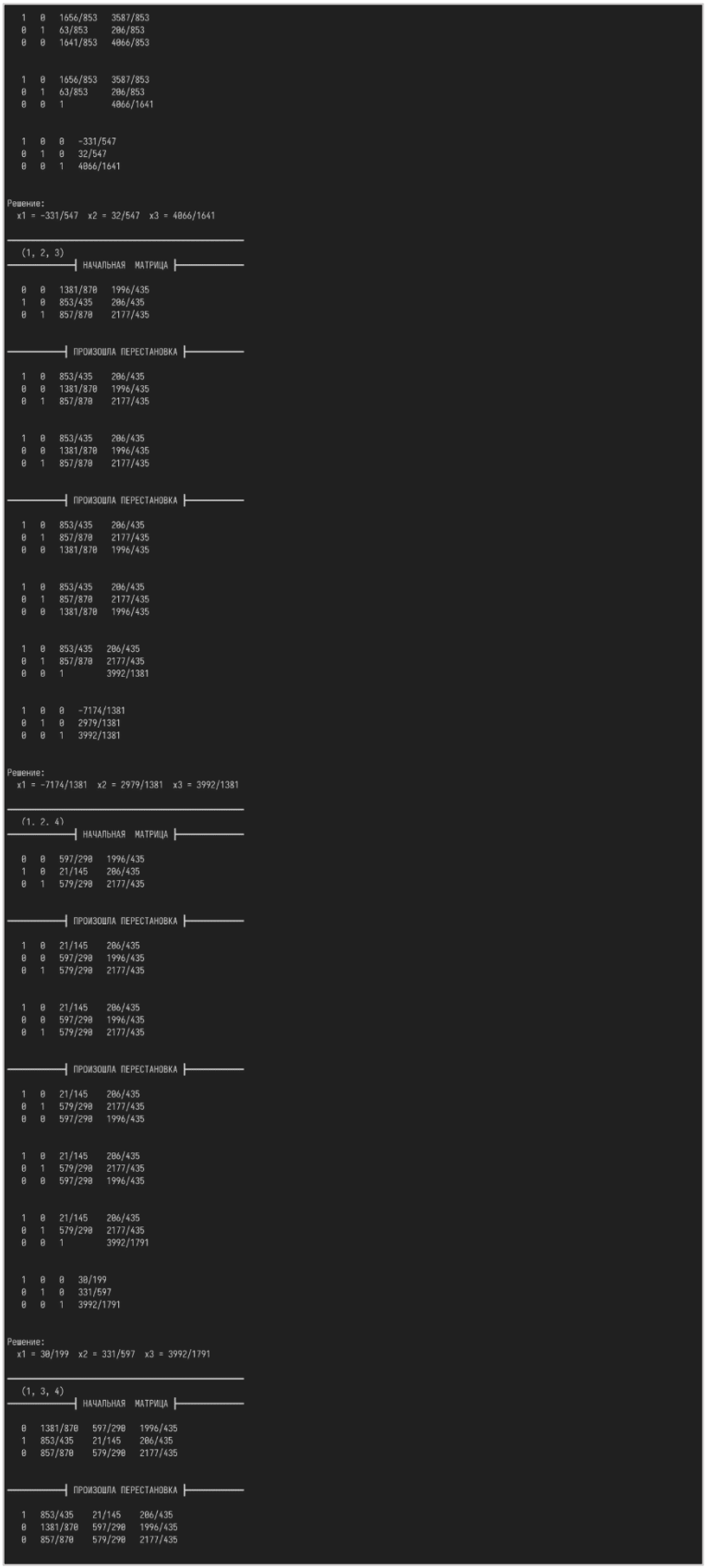
2.1. Предложенная система



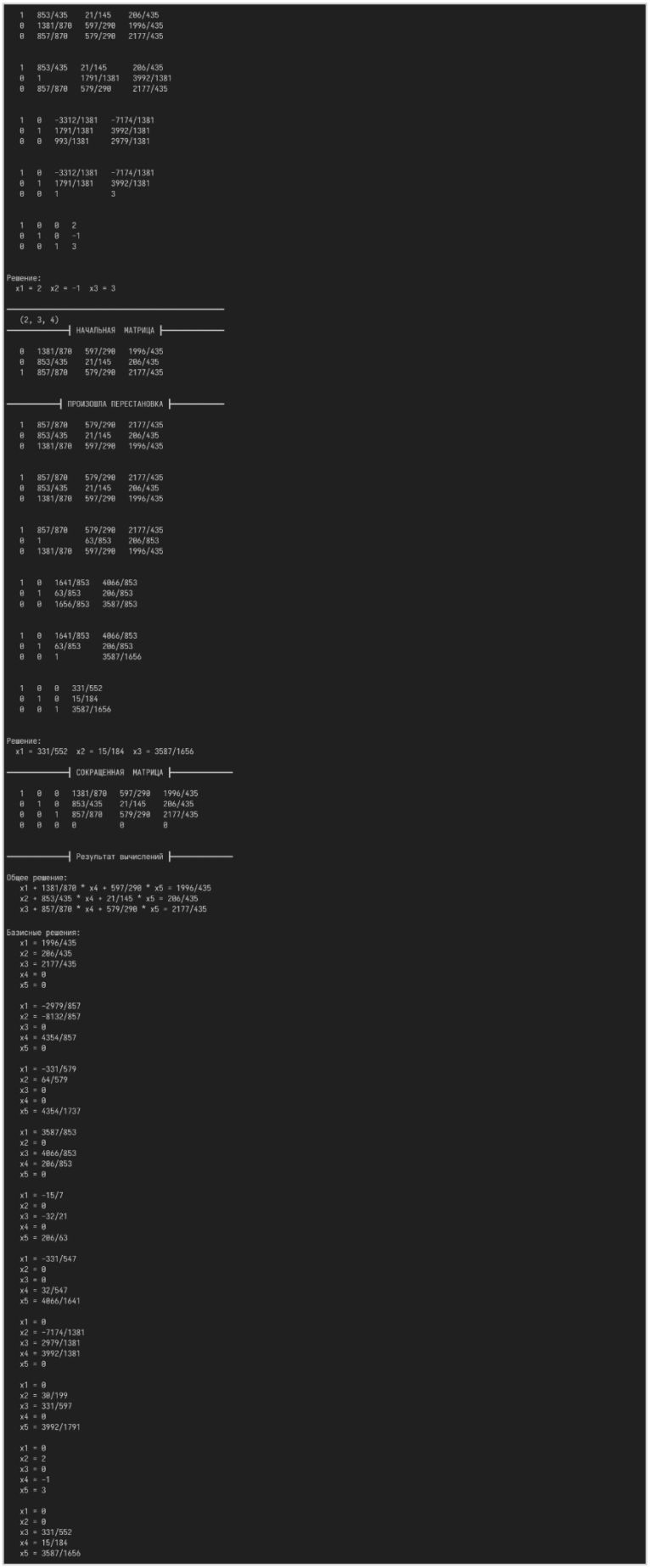
Скриншот 2.1.1. – Вычисление матрицы и начало нахождения базисов



Скриншот 2.1.2. – Нахождение базисов (продолжение)



Скриншот 2.1.3. – Нахождение базисов (продолжение)



Скриншот 2.1.4. – Завершение нахождения базисов и вывод результатов вычисления программы

2.2. Система для тестирования

Скриншот 2.2.1. – Вычисление матрицы и нахождение базисов

Скриншот 2.2.2. – Нахождение базисов (продолжение) и вывод результатов вычисления программы

Приложение. Листинг

Класс простых дробей

fraction.py

```
import math


class Fraction:
   __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) - значения числителя и знаменателя должны быть
целыми числами"
            % (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

   # Сумма двух дробей
   def __add__(self, other):
      if isinstance(other, Fraction):
         return Fraction(
            self._numerator * other._denominator
            + other._numerator * self._denominator,
            self._denominator * other._denominator,
         )
      return NotImplemented

   # Разность двух дробей
   def __sub__(self, other):
      if isinstance(other, Fraction):
         return Fraction(
            self._numerator * other._denominator
            - other._numerator * self._denominator,
            self._denominator * other._denominator,
         )
      return NotImplemented

   # Произведение двух дробей
   def __mul__(self, other):
      if isinstance(other, Fraction):
         return Fraction(
            self._numerator * other._numerator,
            self._denominator * other._denominator,
         )
      return NotImplemented
```

```
    # Частное двух дробей
    def __truediv__(self, other):
        if isinstance(other, Fraction):
            return Fraction(
                self._numerator * other._denominator,
                self._denominator * other._numerator,
            )
        return NotImplemented

    # x < y
    def __lt__(self, other):
        if isinstance(other, Fraction):
            return (
                self._numerator * other._denominator
                < other._numerator * self._denominator
            )
        return NotImplemented

    # x <= y
    def __le__(self, other):
        if isinstance(other, Fraction):
            return (
                self._numerator * other._denominator
                <= other._numerator * self._denominator
            )
        return NotImplemented

    # x == y
    def __eq__(self, other):
        if isinstance(other, Fraction):
            return (
                self._numerator * other._denominator
                == other._numerator * self._denominator
            )
        return NotImplemented

    # x != y
    def __ne__(self, other):
        if isinstance(other, Fraction):
            return (
                self._numerator * other._denominator
                != other._numerator * self._denominator
            )
        return NotImplemented

    # x > y
    def __gt__(self, other):
        if isinstance(other, Fraction):
            return (
                self._numerator * other._denominator
                > other._numerator * self._denominator
            )
        return NotImplemented

    # x >= y
    def __ge__(self, other):
        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 getABS(self):
        return Fraction(abs(self._numerator), abs(self._denominator))
```

Основная программа

main.py

```
from lib.fraction import Fraction
import itertools


def printMatrix(matrix):
    s = [[str(e) for e in row] for row in matrix]
    lens = [max(map(len, col)) for col in zip(*s)]
    fmt = "".join("  {{:{}}}".format(x) for x in lens)
    table = [fmt.format(*row) for row in s]
    print("\n".join(table))
    print("\n")


def methodJordanGauss(matrix, flag=True):
    print(f"""{'—'*14}┤ НАЧАЛЬНАЯ МАТРИЦА ├{'—'*14}\n""")
    printMatrix(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].getABS() < matrix[i][c].getABS():
                index = i
        if index != c:
            matrix[index], matrix[c] = matrix[c], matrix[index]
            print(f"""{'—'*12}┤ ПРОИЗОШЛА ПЕРЕСТАНОВКА ├{'—'*12}\n""")
            printMatrix(matrix)
        if matrix[c][c] == Fraction(0):
            continue
```

```
            # Сокращение строки
            if matrix[c][c] != Fraction(1):
                matrix[c] = [i / matrix[c][c] for i in matrix[c]]
                printMatrix(matrix)
            # По всем строкам для обнуления
            for i in range(len(matrix)):
                if matrix[i][c] == Fraction(0) or i == c:
                    continue
                coefficient = 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] * coefficient
            printMatrix(matrix)

        countNullStr = 0
        for i in matrix:
            nullSumFlag = True
            for j in i[:-1]:
                if j != Fraction(0):
                    nullSumFlag = False
                    break
            # Если элементы строки нулевые и значение не нулевое
            if nullSumFlag and i[-1] != Fraction(0):
                countNullStr = 0
                break
            # Если строка нулевая
            elif nullSumFlag and i[-1] == Fraction(0):
                continue
            countNullStr += 1

        if not countNullStr:
            return None
        elif countNullStr == len(matrix) and countNullStr == len(matrix[0]) - 1:
            return [[matrix[i][-1] for i in range(len(matrix))]]
        else:
            result = [[], []]
            # Общее решение
            for i in matrix:
                tmpSum = Fraction(0)
                for j in i:
                    tmpSum += j.getABS()
                if tmpSum != Fraction(0):
                    result[0].append(i)
            matrix = result[0]
            # Базисное решение
            if flag:
                for i in itertools.combinations(
                    [i for i in range(len(matrix[0]) - 1)], countNullStr
                ):
                    print("—" * 50)
                    print("  ", i)
                    tmpResult = [0 for i in range(len(matrix[0]) - 1)]
                    tmpMatrix = []
                    # Получение необходимых столбов, как строк
                    for j in i:
                        tmpMatrix.append([x[j] for x in matrix])
                    tmpMatrix.append([x[-1] for x in matrix])
```

```
            # Транспонирование
            tmpMatrix = [list(j) for j in zip(*tmpMatrix)]
            result_r = methodJordanGauss(tmpMatrix, False)
            if not result_r:
               print("Нет решения")
            elif len(result_r) == 1:
               print("Решение:")
               ans = result_r[0]
               for j in range(len(ans)):
                  print("  x{} = {}".format(j + 1, ans[j]), end="")
               t = 0
               for j in i:
                  tmpResult[j] = ans[t]
                  t += 1
               print("\n")
               result[1].append(tmpResult)
            elif len(result_r) == 2:
               print("\nСЛАУ имеет множество решений")
      return result


def main__choice(matrix):
   print(
      f"""{'—'*8}┤  Метод Жордана-Гаусса  ├{'—'*8}\n""",
      "\n Выберите способ матрицы:\n",
      "    1) Ввод матрицы с клавиатуры\n",
      '    2) Чтение матрицы из файла в директории "./matrices/"\n',
      "    Выход из программы - любой неперечисленный ввод",
   )
   answer = input(" > ")
   if answer == "1":
      R = int(input("Введите количество строк: "))
      for row in range(R):
         a = list(map(int, input().strip().split(" ")))
         matrix.append(list(map(Fraction, a)))
      return methodJordanGauss(matrix)
   elif answer == "2":
      filename = str(input("Введите название файла в формате .txt: "))
      try:
         f = open("matrices/" + filename + ".txt", "r")
      except:
         print(' \033[91m\033[1m[-]\033[0m Не удалось открыть файл
"input.txt".')
         return
      for line in f:
         a = list(map(int, line.strip().split(" ")))
         matrix.append(list(map(Fraction, a)))
      return methodJordanGauss(matrix)


def main():
   matrix = []
   result = main__choice(matrix)

   print(f"""{'—'*14}┤  СОКРАЩЕННАЯ  МАТРИЦА  ├{'—'*14}\n""")
   printMatrix(matrix)
   print(f"""{'—'*14}┤  Результат вычислений  ├{'—'*14}\n""")
```

```
    if not result:
        print("Нет решения")
    elif len(result) == 1:
        print("Решение:")
        ans = result[0]
        for i in range(len(ans)):
            print("   x{} = {}".format(i + 1, ans[i]))
        print("\n")
    elif len(result) == 2:
        o_ans = result[0]
        ans = result[1]

        print("Общее решение:")
        for i in o_ans:
            tmp_str = "   "
            for j in range(len(i) - 1):
                tmp = i[j]
                if tmp != Fraction(0):
                    if tmp.getABS() != Fraction(1):
                        if tmp < Fraction(0):
                            tmp_str += " - "
                        else:
                            tmp_str += " + "
                        tmp_str += str(tmp.getABS()) + " * "
                    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("   x{} = {}".format(j + 1, i[j]))
            print()
    else:
        print("\033[91m\033[1m[-]\033 Произошла не предвиденная ошибка!")
    return


if __name__ == "__main__":
    main()
```