ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ПМиК

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1  
«Решение систем линейных уравнений методом Жордана-Гаусса»  
по дисциплине «Алгоритмы и вычислительные методы оптимизации»

Выполнил: студент гр. ИП-814  
Краснов И.В.

Проверил: ассистент кафедры ПМиК   
Новожилов Д. И.

Новосибирск 2021

Содержание

[Задание 3](#_Toc65781028)

[Текст программы 4](#_Toc65781029)

[Результат работы программы 7](#_Toc65781030)

# Задание

Решение систем линейных уравнений методом Жордана-Гаусса

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.

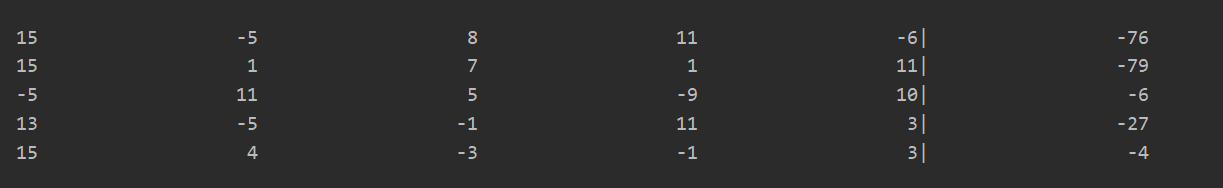
Должна иметься возможность быстро ввести входные данные для различного количества переменных и уравнений. Начальную работу программу необходимо продемонстрировать на предложенной ниже системе (система выбирается по номеру бригады).

# Текст программы

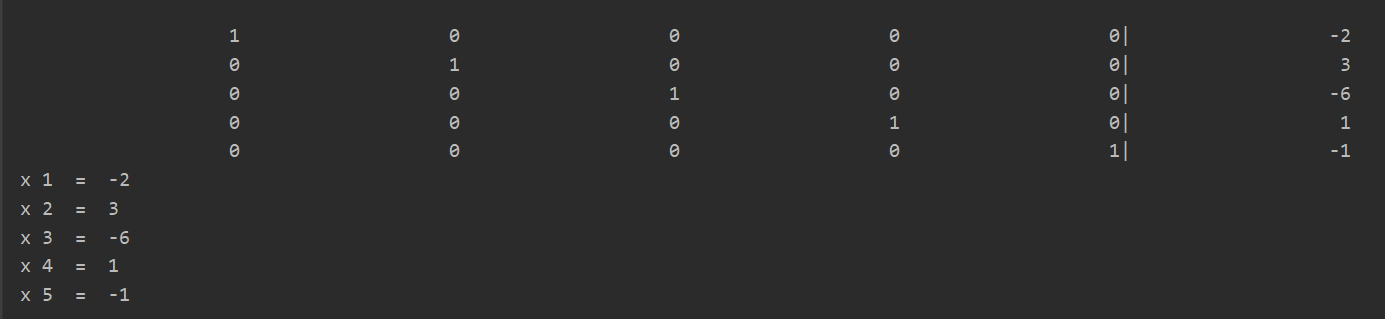
import sys  
'''import copy  
import itertools'''  
  
def gcd(m, n):  
 while m % n != 0:  
 oldm = m  
 oldn = n  
  
 m = oldn  
 n = oldm % oldn  
 return n  
  
def DeleteLine(matrix, rows, cols, line):  
 if line == rows:  
 rows -= 1  
 else:  
 for i in range(line + 1, rows):  
 for j in range(cols):  
 matrix[i][j], matrix[i - 1][j] = matrix[i - 1][j], matrix[i][j]  
 rows -= 1  
 return rows  
  
def PrintMatrix(matrix, rows, cols):  
 print()  
 for i in range(rows):  
 for j in range(cols):  
 if j == (cols - 1):  
 print("|%20s" %matrix[i][j], end='')  
 else:  
 print("%20s" %matrix[i][j], end='')  
 print()

def gauss(matrix, rows, cols):  
 k = 0  
 for t in range(rows):  
 if matrix[k][k].num == 0:  
 k += 1  
 continue  
 for i in range(rows):  
 if k == i:  
 continue  
 for j in range(k + 1, cols):  
 matrix[i][j] = ((matrix[k][k] \* matrix[i][j]) - (matrix[i][k] \* matrix[k][j])) / matrix[k][k]  
 for i in range(rows):  
 if k == i:  
 continue  
 matrix[i][k] = Fraction()  
 k += 1  
 print(t, ") ")  
 PrintMatrix(matrix, rows, cols)  
 print()  
 k = 0  
 for i in range(rows):  
 if matrix[k][k].num == 0:  
 k += 1  
 continue  
 for j in range(cols - 1, k - 1, -1):  
 matrix[i][j] /= matrix[k][k]  
 k += 1  
 i = 0  
 while i < rows:  
 zero\_line = True  
 for k in range(cols - 1):  
 if matrix[i][k].num != 0:  
 zero\_line = False  
 break  
 if zero\_line and matrix[i][cols - 1].num != 0:  
 PrintMatrix(matrix, rows, cols)  
 print("Нет решения!")  
 sys.exit()  
 elif zero\_line == True and matrix[i][cols - 1].num == 0:  
 rows = DeleteLine(matrix, rows, cols, i)  
 i += 1  
 '''if (zero\_line):  
 continue'''  
 PrintMatrix(matrix, rows, cols)  
 return rows  
  
  
class Fraction:  
 def \_\_init\_\_(self, top=0, bottom=1):  
 if bottom != 0:  
 self.num = top  
 self.den = bottom  
 if self.den < 0:  
 self.num \*= -1  
 self.den \*= -1  
 tmp = gcd(self.num, self.den)  
 self.num = self.num // tmp  
 self.den = self.den // tmp  
 else:  
 print("Denominator == 0")  
 sys.exit()  
  
 def \_\_str\_\_(self):  
 if self.den == 1:  
 return str(self.num)  
 else:  
 return str(self.num) + "/" + str(self.den)  
  
 def show(self):  
 if self.den == 1:  
 print(self.num)  
 else:  
 print(self.num, "/", self.den)  
  
 def \_\_add\_\_(self, otherfraction):  
 return Fraction(self.num \* otherfraction.den + self.den \* otherfraction.num, self.den \* otherfraction.den)  
  
 def \_\_iadd\_\_(self, other):  
 tmp = Fraction(self.num \* other.den + self.den \* other.num, self.den \* other.den)  
 self = tmp  
 return tmp  
  
 def \_\_isub\_\_(self, other):  
 tmp = Fraction(self.num \* other.den - self.den \* other.num, self.den \* other.den)  
 self = tmp  
 return tmp  
  
 def \_\_imul\_\_(self, other):  
 tmp = Fraction(self.num \* other.num, self.den \* other.den)  
 self = tmp  
 return tmp  
  
 def \_\_itruediv\_\_(self, other):  
 tmp = Fraction(self.num \* other.den, self.den \* other.num)  
 self = tmp  
 return tmp  
  
 def \_\_sub\_\_(self, otherfraction):  
 return Fraction(self.num \* otherfraction.den - self.den \* otherfraction.num, self.den \* otherfraction.den)  
  
 def \_\_mul\_\_(self, otherfraction):  
 return Fraction(self.num \* otherfraction.num, self.den \* otherfraction.den)  
  
 def \_\_truediv\_\_(self, otherfraction):  
 return Fraction(self.num \* otherfraction.den, self.den \* otherfraction.num)  
  
  
 def \_\_eq\_\_(self, other):  
 firstnum = self.num \* other.den  
 secondnum = other.num \* self.den  
 return firstnum == secondnum  
  
 def \_\_ne\_\_(self, other):  
 firstnum = self.num \* other.den  
 secondnum = other.num \* self.den  
 return firstnum != secondnum  
  
  
number\_of\_rows = int(input())  
number\_of\_cols = int(input())  
matrixCin = [[0] \* number\_of\_cols for i in range(number\_of\_rows)]  
matrix = [[Fraction(0, 1)] \* number\_of\_cols for i in range(number\_of\_rows)]  
  
for i in range(number\_of\_rows):  
 stri = input()  
 matrixCin[i] = stri.strip().split(" ")  
 for j in range(number\_of\_cols):  
 matrix[i][j] = Fraction(int(matrixCin[i][j]), 1)  
  
PrintMatrix(matrix, number\_of\_rows, number\_of\_cols)  
  
number\_of\_rows = gauss(matrix, number\_of\_rows, number\_of\_cols)  
sign = Fraction(-1, 1)  
for i in range(number\_of\_rows):  
 print('x', i + 1, " = ", matrix[i][number\_of\_cols - 1], end=" ")  
 for j in range(number\_of\_cols - 1):  
 if j != i and matrix[i][j].num != 0:  
 if matrix[i][j].num > 0:  
 print(" - ", matrix[i][j], 'x', j + 1, end=" ")  
 else:  
 print(" + ", matrix[i][j] \* sign, 'x', j + 1, end=" ")  
 print()

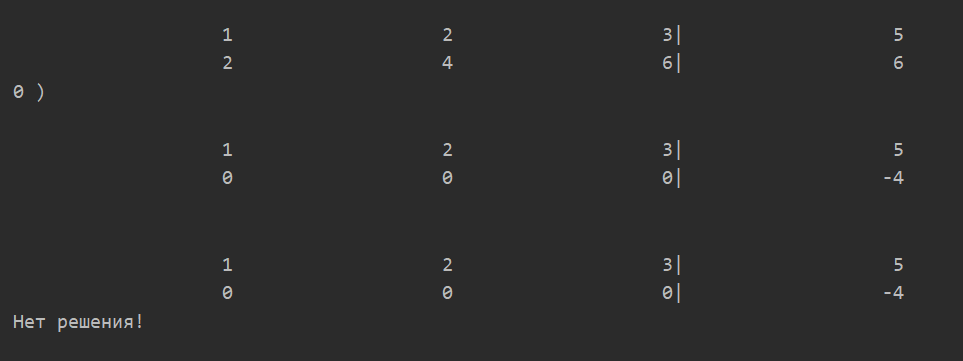
# Результат работы программы

**

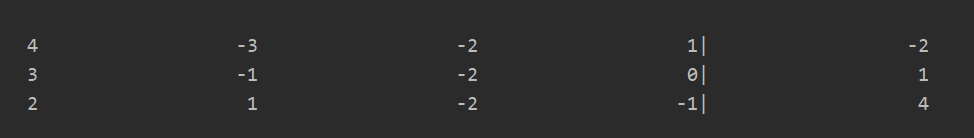
*Рис. 1 Система из задания*

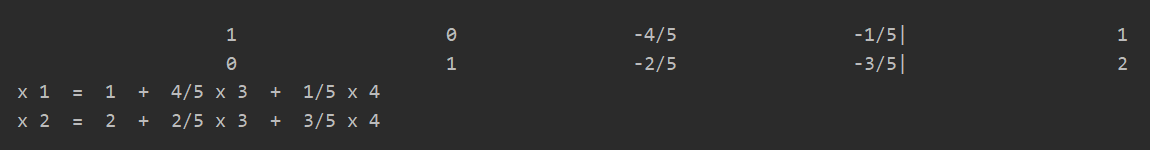
******

*Рис. 2 Решение системы*

******

*Рис. 3 Система, которая не имеет решения*

******

******

*Рис. 4, 5 Система, имеющая бесконечно много решений*