ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ПМиК

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2  
«Нахождение всех базисных решений системы линейных уравнений»  
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 14](#_Toc65781030)

# Задание

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. 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.

# 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 NextSet(a, n, m):

'''while True:

i = m - 1

while i >= 0 and a[i] + m - i + 1 > n:

i -= 1

if i < 0:

return

print(a[i])

a[i] += 1

for j in range(i + 1, m):

a[j] = a[j - 1] + 1

#Print(a, m)'''

k = m

for i in range(k - 1, -1, -1):

if a[i] < (n - k + i + 1):

a[i] += 1

for j in range(i + 1, k):

a[j] = a[j - 1] + 1

return True

return False

def Print(a, m):

for i in range(m):

print("x", a[i], end = ' ')

print()

'''def comb(n, k):

#d = list(range(0, k))

d = [i + 1 for i in range(k - 1)]

yield d

while True:

i = k - 1

while i >= 0 and d[i] + k - i + 1 > n:

i -= 1

if i < 0:

return

d[i] += 1

for j in range(i + 1, k):

d[j] = d[j - 1] + 1

yield d'''

def perm(new\_matrix, k, line, rows, cols):

prov = True

for i in range(k + 1, rows):

for j in range(cols):

if new\_matrix[i][line] != 0:

prov = False

new\_matrix[i][j], new\_matrix[i - 1][j] = new\_matrix[i - 1][j], new\_matrix[i][j]

if prov == False:

break

'''def combinations(elements, size):

if len(elements) == size or size == 1:

return elements

ret = []

for i, item in enumerate(elements):

for j in combinations(elements[i + 1:], size - 1):

ret.append(item + j)

return ret

'''

def find\_basis(new\_matrix, matrix, a, rows, cols):

new\_matrix = copy.deepcopy(matrix)

answer = [Fraction(0, 1) for i in range(cols)]

sign = Fraction(-1, 1)

for i in range(rows):

if new\_matrix[i][a[i] - 1].num == 0:

j = i + 1

while j < rows:

if new\_matrix[j][a[i] - 1].num != 0:

for k in range(cols + 1):

new\_matrix[i][k], new\_matrix[j][k] = new\_matrix[j][k], new\_matrix[i][k]

break

else :

j += 1

if j == rows:

print("Не могут быть вместе в базисе")

new\_matrix = []

return

PrintMatrix(new\_matrix, rows, cols + 1)

for i in range(rows):

temp = Fraction(new\_matrix[i][a[i] - 1].num, new\_matrix[i][a[i] - 1].den)

for j in range(cols + 1):

new\_matrix[i][j] /= temp

PrintMatrix(new\_matrix, rows, cols + 1)

for i in range(rows):

temp = Fraction(new\_matrix[i][a[i] - 1].num, new\_matrix[i][a[i] - 1].den)

for j in range(cols + 1):

new\_matrix[i][j] /= temp

for j in range(rows):

if i == j:

continue

else:

if new\_matrix[j][a[i] - 1].num != 0:

temp = Fraction(new\_matrix[j][a[i] - 1].num, new\_matrix[j][a[i] - 1].den)

#temp1 = Fraction(new\_matrix[i][a[i] - 1].num, new\_matrix[i][a[i] - 1].den)

for k in range(cols + 1):

temp1 = Fraction(new\_matrix[i][k].num, new\_matrix[i][k].den)

new\_matrix[j][k] += temp \* temp1 \* sign

#PrintMatrix(new\_matrix, rows, cols + 1)

PrintMatrix(new\_matrix, rows, cols + 1)

for i in range(rows):

answer[a[i] - 1] = Fraction(new\_matrix[i][cols].num, new\_matrix[i][cols].den)

#answer[j] = Fraction(new\_matrix[k][cols - 1].num, new\_matrix[k][cols - 1].den)

#Print(a, rows)

print("Ответ (", end="")

for i in range(cols):

print(answer[i], end=" ")

answer[i] = Fraction(0, 1)

print(")")

def basis(matrix, rows, cols):

new\_matrix = copy.deepcopy(matrix)

a = [i + 1 for i in range(cols - 1)]

Print(a, rows)

find\_basis(new\_matrix, matrix, a, rows, cols - 1)

if cols - 1 >= rows:

while NextSet(a, cols - 1, rows):

Print(a, rows)

find\_basis(new\_matrix, matrix, a, rows, cols - 1)

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

return Fraction(self.num \* other, self.den)'''

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=" ")

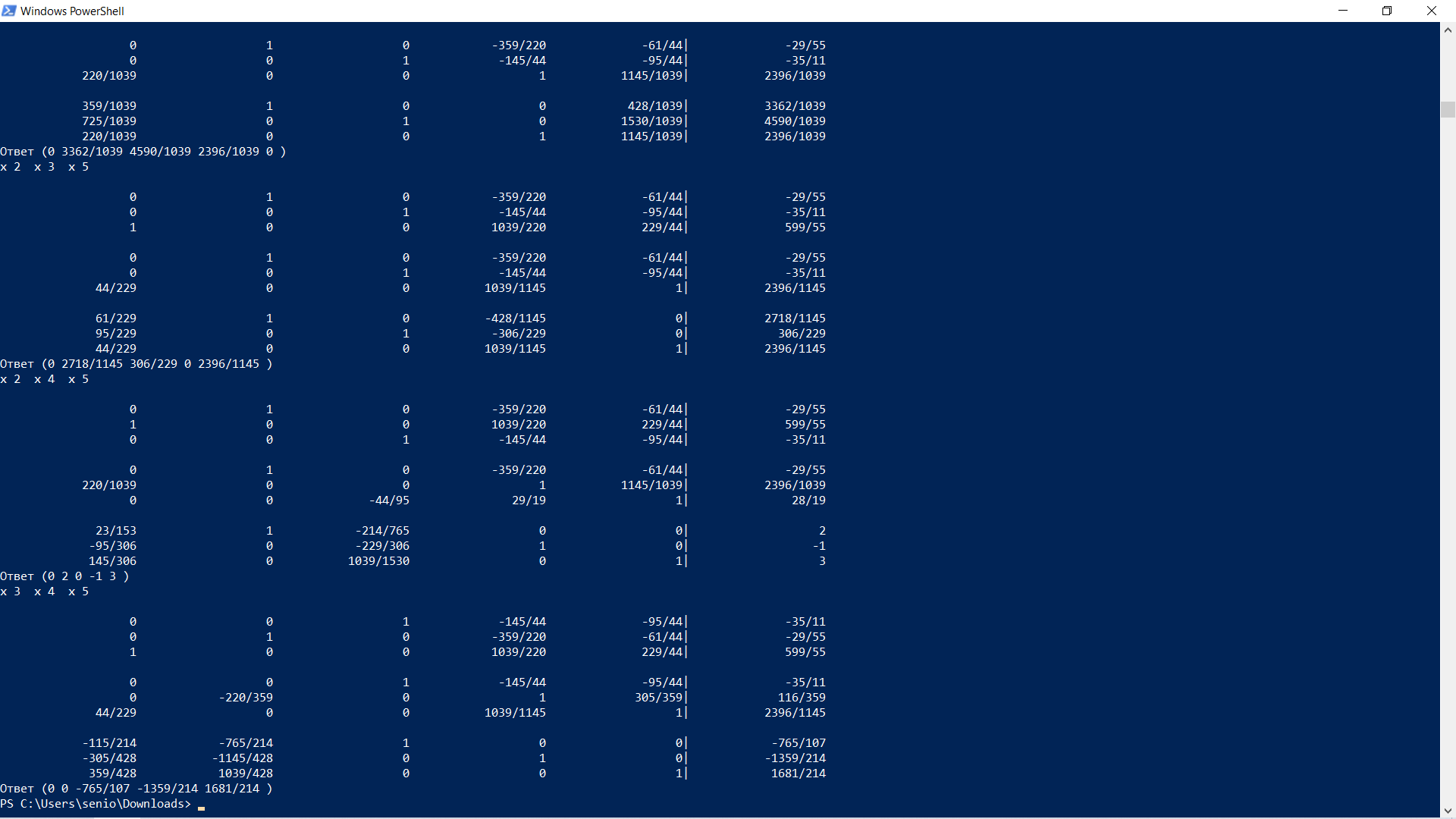
else:

print(" + ", matrix[i][j] \* sign, 'x', j + 1, end=" ")

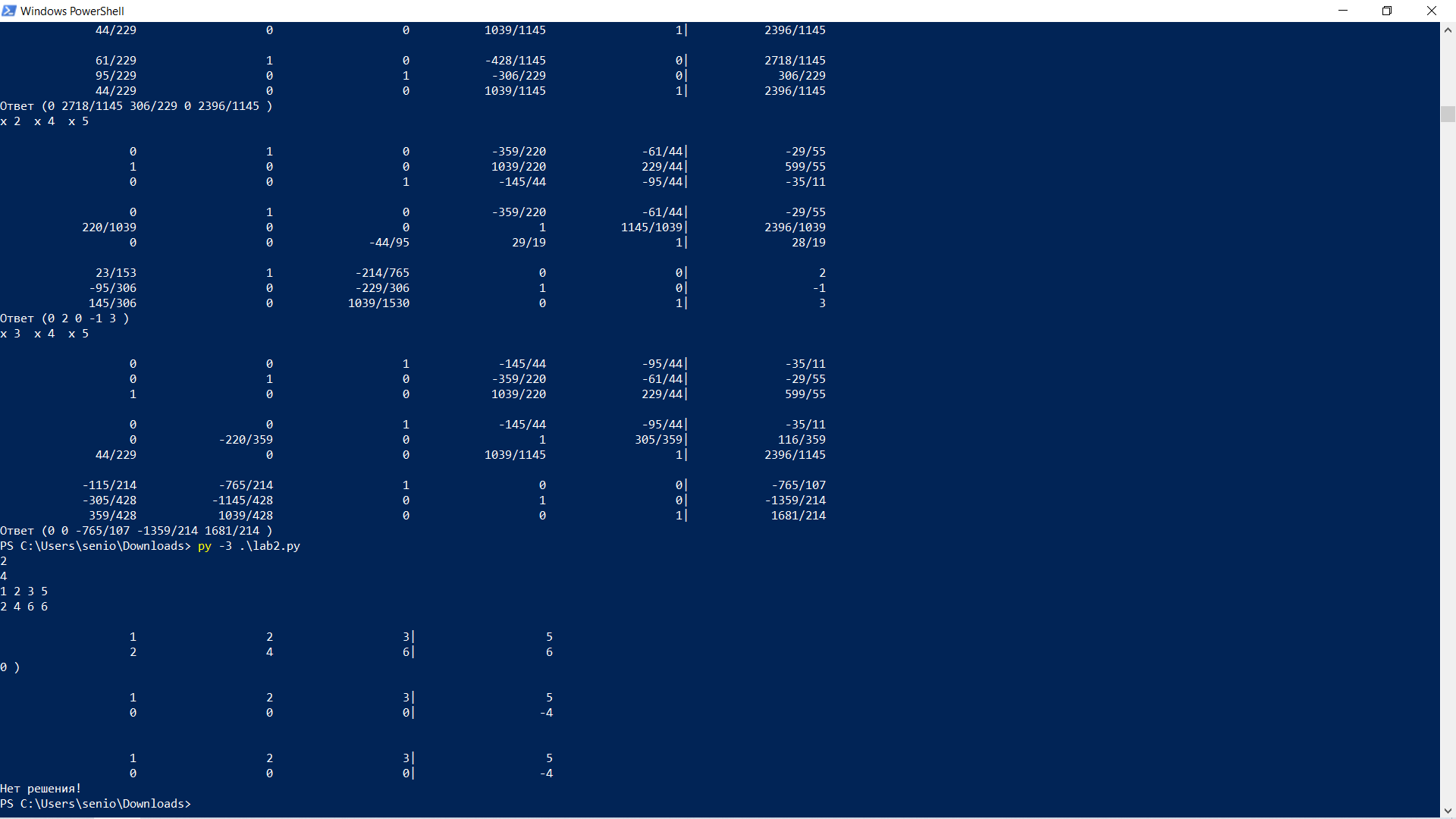
print()

basis(matrix, number\_of\_rows, number\_of\_cols)

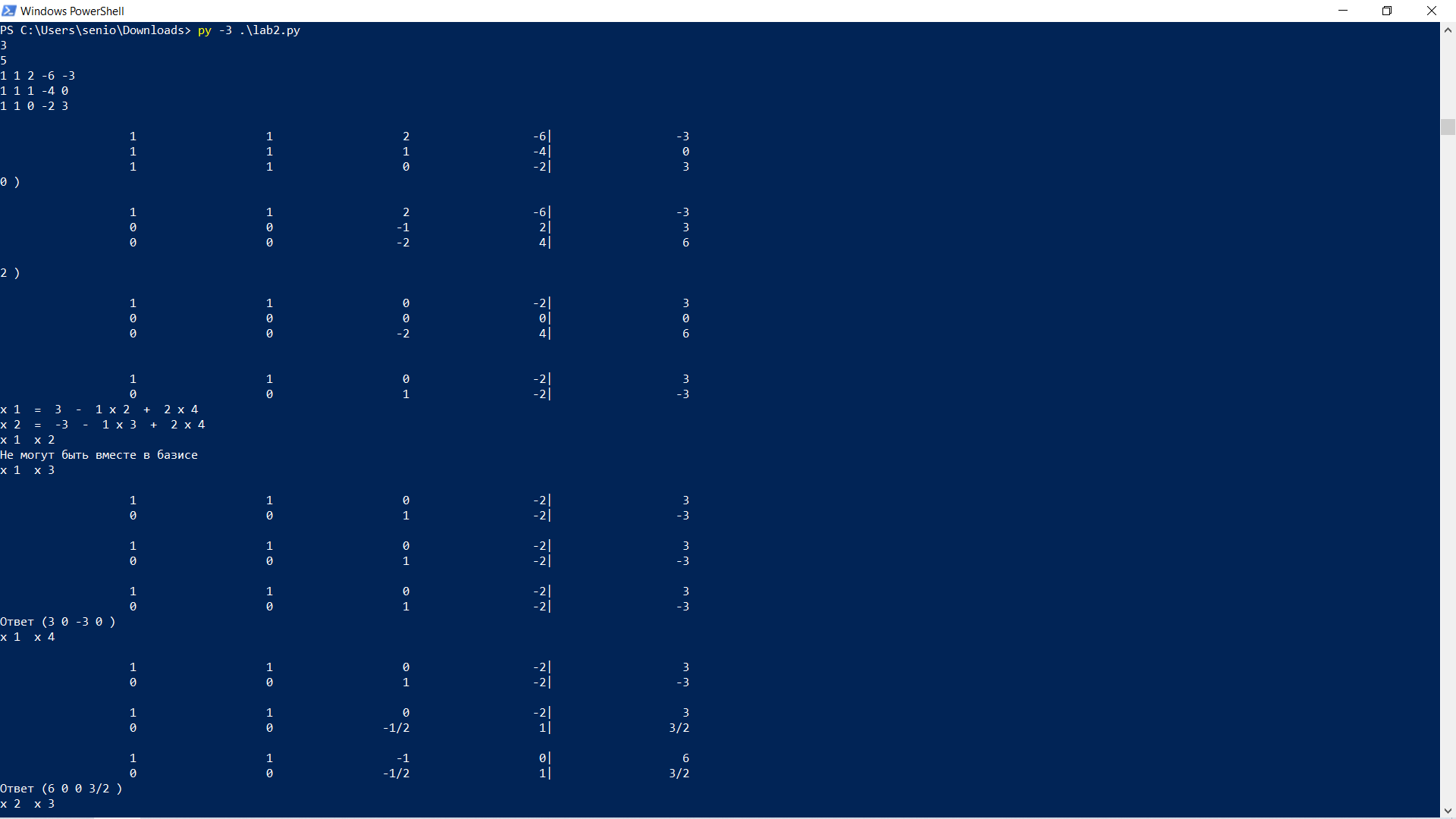
# Результат работы программы



*Рис. 1 Решение системы из задания (вариант 1)*



*Рис. 2 Система, не имеющая решений*



*Рис. 3 Система, в которой некоторые переменные не могут ю\быть вместе в базисе*