

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
Текст программы	4
Результат работы программы.....	14

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

0	1	0	-359/220	-61/44	-29/55
0	0	1	-145/44	-95/44	-35/11
220/1039	0	0	1	1145/1039	2396/1039
359/1039	1	0	0	428/1039	3362/1039
725/1039	0	1	0	1530/1039	4590/1039
220/1039	0	0	1	1145/1039	2396/1039
Ответ (0 3362/1039 4590/1039 2396/1039 0)					
x 2 x 3 x 5					
0	1	0	-359/220	-61/44	-29/55
0	0	1	-145/44	-95/44	-35/11
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
44/229	0	0	1039/1145	1	2396/1145
61/229	1	0	-428/1145	0	2718/1145
95/229	0	1	-306/229	0	306/229
44/229	0	0	1039/1145	1	2396/1145
Ответ (0 2718/1145 306/229 0 2396/1145)					
x 2 x 4 x 5					
0	1	0	-359/220	-61/44	-29/55
1	0	0	1039/220	229/44	599/55
0	0	1	-145/44	-95/44	-35/11
0	1	0	-359/220	-61/44	-29/55
220/1039	0	0	1	1145/1039	2396/1039
0	0	-44/95	29/19	1	28/19
23/153	1	-214/765	0	0	2
-95/306	0	-229/306	1	0	-1
145/306	0	1039/1530	0	1	3
Ответ (0 2 0 -1 3)					
x 3 x 4 x 5					
0	0	1	-145/44	-95/44	-35/11
0	1	0	-359/220	-61/44	-29/55
1	0	0	1039/220	229/44	599/55
0	0	1	-145/44	-95/44	-35/11
0	-220/359	0	1	305/359	116/359
44/229	0	0	1039/1145	1	2396/1145
-115/214	-765/214	1	0	0	-765/107
-305/428	-1145/428	0	1	0	-1359/214
359/428	1039/428	0	0	1	1681/214
Ответ (0 0 -765/107 -1359/214 1681/214)					

Рис. 1 Решение системы из задания (вариант 1)

2					
4					
1 2 3 5					
2 4 6 6					
	1	2	3	5	
	2	4	6	6	
0)					
	1	2	3	5	
	0	0	0	-4	
	1	2	3	5	
	0	0	0	-4	
Нет решения!					

Рис. 2 Система, не имеющая решений

```

5
1 1 2 -6 -3
1 1 1 -4 0
1 1 0 -2 3

0 )
1 1 2 -6| -3
1 1 -4| 0
1 1 -2| 3

2 )
1 1 2 -6| -3
0 0 -1| 3
0 0 -2| 6

3 )
1 1 0 -2| 3
0 0 0 -2| 6
0 0 0 4| 6

x 1 = 3 - 1 x 2 + 2 x 4
x 2 = -3 - 1 x 3 + 2 x 4
x 1 x 2
Не могут быть вместе в базисе
x 1 x 3

1 1 0 -2| 3
0 0 1 -2| -3

1 1 0 -2| 3
0 0 1 -2| -3

1 1 0 -2| 3
0 0 1 -2| -3

Ответ (3 0 -3 0 )
x 1 x 4

1 1 0 -2| 3
0 0 0 -1| -3

1 1 0 -2| 3
0 0 -1/2 1| 3/2

1 1 0 0| 6
0 0 -1/2 1| 3/2

Ответ (6 0 0 3/2 )

```

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