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 №3

Выполнил:

студент гр. ИП-012 Николаев А.Д. \_

ФИО студента

«13» февраля 2023 г.

Новосибирск 2023 г.

**Оглавление**

[Постановка задачи 3](#_Toc127393577)

[Примеры работы программы 4](#_Toc127393578)

[Исходный код программы 6](#_Toc127393579)

# **Постановка задачи**

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

На вход программе подаются коэффициенты системы линейный уравнений (считываются из файла в виде матрицы размера m x (n+1)):

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

# **Примеры работы программы**

Изображение выглядит как стол

Автоматически созданное описание

Рис. 1 Пример работы программы с единственным решением (1-ая часть)

Изображение выглядит как стол

Автоматически созданное описание

Рис. 2 Пример работы программы с единственным решением (2-ая часть)

Изображение выглядит как стол

Автоматически созданное описание

Рис. 3 Пример работы программы с единственным решением (3-ья часть)

Изображение выглядит как стол

Автоматически созданное описание

Рис. 4 Пример работы программы с единственным решением (4-ая часть)

Изображение выглядит как стол

Автоматически созданное описание

Рис. 5 Пример работы программы с единственным решением (5-ая часть)

# **Исходный код программы**

import java.io.File

import kotlin.math.abs

import java.util.Scanner

fun gcd(a: Long, b: Long) : Long

{

if((a%b)==0L) return b;

if((b%a)==0L) return a;

if(a>b) return gcd(a%b,b);

return gcd(a,b%a);

}

fun lcm(a: Long, b: Long) : Long

{

return (a\*b)/gcd(a,b);

}

class Frac(\_num: Long=0,\_den: Long=1)

{

var num: Long=0

var den: Long=1

init

{

num=\_num

den=\_den

}

operator fun plus(other: Frac) : Frac

{

val LCM=lcm(den,other.den)

val Num: Long=num\*(LCM/den)+other.num\*(LCM/other.den)

if(Num==0L) return Frac()

else

{

val GCD=gcd(abs(Num),LCM)

return Frac(Num/GCD,LCM/GCD)

}

}

operator fun minus(other: Frac) : Frac

{

return this+Frac(-other.num,other.den)

}

operator fun times(other: Frac) : Frac

{

if((num==0L) or (other.num==0L)) return Frac(0);

val GCD\_a=gcd(abs(num),abs(other.den))

val GCD\_b=gcd(abs(other.num),abs(den))

return Frac((num/GCD\_a)\*(other.num/GCD\_b),(den/GCD\_b)\*(other.den/GCD\_a))

}

operator fun div(other: Frac) : Frac

{

if(other.num<0L) return this\*Frac(-other.den,-other.num)

return this\*Frac(other.den,other.num)

}

operator fun compareTo(other:Frac) : Int

{

val a=num.toDouble()\*other.den.toDouble()

val b=den.toDouble()\*other.num.toDouble()

return (a-b).toInt()

}

override fun toString(): String

{

return "($num/$den)"

}

fun inTable(): String

{

return String.format("%8d/%-8d",num,den)

}

}

fun printMatr(A: MutableList<MutableList<Frac>>,B: MutableList<Frac>)

{

for(i in 0 until A.size)

{

for(j in 0 until A[i].size) print(A[i][j].inTable())

print(" | ")

println(B[i].inTable())

}

println("")

}

fun calculateMatr(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>, k: Int, s:Int)

{

val temp=Frac(A[k][s].num,A[k][s].den)

var temp2: Frac

for(j in 0 until A[k].size) A[k][j]=(A[k][j]/temp)

B[k]=(B[k]/temp)

for(i in 0 until A.size)

{

if(i==k) continue

temp2=Frac(A[i][s].num,A[i][s].den)

for(j in 0 until A[k].size)

{

A[i][j]=A[i][j]-A[k][j]\*temp2

}

B[i]=B[i]-(B[k]\*temp2)

}

}

fun checkMatr(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>): Int

{

var size=A.size

var flag: Boolean

var i=0

while(i<size)

{

flag=true

for(j in 0 until A[i].size)

{

if(A[i][j].num!=0L)

{

flag=false

break

}

}

if(flag)

{

if(B[i].num==0L)

{

size--

B.removeAt(i)

A.removeAt(i)

i--

}

else return -1

}

i++

}

return size

}

fun printSolution(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>, SetNum: MutableList<Int>)

{

val r=SetNum.size

val n=A[0].size

for(i in 0 until r)

{

print("x${SetNum[i]+1}=${B[i]}")

for(j in 0 until n) if(j !in SetNum) print(" - ${A[i][j]}x${j+1}")

println(";")

}

println("\n")

}

fun methodJordanGauss(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>) : Int

{

var max: Int?

var n=A[0].size

var m=A.size

var r=m

var k=0

var s=0

printMatr(A,B)

while(k<r)

{

max=null

for(i in k until m)

{

if(A[i][s].num!=0L)

{

if(max==null) max=i

else if(A[max][s]<A[i][s]) max=i

}

}

if(max!=null)

{

val temp\_A=A[max]

A[max]=A[k]

A[k]=temp\_A

val temp\_B=B[max]

B[max]=B[k]

B[k]=temp\_B

calculateMatr(A,B,k,s)

m=checkMatr(A,B)

r=m

if(m==-1)

{

println("System hasn't solution")

return -1

}

k++

}

s++

}

return r

}

fun methodJordanGauss(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>, SetNum: MutableList<Int>, count: Int): Boolean

{

var flag: Boolean

var n=A[0].size

var m=A.size

var r=m

var k=0

var s=0

printMatr(A,B)

while(k<r)

{

flag=false

s=SetNum[k]

for(i in k until m)

{

if(A[i][s].num!=0L)

{

flag=true

val temp\_A=A[i]

A[i]=A[k]

A[k]=temp\_A

val temp\_B=B[i]

B[i]=B[k]

B[k]=temp\_B

calculateMatr(A,B,k,s)

printMatr(A,B)

k++

break

}

}

if(!flag)

{

for(Num in SetNum)

{

print("x${Num+1} ")

}

println("- can not be basis solution")

return false

}

}

for(Num in SetNum)

{

print("x${Num+1} ")

}

println("- is ${count+1} Basis Solution\n")

printSolution(A,B,SetNum)

return true

}

fun nextSetNum(SetNum: MutableList<Int>, p: Int, n:Int) : Int

{

val k=SetNum.size

val p2: Int

SetNum[p]+=1

for(i in p+1 until k) SetNum[i]=SetNum[i-1]+1

if(SetNum[k-1]==n-1) p2=p-1

else p2=k-1

return p2

}

fun allBasisSolution(A: MutableList<MutableList<Frac>>, B: MutableList<Frac>, r: Int)

{

var SetNum = MutableList(2,{0})

SetNum.clear()

val n=A[0].size

var count=0

var p=r-1

for(i in 0 until r) SetNum.add(i)

if(methodJordanGauss(A,B,SetNum,count)) count++

while(p>=0)

{

p=nextSetNum(SetNum,p,n)

if(methodJordanGauss(A,B,SetNum,count)) count++

}

println("Всего $count базисных решений")

}

fun main(){

val scan=Scanner(File("In1.txt"))

val n=scan.nextInt()

val m=scan.nextInt()

var Matr=MutableList(m, {MutableList(n,{Frac(0)})})

var Vect\_B=MutableList(m,{Frac(0)})

var SetNum = MutableList(2,{0})

SetNum.clear()

for(i in 0..m-1)

{

for(j in 0..n-1)

{

Matr[i][j].num=scan.nextInt().toLong()

}

Vect\_B[i].num=scan.nextInt().toLong()

}

val r=methodJordanGauss(Matr,Vect\_B)

if(r!=-1) allBasisSolution(Matr,Vect\_B,r)

}