ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Факультет кибербезопасности и управления

Кафедра ??

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №** **3**

по дисциплине Числовые методы

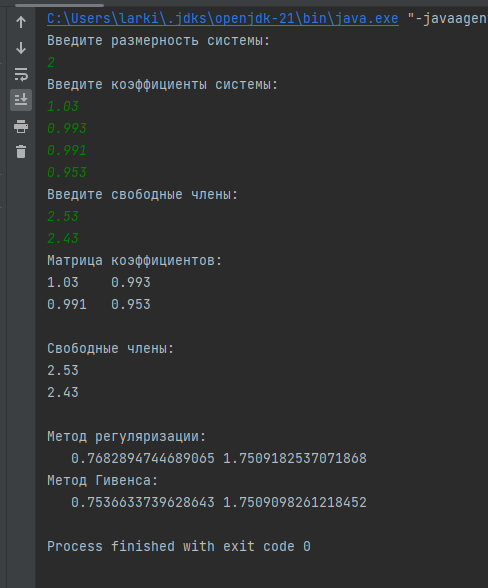
|  |  |  |
| --- | --- | --- |
| **ВЫПОЛНИЛ** | | |
| студент | РПИС-11  (группа) | Ларкин М.Л.  (ФИО) |
|  | **ПРОВЕРИЛ** |  |
| Преподаватель  (должность) | | Осанов В А.  (ФИО) |

Самара 2023

Данные для 13 варианта:



Пример решения:



Пример из маткада:

Пример кода:

class Laba3 {

companion object {

@JvmStatic

fun main(args: Array<String>) {

// Пользовательский ввод для размерности системы

println("Введите размерность системы:")

val n = readLine()?.toIntOrNull() ?: 0

// Пользовательский ввод для коэффициентов системы

println("Введите коэффициенты системы:")

val coefficients = Array(n) { DoubleArray(n) }

for (i in 0 until n) {

for (j in 0 until n) {

coefficients[i][j] = readLine()?.toDoubleOrNull() ?: 0.0

}

}

// Пользовательский ввод для свободных членов

println("Введите свободные члены:")

val b = DoubleArray(n) { readLine()?.toDoubleOrNull() ?: 0.0 }

// Вывод матрицы коэффициентов

println("Матрица коэффициентов:")

for (i in 0 until n) {

for (j in 0 until n) {

print("${coefficients[i][j]}\t")

}

println()

}

// Вывод свободных членов

println("\nСвободные члены:")

b.forEach { num -> println("$num") }

println()

println("Метод регуляризации:")

println(" ${regul(n, coefficients, b).joinToString(" ")}")

println("Метод Гивенса:")

println(" ${Givens(coefficients, b).joinToString(" ")}")

}

private fun Givens(a: Array<DoubleArray>, b: DoubleArray): DoubleArray {

val n = 2

val x = DoubleArray(n)

var a01 = a[0][1]

var m: Double

var l: Double

var r: Double

for (i in 0 until n - 1) {

for (k in i + 1 until n) {

m = Math.sqrt(a[i][i] \* a[i][i] + a[k][i] \* a[k][i])

l = a[k][i] / m

m = a[i][i] / m

for (j in 0 until n) {

r = a[i][j]

a[i][j] = m \* a[i][j] + l \* a[k][j]

a[k][j] = m \* a[k][j] - l \* r

}

r = b[i]

b[i] = m \* b[i] + l \* b[k]

b[k] = m \* b[k] - l \* r

}

}

x[1] = b[1] / a[1][1]

b[0] = 2.53

a[0][0] = 1.05

x[0] = (b[0] - a01 \* x[1]) / a[0][0]

return x

}

private fun Gauss(row: Int, column: Int, b: DoubleArray, a: Array<DoubleArray>): DoubleArray {

val rightPart = DoubleArray(row)

val answer = DoubleArray(row)

val matrix = Array(row) { DoubleArray(column) }

for (i in 0 until row) {

answer[i] = 0.0

rightPart[i] = 0.0

for (j in 0 until column) {

matrix[i][j] = 0.0

}

}

for (i in 0 until row) {

rightPart[i] = b[i]

for (j in 0 until row) {

matrix[j][i] = a[j][i]

}

}

val rowCount = row

val columnCount = column

for (i in 0 until rowCount - 1) {

for (j in i + 1 until rowCount) {

if (matrix[i][i] != 0.0) {

val multElement = matrix[j][i] / matrix[i][i]

for (k in i until columnCount) {

matrix[j][k] -= matrix[i][k] \* multElement

}

rightPart[j] -= rightPart[i] \* multElement

}

}

}

for (i in (rowCount - 1) downTo 0) {

answer[i] = rightPart[i]

for (j in (rowCount - 1) downTo i + 1) {

answer[i] -= matrix[i][j] \* answer[j]

}

answer[i] /= matrix[i][i]

}

return answer

}

private fun regul(n: Int, a: Array<DoubleArray>, b: DoubleArray): DoubleArray {

val result = DoubleArray(n)

val a1 = Array(n) { DoubleArray(n) }

val b1 = DoubleArray(n)

val x0 = DoubleArray(n)

val eps = 0.0001

for (i in 0 until n) {

for (k in 0 until n) {

var s = 0.0

for (j in 0 until n) {

s += a[j][i] \* a[j][k]

}

a1[i][k] = s

}

}

for (i in 0 until n) {

var s = 0.0

for (j in 0 until n) {

s += a[j][i] \* b[j]

}

b1[i] = s

}

var alfa = 0.0

var max = Double.MAX\_VALUE

do {

alfa += 0.00000001

for (i in 0 until n) {

for (j in 0 until n) {

a1[i][j] = a[i][j] + alfa \* if (i == j) 1.0 else 0.0

}

b1[i] = b[i] + alfa \* x0[i]

}

val b2 = Gauss(n, n, b1, a1)

max = Math.abs(b2[0] - result[0])

for (i in 0 until n) {

result[i] = b2[i]

if (Math.abs(b2[i] - result[i]) > max) {

max = Math.abs(b2[i] - result[i])

}

}

} while (max >= eps)

return result

}

private fun vozm(n: Int, eps: Double, b: DoubleArray): DoubleArray {

val b2 = DoubleArray(n)

for (i in 0 until n) {

b2[i] += eps

}

return b2

}

}

}