**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет: ПИиКТ

**Отчет**

по лабораторной работе №1

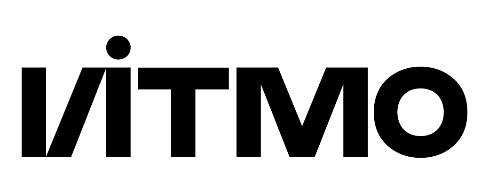
«Решение системы линейных алгебраических уравнений СЛАУ»

по дисциплине «**Вычислительная математика**»

Выполнил: Трусковский Г.А.

Группа: Р3214

Проверил: Малышева Т.А.



Санкт-Петербург 2025

**Цель работы**

Изучение методов решения СЛАУ и реализация одного из них по варианту с помощью выбранного языка программирования.

**Описание метода**

Классический метод решения системы линейных алгебраических уравнений (СЛАУ). Назван в честь немецкого математика Карла Фридриха Гаусса. Это метод последовательного исключения переменных, когда с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы.

**Листинг программы**

[**https://github.com/Sinchi1/FirstLabBi4Mat/blob/master/src/main/kotlin/Runner/GaussFunction.kt**](https://github.com/Sinchi1/FirstLabBi4Mat/blob/master/src/main/kotlin/Runner/GaussFunction.kt)

private fun gauss(A: MutableList<MutableList<Double>>, B: MutableList<Double>): List<Double>? {  
 val n = B.size  
  
 if (A.size != n || A.*any* **{ it**.size != n **}**) {  
 *println*("Ошибка: Матрица коэффициентов должна быть квадратной и соответствовать размеру вектора правых частей!")  
 return null  
 }  
  
 var column = 0  
 while (column < n) {  
 *println*("\nИщем максимальный по модулю элемент в ${column + 1}-м столбце:")  
 var currentRow: Int? = null  
  
 for (r in column *until* n) {  
 if (currentRow == null || kotlin.math.*abs*(A[r][column]) > kotlin.math.*abs*(A[currentRow][column])) {  
 currentRow = r  
 }  
 }  
  
 if (currentRow == null || A[currentRow][column] == 0.0) {  
 *println*("\n Решений нет (нулевой столбец в матрице).")  
 return null  
 }  
  
 program.customPrint(A, B, Pair(currentRow, column))  
  
 if (currentRow != column) {  
 *println*("\nПереставляем строку с найденным элементом повыше:")  
 program.swapRows(A, B, currentRow, column)  
 program.customPrint(A, B, Pair(column, column))  
 }  
  
 *println*("\nНормализуем строку с найденным элементом:")  
 val pivot = A[column][column]  
 if (pivot == 0.0) {  
 *println*("Ошибка: деление на ноль!")  
 return null  
 }  
  
 program.divideRow(A, B, column, pivot)  
 program.customPrint(A, B, Pair(column, column))  
  
 *println*("\nОбрабатываем нижележащие строки:")  
 for (r in column + 1 *until* n) {  
 program.combineRows(A, B, r, column, -A[r][column])  
 }  
  
 program.customPrint(A, B, Pair(column, column))  
 column++  
 }  
  
 *println*("\nМатрица приведена к треугольному виду, считаем решение... \n")  
 val X = *MutableList*(n) **{** 0.0 **}** for (i in n - 1 *downTo* 0) {  
 var sum = 0.0  
 for (j in i + 1 *until* n) {  
 sum += X[j] \* A[i][j]  
 }  
 X[i] = B[i] - sum  
 }  
  
 *println*("Получили ответ:")  
 X.*forEachIndexed* **{** index, x **->** *println*("X${index + 1} =\t${"%.2f".*format*(Locale.*US*, x)}") **}** return X  
}  
  
private fun determinant(matrix: List<List<Double>>): Double {  
 val n = matrix.size  
 if (matrix.*any* **{ it**.size != n **}**) {  
 *println*("Ошибка: Матрица должна быть квадратной для вычисления определителя!")  
 return Double.NaN  
 }  
  
 val A = matrix.*map* **{ it**.*toMutableList*() **}**.*toMutableList*()  
 var det = 1.0  
  
 for (i in 0 *until* n) {  
 var pivotRow = i  
 for (r in i *until* n) {  
 if (kotlin.math.*abs*(A[r][i]) > kotlin.math.*abs*(A[pivotRow][i])) {  
 pivotRow = r  
 }  
 }  
  
 if (A[pivotRow][i] == 0.0) return 0.0  
  
 if (pivotRow != i) {  
 Collections.swap(A, i, pivotRow)  
 det = -det  
 }  
  
 val pivot = A[i][i]  
 if (pivot == 0.0) return 0.0  
 det \*= pivot  
  
 for (r in i + 1 *until* n) {  
 val factor = A[r][i] / pivot  
 for (c in i *until* n) {  
 A[r][c] -= factor \* A[i][c]  
 }  
 }  
 }  
 return det  
}

**Пример работы программы**

Введите 1 если ввод с клавиатуры

Введите 0 если ввод с файла

Ввод:

0

Введите название файла:

note.txt

Исходная система:

1.000 2.000 3.000 | 1.000

3.000 2.000 5.000 | 1.000

0.000 1.000 2.000 | 1.000

Решаем:

Ищем максимальный по модулю элемент в 1-м столбце:

1.000 2.000 3.000 | 1.000

3.000\* 2.000 5.000 | 1.000

0.000 1.000 2.000 | 1.000

Переставляем строку с найденным элементом повыше:

3.000\* 2.000 5.000 | 1.000

1.000 2.000 3.000 | 1.000

0.000 1.000 2.000 | 1.000

Нормализуем строку с найденным элементом:

1.000\* 0.667 1.667 | 0.333

1.000 2.000 3.000 | 1.000

0.000 1.000 2.000 | 1.000

Обрабатываем нижележащие строки:

1.000\* 0.667 1.667 | 0.333

0.000 1.333 1.333 | 0.667

0.000 1.000 2.000 | 1.000

Ищем максимальный по модулю элемент в 2-м столбце:

1.000 0.667 1.667 | 0.333

0.000 1.333\* 1.333 | 0.667

0.000 1.000 2.000 | 1.000

Нормализуем строку с найденным элементом:

1.000 0.667 1.667 | 0.333

0.000 1.000\* 1.000 | 0.500

0.000 1.000 2.000 | 1.000

Обрабатываем нижележащие строки:

1.000 0.667 1.667 | 0.333

0.000 1.000\* 1.000 | 0.500

0.000 0.000 1.000 | 0.500

Ищем максимальный по модулю элемент в 3-м столбце:

1.000 0.667 1.667 | 0.333

0.000 1.000 1.000 | 0.500

0.000 0.000 1.000\* | 0.500

Нормализуем строку с найденным элементом:

1.000 0.667 1.667 | 0.333

0.000 1.000 1.000 | 0.500

0.000 0.000 1.000\* | 0.500

Обрабатываем нижележащие строки:

1.000 0.667 1.667 | 0.333

0.000 1.000 1.000 | 0.500

0.000 0.000 1.000\* | 0.500

Матрица приведена к треугольному виду, считаем решение...

Получили ответ:

X1 = -0.50

X2 = 0.00

X3 = 0.50

Определитель: 1.0

Process finished with exit code 0

**Вывод**

По итогу выполнения лабораторной работы научился реализовывать один из прямых методов вычисления СЛАУ, и вследствие этого лучше разобрался в механике его функционирования.

Красные строки, выравнивание по всей ширине документа, автоматические переносы.