Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**­­­­­Лабораторная работа №1**

**по дисциплине «Вычислительная математика»**

**«Решение системы линейных алгебраических**

**уравнений СЛАУ»**

Вариант №3

Группа: P3212

Выполнил: Балин А. А.

Проверила: Наумова Н. А.

# Цель работы

Разработать приложение для решения систем линейных алгебраических уравнений методом Гаусса (прямой и обратный ход)

# Задачи

Необходимо было реализовать:

* Ввод размерности системы уравнений через файл или с консоли
* Ввод матрицы коэффициентов системы линейных уравнений через файл или с консоли
* Вычисление определителя
* Вывод треугольной матрицы с преобразованным столбцом B
* Вывод вектора неизвестных
* Вывод вектора невязок

Дополнительно реализовано:

* Генерация матриц

# Блок-схема реализованного алгоритма

Изображение выглядит как снимок экрана, текст, диаграмма, дизайн

Автоматически созданное описание

# Реализация (Python)

class Matrix:

    def \_\_init\_\_(self, rows):

        self.rows = rows

        self.n = len(rows)

        self.m = len(rows[0])

        if any(len(row) != self.m for row in rows):

            raise ValueError("All rows must have the same length")

        if self.n != self.m - 1:

            raise ValueError("Matrix must be (n x n+1)")

        if any(not isinstance(x, (int, float)) for x in sum(rows, [])):

            raise ValueError("All elements must be numbers")

        self.triangular\_rows = None

        self.permutations = -1

    def triangular\_matrix(self):

        perm = 0

        result = [row[:] for row in self.rows]

        for i in range(self.n):

            if result[i][i] == 0:

                for j in range(i + 1, self.n):

                    if result[j][i] != 0:

                        result[i], result[j] = result[j], result[i]

                        perm+=1

                        break

                else:

                    return -1

            for j in range(i + 1, self.n):

                factor = result[j][i] / result[i][i]

                for k in range(self.n + 1):

                    result[j][k] -= factor \* result[i][k]

        self.triangular\_rows = result

        self.permutations = perm

        return perm

    def determinant(self):

        if self.permutations >=0:

            result = 1

            for i in range(self.n):

                result \*= self.triangular\_rows[i][i]

            return result \* (-1)\*\*self.permutations

        else:

            return 0

    def solve\_system\_gauss(self):

        if self.determinant() == 0:

            raise ValueError()

        solutions = [0] \* self.n

        for i in range(self.n - 1, -1, -1):

            solution = self.triangular\_rows[i][self.n]

            for j in range(i + 1, self.n):

                solution -= self.triangular\_rows[i][j] \* solutions[j]

            solutions[i] = solution / self.triangular\_rows[i][i]

        return solutions

    def \_\_str\_\_(self):

        col\_widths = [max(len(str(elem)) for elem in col) for col in zip(\*self.rows)]

        matrix\_str = "\nMatrix:\n\n"

        for row in self.rows:

            for i, elem in enumerate(row):

                matrix\_str += str(elem).ljust(col\_widths[i]) + " | "

            matrix\_str = matrix\_str[:-3]

            matrix\_str += "\n" + "-" \* (sum(col\_widths) + 3 \* (len(row) - 1)) + "\n"

        return matrix\_str[:-1]

    @staticmethod

    def residual\_vector(data, solution):

        if len(data) != len(solution):

            raise ValueError("Data and solution lengths do not match")

        residual = []

        for i in range(len(data)):

            eq\_residual = data[i][-1]

            for j in range(len(solution)):

                eq\_residual -= data[i][j] \* solution[j]

            residual.append(eq\_residual)

        return residual

    def print\_triangular(self):

        col\_widths = [max(len(str(elem)) for elem in col) for col in zip(\*self.triangular\_rows)]

        matrix\_str = "\nMatrix triangular:\n\n"

        for row in self.triangular\_rows:

            for i, elem in enumerate(row):

                matrix\_str += str(elem).ljust(col\_widths[i]) + " | "

            matrix\_str = matrix\_str[:-3]

            matrix\_str += "\n" + "-" \* (sum(col\_widths) + 3 \* (len(row) - 1)) + "\n"

        return matrix\_str[:-1]

# Полная реализация

<https://github.com/ta4ilka69/docs_for_labs/tree/main/Вычмат/lab1>

# Пример работы алгоритма

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

# Вывод

В ходе реализации данной лабораторной работы я ознакомился с работой метода Гаусса, предназначенного для решения совместных определенных систем линейных алгебраических уравнений (СЛАУ). Данный алгоритм относится к виду прямых, что, к сожалению, не исключает погрешность вычислений чисел с плавающей точкой, что можно видеть по вектору невязок.