МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Поволжский государственный университет телекоммуникаций и информатики»

**Лабораторная работа №2**

**Решение систем линейных алгебраических уравнений. Приближённые методы.**

**В-13**

Выполнил:

Селихов А.С.

Проверила:

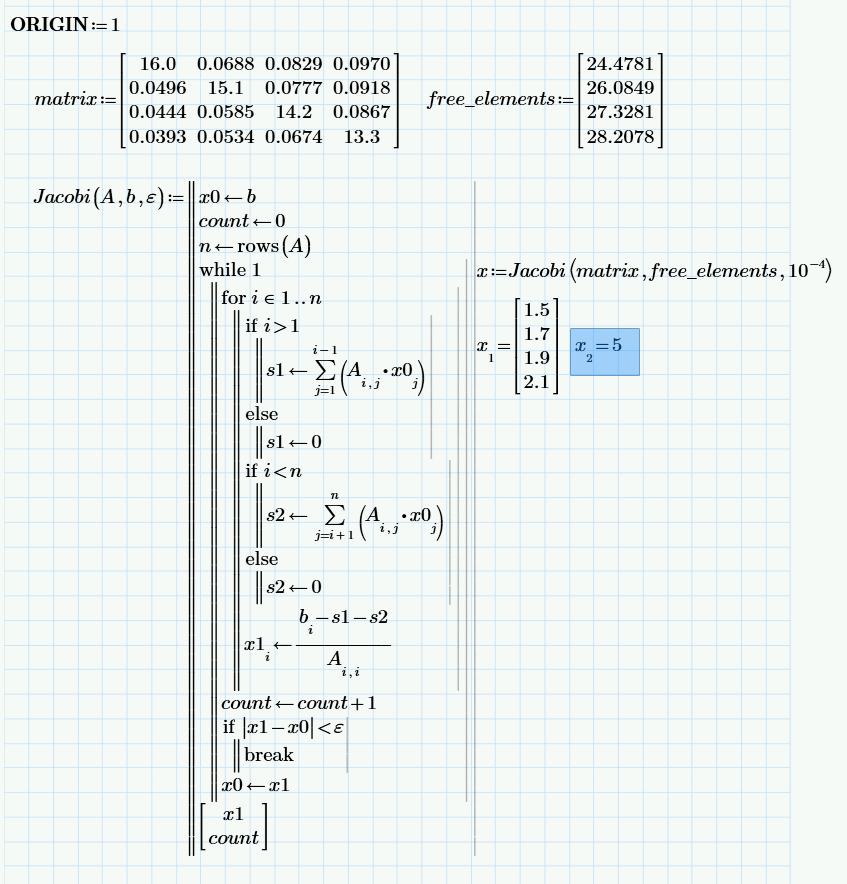
Ахметшина Э.Г.

Самара

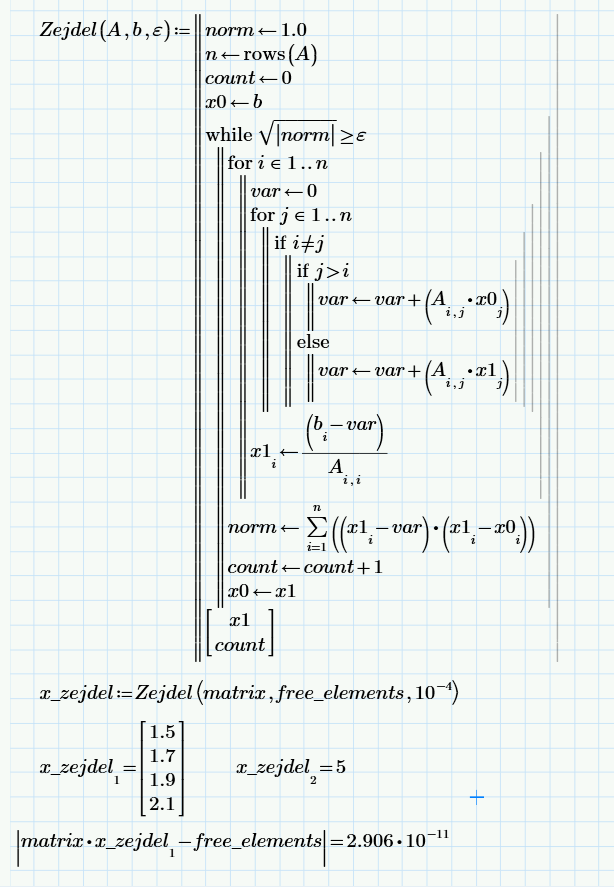
2021

**Цель работы:** Изучить известные методы решения систем линейных алгебраических уравнений итерационными методами.

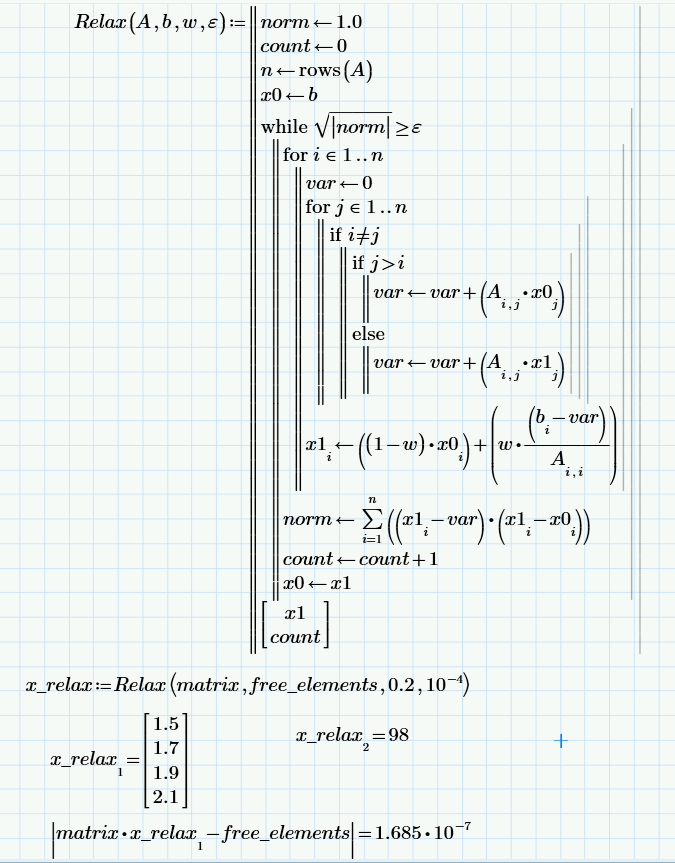
1. Метод Якоби.



1. Метод Зейделя.



1. Обобщённый метод Зейделя ( Метол Релаксации)



1. Код программы

use std::io;

use methods::\*;

fn main() {

    let mut matrix = Vec::<Vec<f64>>::new();

    let mut free\_elements = Vec::<f64>::new();

    let mut size = usize::MIN;

    println!("Ручной ввод?\n(1)Да;\n(2)Нет");

    let mut choice = String::new();

    io::stdin().read\_line(&mut choice).expect("Failed to read");

    let mut inp = 0;

    match choice.trim().parse::<usize>() {

        Ok(v) => {

            inp = v;

        }

        Err(..) => eprintln!("Не подходящий формат: {}", choice.trim()),

    };

    if inp == 0 {

        println!("Введите размерность для квадратной матрицы");

        let mut input\_text = String::new();

        io::stdin()

            .read\_line(&mut input\_text)

            .expect("Failed to read");

        let trimmed = input\_text.trim();

        match trimmed.parse::<usize>() {

            Ok(v) => {

                size = v;

            }

            Err(..) => println!("Не подходящий формат: {}", trimmed),

        };

        for \_i in 0..size {

            matrix.push(vec![0.0; size]);

            free\_elements.push(0.0);

        }

        println!("Введите коэфы и свободные члены: ");

        for i in 0..size {

            for j in 0..size {

                let mut input\_text = String::new();

                io::stdin()

                    .read\_line(&mut input\_text)

                    .expect("Failed to read");

                let trimmed = input\_text.trim();

                match trimmed.parse::<f64>() {

                    Ok(v) => {

                        matrix[i][j] = v;

                    }

                    Err(..) => println!("Не подходящий формат: {}", trimmed),

                };

            }

            let mut input\_text = String::new();

            io::stdin()

                .read\_line(&mut input\_text)

                .expect("Failed to read");

            let trimmed = input\_text.trim();

            match trimmed.parse::<f64>() {

                Ok(v) => {

                    free\_elements[i] = v;

                }

                Err(..) => println!("Не подходящий формат: {}", trimmed),

            };

        }

    } else {

        matrix = vec![

            vec![16.0, 0.0688, 0.0829, 0.0970],

            vec![0.0496, 15.1, 0.0777, 0.0918],

            vec![0.0444, 0.0585, 14.2, 0.0867],

            vec![0.0393, 0.0534, 0.0674, 13.3],

        ];

        free\_elements = vec![24.4781, 26.0849, 27.3281, 28.2078];

        size = 4;

    }

    println!("--------------------------------------------------------------------------------");

    println!("Исходные данные: ");

    println!("");

    for i in 0..size {

        for j in 0..size {

            if j != 0 {

                if matrix[i][j] < 0.0 {

                    print!(" - {}x{}", matrix[i][j].abs(), i);

                } else {

                    print!(" + {}x{}", matrix[i][j], i);

                }

            } else {

                print!(" {}x{}", matrix[i][j], i);

            }

        }

        print!(" = {}\n", free\_elements[i]);

    }

    let answer\_for\_jacobi = jacobi(matrix.clone(), free\_elements.clone());

    let answer\_for\_zejdel = zejdel(matrix.clone(), free\_elements.clone());

    println!("--------------------------------------------------------------------------------");

    println!("Корни системы по Якоби:");

    for i in 0..size {

        print!("x{} = {} ", i, answer\_for\_jacobi.0[i]);

    }

    print!("Кол-во итераций = {}\n", answer\_for\_jacobi.1);

    println!("Проверка");

    for i in 0..size {

        let mut ans = 0.0;

        for j in 0..size {

            ans += matrix[i][j] \* answer\_for\_jacobi.0[j];

        }

        println!("x[{}] = {}", i, ans - free\_elements[i]);

    }

    println!("---------------------------------------------------------------------------------");

    println!("Корни системы по Зейделю:");

    for i in 0..size {

        print!("x{} = {} ", i, answer\_for\_zejdel.0[i]);

    }

    print!("Кол-во итераций = {}\n", answer\_for\_zejdel.1);

    println!("Проверка");

    for i in 0..size {

        let mut ans = 0.0;

        for j in 0..size {

            ans += matrix[i][j] \* answer\_for\_zejdel.0[j];

        }

        println!("x[{}] = {}", i, ans - free\_elements[i]);

    }

    println!("---------------------------------------------------------------------------------");

    println!("Корни системы по обобщённому Зейделю:");

    for relax in [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8].into\_iter() {

        println!(

            "---------------------------------------------------------------------------------"

        );

        println!("Ответ при релаксации = {}", &relax);

        let answer\_for\_zejdelex = zejdelex(matrix.clone(), free\_elements.clone(), relax.clone());

        for i in 0..size {

            print!("x{} = {} ", i, answer\_for\_zejdelex.0[i]);

        }

        print!("Кол-во итераций = {} w={}\n", answer\_for\_zejdelex.1, relax);

        println!("Проверка");

        for i in 0..size {

            let mut ans = 0.0;

            for j in 0..size {

                ans += matrix[i][j] \* answer\_for\_zejdelex.0[j];

            }

            println!("x[{}] = {}", i, ans - free\_elements[i]);

        }

    }

}

pub fn zejdelex(matrix: Vec<Vec<f64>>, free\_elements: Vec<f64>, relax: f64) -> (Vec<f64>, usize)

{

    let size: usize = free\_elements.len();

    let mut answer = vec![0.0; size];

    let mut prev = vec![0.0; size];

    let mut iter\_count = 0;

    let mut norm = f64::MAX;

    while norm >= EPS {

        iter\_count += 1;

        for i in 0..size {

            answer[i] = free\_elements[i];

            for j in 0..size {

                if i != j {

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

                }

            }

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

            answer[i] = relax \* answer[i] + (1.0 - relax) \* prev[i];

        }

        for i in 0..size {

            if (answer[i] - prev[i]).abs() < norm {

                norm = (answer[i] - prev[i]).abs();

            }

            prev[i] = answer[i];

        }

    }

    (answer, iter\_count)

}

pub fn zejdel(matrix: Vec<Vec<f64>>, free\_elements: Vec<f64>) -> (Vec<f64>, usize) {

    let size: usize = free\_elements.len();

    let mut answer = vec![0.0; size];

    let mut prev = vec![0.0; size];

    let mut iter\_count = 0;

    let mut norm = f64::MAX;

    while norm.sqrt() >= EPS {

        norm = 0.0;

        iter\_count += 1;

        for i in 0..size {

            prev[i] = answer[i];

        }

        for i in 0..size {

            let mut var = 0.0;

            for j in 0..size {

                if j != i {

                    var += matrix[i][j] \* answer[j];

                }

            }

            answer[i] = (free\_elements[i] - var) / matrix[i][i];

            norm += (answer[i] - prev[i]).powi(2);

        }

    }

    (answer, iter\_count)

}

pub fn jacobi(matrix: Vec<Vec<f64>>, free\_elements: Vec<f64>) -> (Vec<f64>, usize) {

    let size: usize = free\_elements.len();

    let mut answer: Vec<f64> = vec![1.0; size];

    let mut temp\_answer: Vec<f64> = vec![0.0; size];

    let mut norm = f64::MAX;

    let mut iter\_count = 0;

    while norm >= EPS {

        iter\_count += 1;

        for i in 0..size {

            temp\_answer[i] = 0.0 - free\_elements[i];

            for g in 0..size {

                if i != g {

                    temp\_answer[i] += matrix[i][g] \* answer[g];

                }

            }

            temp\_answer[i] /= 0.0 - matrix[i][i];

        }

        norm = (answer[0] - temp\_answer[0]).abs();

        for h in 0..size {

            if (answer[h] - temp\_answer[h]).abs() > norm {

                norm = (answer[h] - temp\_answer[h]).abs();

            }

            answer[h] = temp\_answer[h];

        }

    }

    (answer, iter\_count)

}

