Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»
Факультет программной инженерии и компьютерной техники

Вариант №13
Лабораторная работа №1
по дисциплине
Вычислительная математика

Выполнил студент группы P3212
Соколов Анатолий Владимирович
Преподаватель:
Наумова Надежда Александровна

г. Санкт-Петербург
2024г.

Содержание

1 Задание

В программе реализуемый численный метод решения системы линейных алгебраических уравнений (СЛАУ) должен быть реализован в виде отдельного класса /метода/функции, в который исходные/выходные данные передаются в качестве параметров.
Задавать размерность матрицы ($n < 20$) из файла или с клавиатуры – по выбору конечного пользователя.
Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).
Сформировать не менее 3 файлов (тестов) с различным набором данных.
Программа должна быть протестирована на различных наборах данных, в том числе и некорректных.

1.1 Вариант

Для итерационных методов должно быть реализовано:

- Точность задается с клавиатуры/файла
- Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
- Вывод вектора неизвестных: $x1, x2, \ldots , x_n$
- Вывод количества итераций, за которое было найдено решение.
- Вывод вектора погрешностей: $|x_i^K - x_i^{k-1}|$

Метод	№ варианта
Метод Гаусса	1, 3, 5, 8, 21, 24, 26, 28, 31, 39
Метод Гаусса с выбором главного элемента по столбцам	11, 17, 19, 22, 25, 27, 30, 34, 40
Метод простых итераций	2, 4, 6, 7, 10, 13, 15, 23, 32, 36, 38
Метод Гаусса-Зейделя	9, 12, 14, 16, 18, 20, 29, 33, 35, 37

1.2 Цель работы

Рассчитать систему линейных алгебраических уравнений СЛАУ и изучить МПИ.

1.3 Описание метода, расчетные формулы

Описание метода

Итерационные методы это методы последовательных приближений.

Задается некоторое начальное приближение. Далее с помощью определенного алгоритма проводится один цикл вычислений - итерация. В результате итерации находят новое приближение. Итерации проводятся до получения решения с требуемой точностью.

Метод простых итераций — это численный метод решения уравнений, включая алгебраические и дифференциальные уравнения. Основная идея метода состоит в том, чтобы преобразовать исходное уравнение таким образом, чтобы решение могло быть представлено как предел последовательности, получаемой итеративным способом.

Для алгебраических уравнений метод простых итераций можно описать следующим образом:

1. Преобразование уравнения: Исходное уравнение $f(x) = 0$ преобразуется в эквивалентное уравнение вида $x = \phi(x)$, где функция $\phi(x)$ должна быть выбрана таким образом, чтобы последовательность, порождаемая этой функцией, сходилась к корню исходного уравнения.

2. Выбор начального приближения: Выбирается начальное приближение к корню уравнения, обозначаемое как x_0.

3. Итерационный процесс: На каждом шаге вычисляется следующее приближение к корню по формуле $x_{n+1} = \phi(x_n)$, где n — номер текущей итерации. Этот процесс повторяется до тех пор, пока разность между последовательными приближениями не станет меньше заданной точности ε, т.е. $|x_{n+1} - x_n| < \varepsilon$.

4. Критерий остановки: Итерационный процесс продолжается до тех пор, пока не будет достигнута заданная точность или не будет выполнено максимально допустимое количество итераций.

Расчетные формулы

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \sum_{j=1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j^k, i = 1, 2, \ldots, n$$

$$c_{ij} = \begin{cases} 0, i = j \\ -\frac{a_{ij}}{a_{ii}} \end{cases}$$

$$d_i = \frac{b_i}{a_{ii}}, i = 1, 2, \ldots, n$$

$$x^{k+1} = Cx^k + d$$

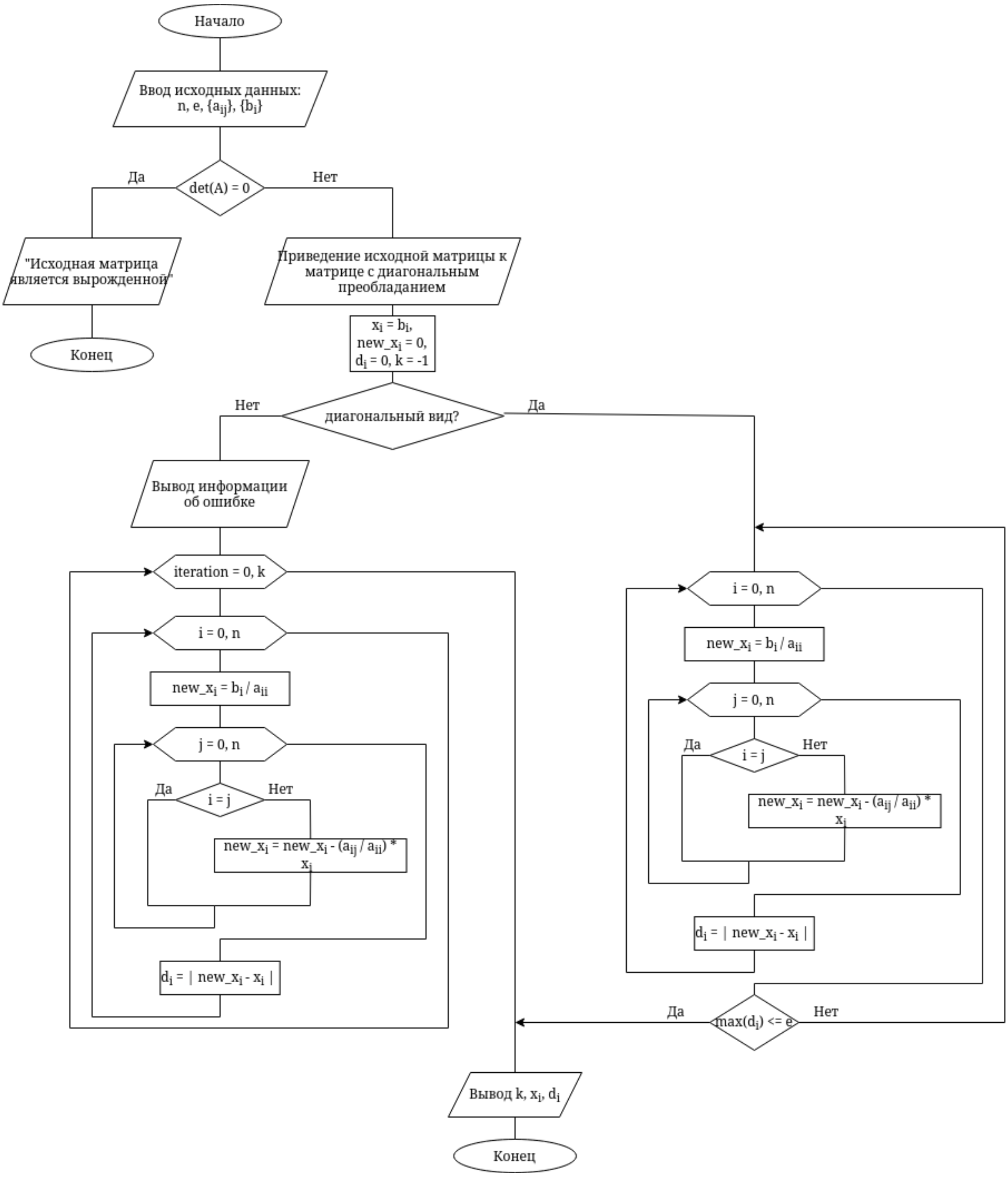
1.4 Реализация численного метода

```
fn shuffle(&mut self) -> (bool, Vec<Vec<f64>>) {
    let mut biggest = vec![-1; self.n];
    let mut biggest_set = HashSet::new();
    let mut found_strict = false;

    for i in 0..self.n {
        let sum: f64 = self.a[i].iter().sum();
        for j in 0..self.n {
            if 2.0 * self.a[i][j] >= sum {
            if 2.0 * self.a[i][j] > sum {
                found_strict = true;
            }
            biggest[i] = j as isize;
            biggest_set.insert(j);
            break;
            }
        }
        if biggest[i] == -1 {
            return (false, vec![vec![0.0]]);
        }
    }

    if !found_strict || biggest.len() != biggest_set.len() {
        return (false, vec![vec![0.0]]);
```

```rust
    }

    let mut shuffled_a = vec![vec![]; self.n];
    let mut shuffled_b = vec![0.0; self.n];

    for i in 0..self.n {
        let index = biggest[i] as usize;
        shuffled_a[index] = self.a[i].clone();
        shuffled_b[index] = self.b[i];
    }

    self.a = shuffled_a.clone();
    self.b = shuffled_b;

    (true, shuffled_a)
    }

fn find_c_and_d(coefficients: Vec<Vec<f64>>) -> Vec<Vec<f64>> {
    let n = coefficients.len(); // The number of rows, assuming a square matrix for coefficients
    let mut c = vec![vec![0.0; n]; n]; // Initialize C matrix with zeros

    for i in 0..n {
        // Diagonal element of the current row
        let diag_elem = coefficients[i][i];

        for j in 0..n {
            // Check if the current element is not on the diagonal
            if i != j {
            // C matrix is -1 times the original coefficient matrix divided by the diagonal element
            c[i][j] = -coefficients[i][j] / diag_elem;
            }
        }
        // The diagonal elements of C are set to zero
        c[i][i] = 0.0;
    }

    c
    }

fn iterate(&mut self) {
    let mut new_sol = vec![0.0; self.n];
    for i in 0..self.n {
        new_sol[i] = self.b[i] / self.a[i][i] - self.sum_sol_row(i);
        self.sol_acc[i] = (new_sol[i] - self.sol[i]).abs();
    }
    self.sol = new_sol;
    self.sol_iter += 1;
    }

    pub fn solve(&mut self) -> Json<serde_json::Value> {
    let mut err = String::new();

    if !self.shuffle().0 {
        err = String::from("Невозможно привестикдиагональномупреобладанию.")
    }

    self.shuffled_matrix = self.shuffle().1;

```

```
    while self.sol_acc.iter().max_by(|a, b| a.total_cmp(b)).unwrap() > &
    ↪ self.acc
        && self.sol_iter < self.max_iter
    {
        self.iterate();
    }

    if !self.shuffled_matrix.is_empty() {
        self.c = Matrix::find_c_and_d(self.shuffle().1);
        return Json(json!({
            "sol": self.sol,
            "acc": self.sol_acc,
            "iter": self.sol_iter,
            "c": self.c,
            "mtrx": self.shuffled_matrix,
            "err": err,
        }));
    }

    Json(json!({
        "sol": self.sol,
        "acc": self.sol_acc,
        "iter": self.sol_iter,
        "mtrx": self.shuffled_matrix,
        "err": err,
    }))
    }
```

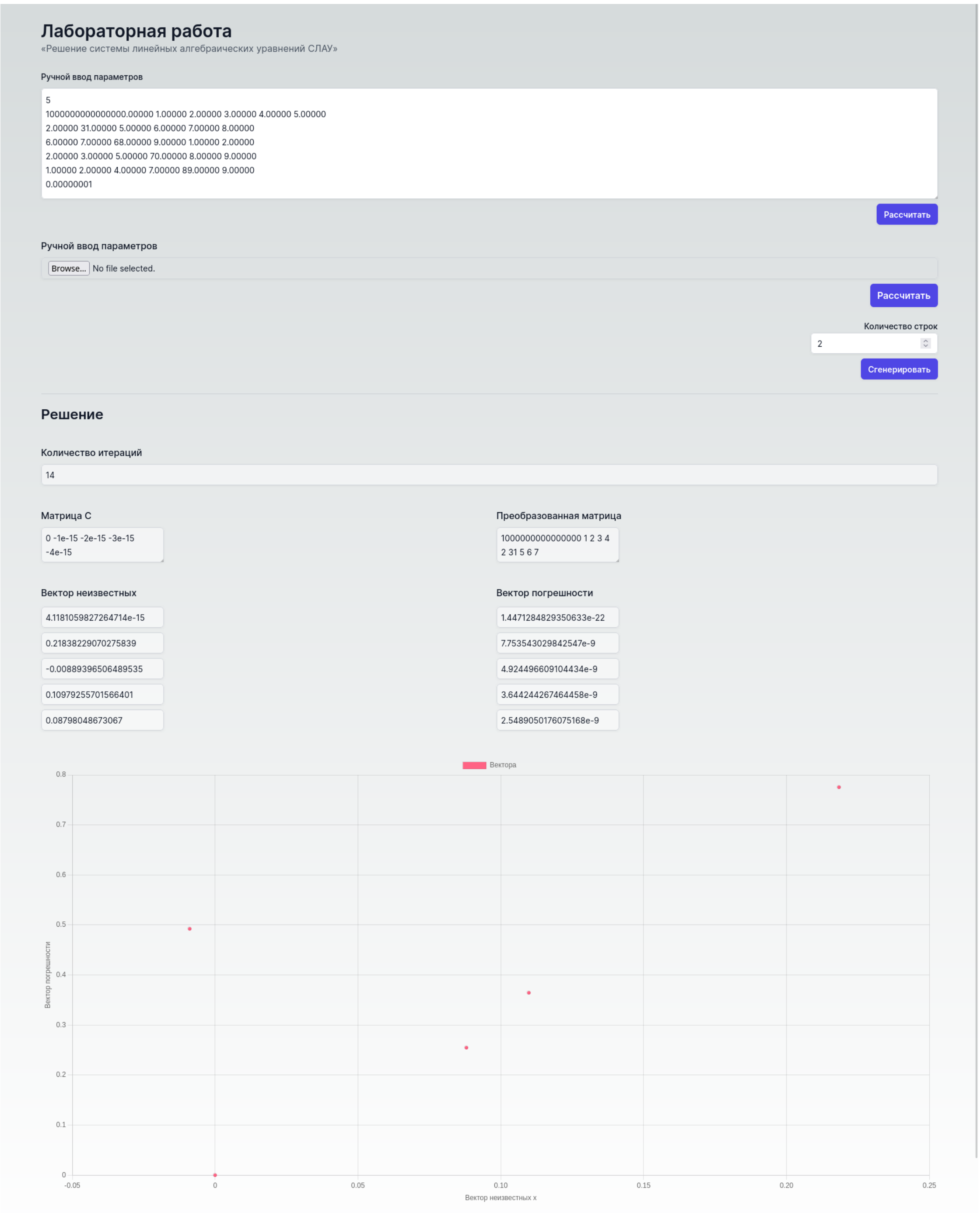
1.5 Блок-схема реализованного алгоритма

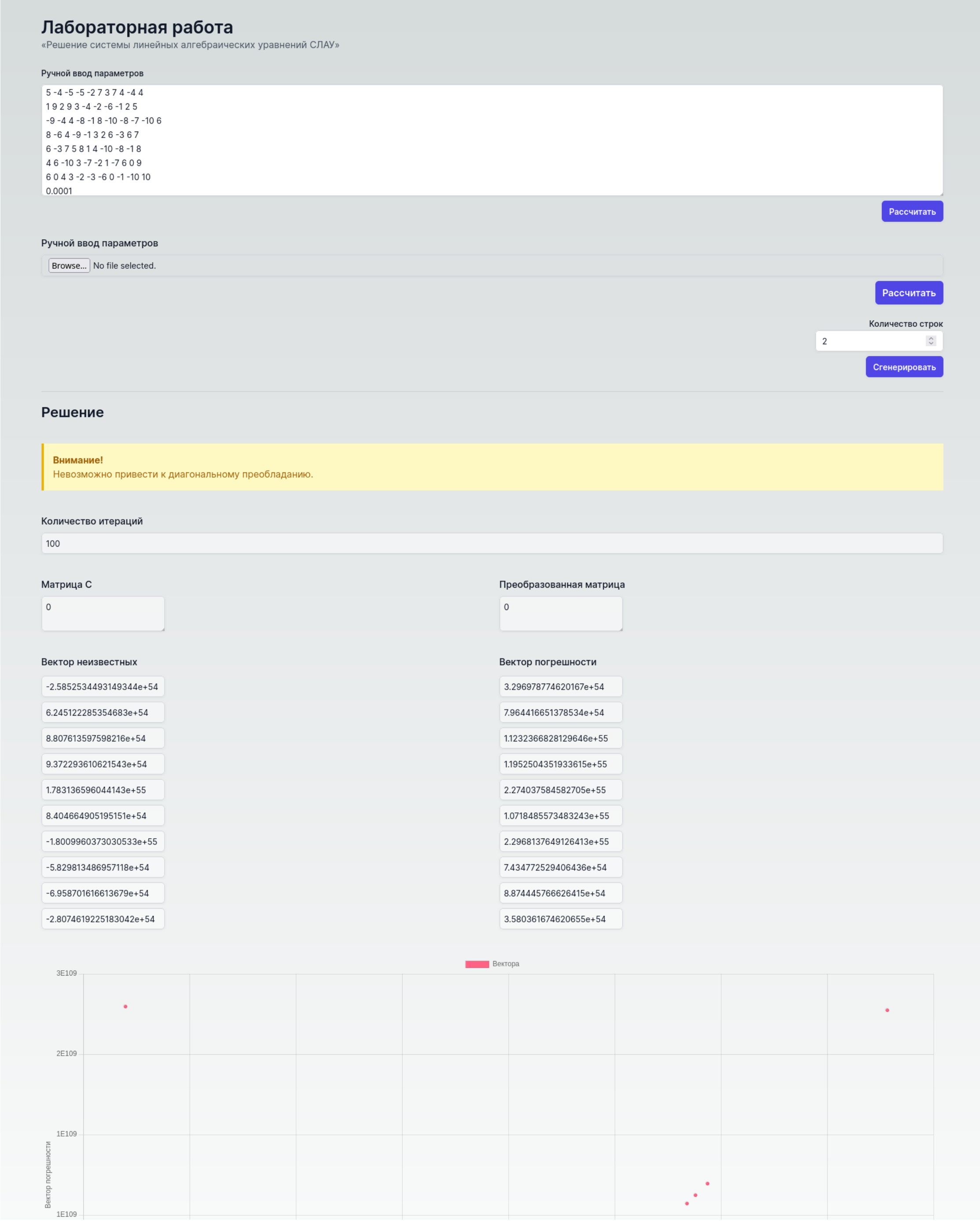


1.6 Ссылка на GitHub с основной реализацией

Github

1.7 Примеры и результаты работы программы





2 Заключение

Я познакомился с новым для меня и крайне необыкновенным вычислением СЛАУ на языке rust.

3 Список литературы

[1] Слайды с лекций (2023). // Кафедра информатики и вычислительной техники – Малышева Татьяна Алексеевна, к.т.н., доцент.