

ВЫЧМАТ ОТЧЁТ ЛАБ1

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»

Вариант №7

Выполнил: Студент группы Р3213
Молчанов Фёдор Денисович

Проверила: Машина Е.А.

г. Санкт-Петербург

2025 г.

1. Цель работы

Целью данной лабораторной работы является изучение и реализация итерационного метода для решения систем линейных алгебраических уравнений (СЛАУ). В работе используется метод простых итераций (метод Якоби) с предварительной проверкой и приведением матрицы коэффициентов к диагональному преобладанию. Программа реализована на языке C++.

2. Описание метода, расчетные формулы

Итерационные методы решения СЛАУ позволяют получить последовательность приближённых решений $x(0)$, $x(1)$, ..., $x(k)$, которая сходится к точному решению x^* . Метод Якоби основан на следующей рабочей формуле:

$$x(k+1)_i = (1/a_{ii}) * (b_i - \sum_{j=1, j \neq i}^n a_{ij} * x(k)_j), i = 1, \dots, n$$

где:

a_{ij} — элементы матрицы коэффициентов,

b_i — элементы вектора свободных членов,

$x(k)_i$ — приближение решения на k -ой итерации.

Критерием остановки является достижение требуемой точности:

$$\max |x(k+1)_i - x(k)_i| < \varepsilon$$

Перед началом итерационного процесса проводится проверка матрицы на диагональное преобладание. Если матрица не обладает диагональным преобладанием, выполняется попытка перестановки строк.

3. Листинг программы

```
void MatrixProcessor::runJacobiMethod(double epsilon, int matrixSize) {
    mainWindowRef.resultsWidget->addText("Running Jacobi method...");

    // Retrieve matrix and RHS values
    QVector<QVector<double>> matrix = matrixWidgetRef.getMatrixValues();
    QVector<double> rhs = matrixWidgetRef.getRHSValues();

    // Ensure matrix is square and valid
    if (matrix.size() != matrixSize || rhs.size() != matrixSize) {
        mainWindowRef.resultsWidget->setResultText("Matrix or RHS size mismatch.");
        return;
    }

    qDebug() << "Matrix in jacobi method:";
    matrixWidgetRef.debugPrintDoubleArray(matrix, matrix.size());

    // Ensure matrix is diagonally dominant
    makeDiagDominant(matrix, rhs, matrixSize);
    qDebug() << "Diagonalised matrix:";
    matrixWidgetRef.debugPrintDoubleArray(matrix, matrix.size());
    mainWindowRef.resultsWidget->addText("The norm of the diagonal matrix is: " + QString::number(matrixNorm(matrix)));

    // Initial guess (e.g., all zeros)
    QVector<double> x(matrixSize, 0.0);
    QVector<double> newX(matrixSize, 0.0);

    int iteration = 0;
    double error = epsilon + 1.0; // To start the loop
    QString resultTable =
        "| k | x1^k | x2^k | ..... | max diff |\n"
        "| --- | - - - - - | - - - - - | - - - - - | - - - - - |\n";

    // Run the iteration loop
    while (error >= epsilon) {
        error = 0.0;
```

```
double maxDiff = 0.0;

// Start iteration
for (int i = 0; i < matrixSize; ++i) {
    double sum = rhs[i];

    for (int j = 0; j < matrixSize; ++j) {
        if (i != j) {
            sum -= matrix[i][j] * x[j];
        }
    }

    newX[i] = sum / matrix[i][i];
}

// Calculate the maximum difference between new and old x values
for (int i = 0; i < matrixSize; ++i) {
    double diff = std::fabs(newX[i] - x[i]);
    maxDiff = std::max(maxDiff, diff);
    error += std::pow(newX[i] - x[i], 2);
}

error = std::sqrt(error); // Root sum squared error
x = newX; // Update the solution

// Append iteration results to the result table
resultTable += "| " + QString::number(iteration) + " | ";
for (int i = 0; i < matrixSize; ++i) {
    resultTable += QString::number(x[i], 'f', 6) + " | ";
}

if (maxDiff == x[x.size()-1])
    resultTable += " ----- |\n";
else
    resultTable += QString::number(maxDiff, 'f', 6) + " |\n";

++iteration;

qDebug() << "Iteration" << iteration << "Error" << error << "Max
Diff" << maxDiff;
}

// Final result message
QString result = "Jacobi Method Converged in " +
QString::number(iteration) + " iterations.\n";
for (int i = 0; i < matrixSize; ++i) {
    result += "X" + QString::number(i+1) + " = " +
QString::number(x[i]) + "\n";
}
```

```
// Update the results widget with the final table and result
mainWindowRef.resultsWidget->addText(resultTable);
mainWindowRef.resultsWidget->addText(result);
}
```

4. пример выполнения программы

Running Jacobi method...

The norm of the diagonal matrix is: 4

k	x1^k	x2^k	max diff
0	1.200000	1.300000	1.400000	-----
1	0.930000	0.920000	0.900000	0.500000
2	1.018000	1.024000	1.030000	0.130000
3	0.994600	0.993400	0.991600	0.038400
4	1.001500	1.001920	1.002400	0.010800
5	0.999568	0.999460	0.999316	0.003084

Jacobi Method Converged in 6 iterations.

X1 = 0.999568

X2 = 0.99946

X3 = 0.999316

5. Выводы

В ходе выполнения работы был реализован метод Якоби на C++. Основные выводы:

- Метод позволяет решать СЛАУ при диагональном преобладании матрицы.

- Если матрица не обладает диагональным преобладанием, можно попытаться переставить строки.
- Метод прост в реализации, но сходимость зависит от условий матрицы.

Заключение: были изучены основы итерационных методов решения СЛАУ, реализована программа на C++ и получены практические навыки работы с численными методами.