«Национальный исследовательский университет ИТМО»
Мегафакультет компьютерных технологий и управления
Факультет программной инженерии и компьютерной техники
Образовательная программа: «Компьютерные технологии в дизайне»

Отчет по лабораторной работе №1
по курсу «Вычислительная математика»
Тема: метод Гауса-Зейделя

Выполнили:
Савин Денис
Егорова Софья

Проверила:
Машина Е.А.

Санкт-Петербург
2025 г.

Содержание отчета

Оглавление

Цель работы

Целью данной работы является изучение итерационного метода Гаусса-Зейделя для решения систем линейных алгебраических уравнений. Реализация метода в программе и анализ его сходимости. В частности, необходимо:

- Изучить теоретические основы метода, включая анализ условий сходимости (например, необходимость диагонального преобладания или положительной определённости матрицы).
- Реализовать алгоритм метода в программном виде, организовать последовательность итерационных приближений с контролем точности (заданной параметром ε).
- Провести вычислительный эксперимент для конкретной системы уравнений, оценить число итераций до достижения требуемой точности.

Описание метода и расчетные формулы

Метод Гаусса-Зейделя является модификацией метода простой итерации и обеспечивает более быструю сходимость к решению систем уравнений.

Так же как и в методе простых итераций строится эквивалентная

СЛАУ и за начальное приближение принимается вектор правых частей (как правило, но может быть выбран и нулевой, и единичный вектор): $\mathrm{x}i^{0} = (\mathrm{d}1, \mathrm{d}2, \ldots, \mathrm{dn})$.

Рабочая формула метода Гауса-Зейделя:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

Условие сходимости: матрица коэффициентов должна иметь диагональное преобладание.

Итерационный процесс продолжается до выполнения условия точности:

$$\left|x_i^{(k+1)} - x_i^{(k)}\right| < \varepsilon$$

Листинг программы

```
private double[,] A;
private double[] b;

// Метод решения системы линейных уравнений методом Гаусса-Зейделя
public (double[] solution, int iterations) Gauss_Seidel_Count()
{
    // Читаем размерность матрицы
    int n = int.Parse(Size);
    // Читаем точность вычислений
    double epsilon = double.Parse(Accuracy);
    // Читаем максимальное количество итераций
    int M = int.Parse(MaxCountOfIter);

    // Инициализируем матрицу коэффициентов A
    A = new double[n, n];
    // Инициализируем вектор правой части b
    b = new double[n];

    // Заполняем вектор b значениями из RightVector
    for (int i = 0; i < n; i++)
    {
        b[i] = double.Parse(RightVector[i]);
    }

    // Заполняем матрицу A из списка Coefficients
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            int index = i * n + j; // Преобразуем индексы двумерного массива в
одномерный
            A[i, j] = (index < Coefficients.Count) ? Coefficients[index].CoeffA : 0;
        }
    }

    // Проверяем наличие диагонального преобладания
    if (!CheckDiagonalDominance(A))
    {
        MessageBox.Show("Матрица не имеет диагонального преобладания.
Попытка перестановки строк...");
        // Если невозможно достичь диагонального преобладания, завершаем
выполнение
        if (!MakeDiagonalDominance(A, b))
        {
            MessageBox.Show("Невозможно достичь диагонального
```

```
преобладания.");
                //return (null, 0);
            }
        }

        // Преобразуем систему к виду x = Cx + d
        double[,] C = new double[n, n];
        double[] d = new double[n];

        for (int i = 0; i < n; i++)
        {
            // Вычисляем вектор d
            d[i] = b[i] / A[i, i];
            for (int j = 0; j < n; j++)
            {
                if (i == j)
                    C[i, j] = 0; // Диагональные элементы C равны нулю
                else
                    C[i, j] = -A[i, j] / A[i, i]; // Остальные элементы C
            }
        }

        foreach (var item in A)
        {
            Debug.WriteLine(item);
        }

        foreach (var item in b)
        {
            Debug.WriteLine(item);
        }

        // Вычисляем норму матрицы C
        double normC = CalculateMatrixNorm(C);
        Debug.WriteLine($"Норма матрицы C: {normC}");
        Norma = normC.ToString();

        // Инициализируем начальное приближение
        double[] x = new double[n];
        for (int i = 0; i < n; i++)
        {
            x[i] = b[i] / A[i, i];
        }

        // Создаем массив для хранения предыдущих значений x
        double[] prevX = new double[n];
        Array.Copy(x, prevX, n);
```

```
            int k;
            for (k = 0; k < M; k++) // Итерационный процесс
            {
                for (int i = 0; i < n; i++)
                {
                    double sum = 0;
                    for (int j = 0; j < n; j++)
                    {
                        // Сумма произведений элементов матрицы C и значений x
                        sum += C[i, j] * (j < i ? x[j] : prevX[j]);
                    }
                    // Вычисляем новое значение x
                    x[i] = d[i] + sum;
                }

                // Вычисляем вектор погрешностей
                double[] error = new double[n];
                for (int i = 0; i < n; i++)
                {
                    error[i] = Math.Abs(x[i] - prevX[i]);
                }

                // Выводим результаты итерации в Debug
                Debug.WriteLine($"Итерация {k + 1}:");
                Debug.WriteLine($"x1 = {x[0]}, x2 = {x[1]}, x3 = {x[2]}");
                Debug.WriteLine($"Погрешности: e1 = {error[0]}, e2 = {error[1]}, e3 =
{error[2]}");

                // Записываем результаты для отображения в UI
                Vector = $"x1 = {x[0]}, x2 = {x[1]}, x3 = {x[2]}";
                CountOfIter = (k + 1).ToString();
                VectorPogr = $"e1 = {error[0]}, e2 = {error[1]}, e3 = {error[2]}";

                // Проверка сходимости
                double maxDiff = error.Max();
                if (maxDiff < epsilon)
                    break;

                // Обновляем prevX для следующей итерации
                Array.Copy(x, prevX, n);
            }

            return (x, k + 1);
        }
```

```
        // Метод проверки диагонального преобладания матрицы
        private bool CheckDiagonalDominance(double[,] A)
        {
            int n = A.GetLength(0); // Получаем размерность матрицы (количество
строк)
            for (int i = 0; i < n; i++) // Проходим по каждой строке
            {
                double sum = 0; // Инициализируем сумму элементов строки (кроме
диагонального)
                for (int j = 0; j < n; j++) // Проходим по каждому элементу в строке
                {
                    if (i != j) // Пропускаем диагональный элемент
                        sum += Math.Abs(A[i, j]); // Добавляем модуль элемента к сумме
                }
                if (Math.Abs(A[i, i]) <= sum) // Проверяем условие диагонального
преобладания
                    return false; // Если нарушено, возвращаем false
            }
            return true; // Если проверка пройдена для всех строк, возвращаем true
        }

        // Метод перестановки строк для достижения диагонального преобладания
        private bool MakeDiagonalDominance(double[,] A, double[] b)
        {
            int n = A.GetLength(0); // Получаем размерность матрицы
            bool[] usedRows = new bool[n]; // Создаём массив для отслеживания
использованных строк
            double[,] newA = new double[n, n]; // Создаём новую матрицу для
переставленных строк
            double[] newB = new double[n]; // Создаём новый вектор b

            for (int i = 0; i < n; i++) // Проходим по каждой строке
            {
                int bestRow = -1; // Переменная для хранения индекса строки с
наибольшим диагональным элементом
                double maxDiagonal = 0; // Переменная для хранения максимального
диагонального элемента

                for (int j = 0; j < n; j++) // Ищем строку с наибольшим диагональным
элементом
                {
                    if (!usedRows[j] && Math.Abs(A[j, i]) > maxDiagonal) // Проверяем, не
использована ли строка и является ли диагональный элемент максимальным
                    {
                        maxDiagonal = Math.Abs(A[j, i]); // Обновляем максимальный
диагональный элемент
                        bestRow = j; // Запоминаем индекс строки
```

```
                }
            }

            if (bestRow == -1 || maxDiagonal == 0) // Если не удалось найти подходящую строку
                return false; // Возвращаем false (невозможно достичь диагонального преобладания)

            usedRows[bestRow] = true; // Помечаем строку как использованную

            for (int j = 0; j < n; j++) // Копируем строку в новую матрицу
                newA[i, j] = A[bestRow, j];

            newB[i] = b[bestRow]; // Копируем соответствующий элемент вектора b
        }

        Array.Copy(newA, A, A.Length); // Копируем переставленную матрицу обратно в A
        Array.Copy(newB, b, b.Length); // Копируем переставленный вектор обратно в b

        return CheckDiagonalDominance(A); // Проверяем, достигнуто ли диагональное преобладание
    }

    // Метод вычисления нормы матрицы (максимальной суммы элементов в строках)
    private double CalculateMatrixNorm(double[,] A)
    {
        int n = A.GetLength(0); // Получаем размерность матрицы
        double norm = 0; // Инициализируем переменную для хранения нормы
        for (int i = 0; i < n; i++) // Проходим по каждой строке
        {
            double rowSum = 0; // Переменная для хранения суммы элементов строки
            for (int j = 0; j < n; j++) // Считаем сумму модулей элементов строки
            {
                rowSum += Math.Abs(A[i, j]);
            }
            norm = Math.Max(norm, rowSum); // Обновляем норму, если текущая сумма больше
        }
        return norm; // Возвращаем вычисленную норму
    }
```

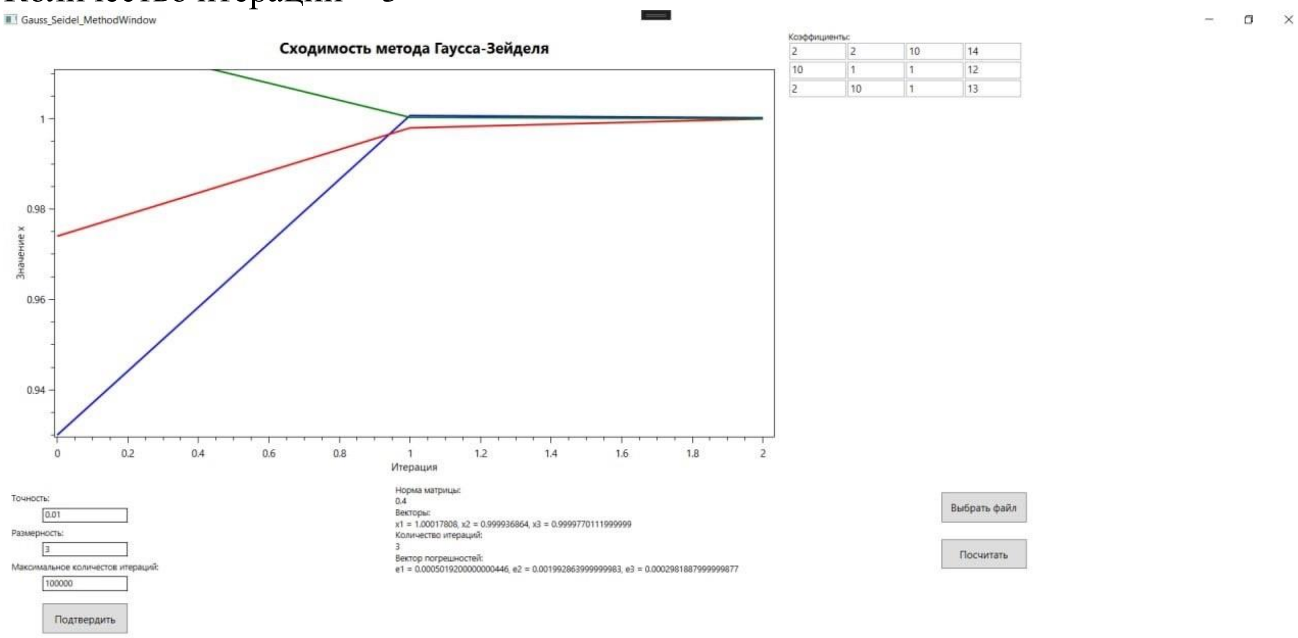
Примеры и результаты программы

1)
2 2 10 = 14
10 1 1 = 12
2 10 1 = 13

точность = 0.01

Норма матрицы C: 0.4
Итерация 1:
x1 = 0.9299999999999999, x2 = 0.9740000000000001, x3 = 1.0191999999999999
Погрешности: e1 = 0.27, e2 = 0.32599999999999996, e3 = 0.3808
Итерация 2:
x1 = 1.00068, x2 = 0.997944, x3 = 1.0002752
Погрешности: e1 = 0.07068000000000008, e2 = 0.023943999999999965, e3 = 0.018924799999999964
Итерация 3:
x1 = 1.00017808, x2 = 0.999936864, x3 = 0.9999770111999999
Погрешности: e1 = 0.0005019200000000446, e2 = 0.001992863999999983, e3 = 0.0002981887999999877

Количество итераций = 3



---

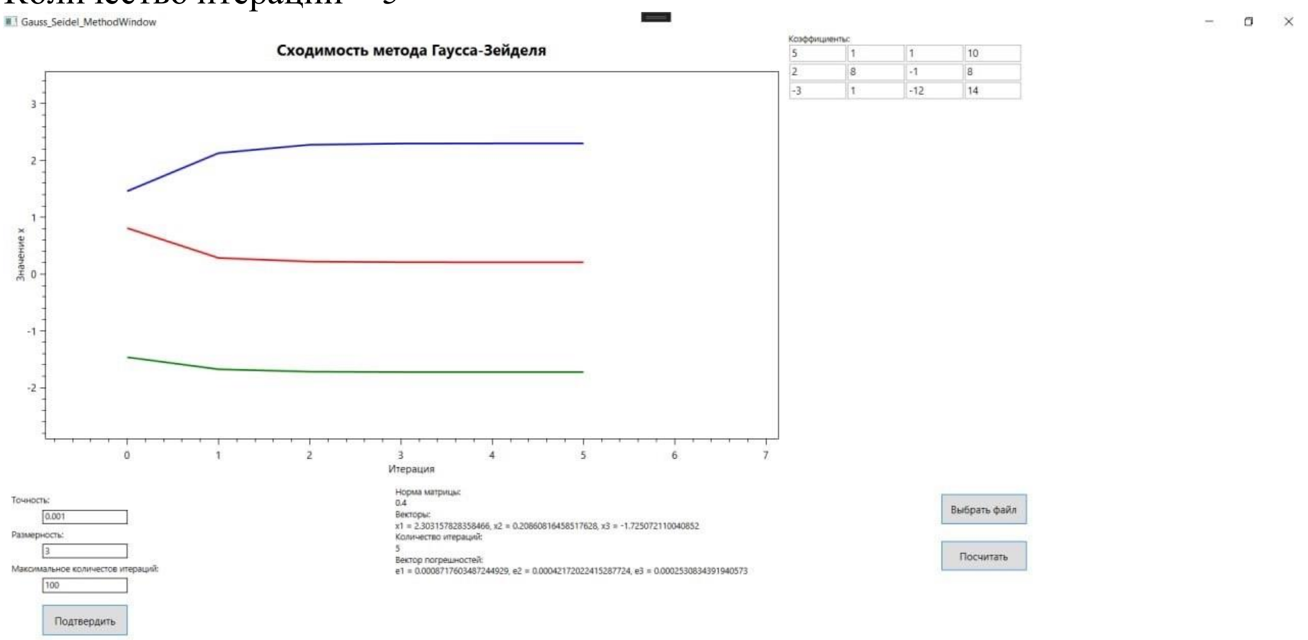
2)
5 1 1 = 10
2 8 -1 = 8
-3 1 -12 = 14

точность = 0.001

Норма матрицы C: 0.4
Итерация 1:

x1 = 2.0333333333333333, x2 = 0.3458333333333333, x3 = -1.6461805555555555
Погрешности: e1 = 0.033333333333333215, e2 = 0.6541666666666667, e3 = 0.4795138888888888
Итерация 2:
x1 = 2.2600694444444445, x2 = 0.2292100694444444, x3 = -1.7125831886574074
Погрешности: e1 = 0.22673611111111125, e2 = 0.11662326388888888, e3 = 0.0664026331018519
Итерация 3:
x1 = 2.2966746238425926, x2 = 0.211758445457175935, x3 = -1.7231887855058836
Погрешности: e1 = 0.03660517939814811, e2 = 0.017451623987268516, e3 = 0.010605596848476173
Итерация 4:
x1 = 2.3022860680097414, x2 = 0.20902988480932916, x3 = -1.7248190266016579
Погрешности: e1 = 0.005611444167148871, e2 = 0.0027285606478467672, e3 = 0.0016302410957742541
Итерация 5:
x1 = 2.303157828358466, x2 = 0.20860816458517628, x3 = -1.725072110040852
Погрешности: e1 = 0.0008717603487244929, e2 = 0.00042172022415287724, e3 = 0.00025308343919405 73

Количество итераций = 5



---

3)

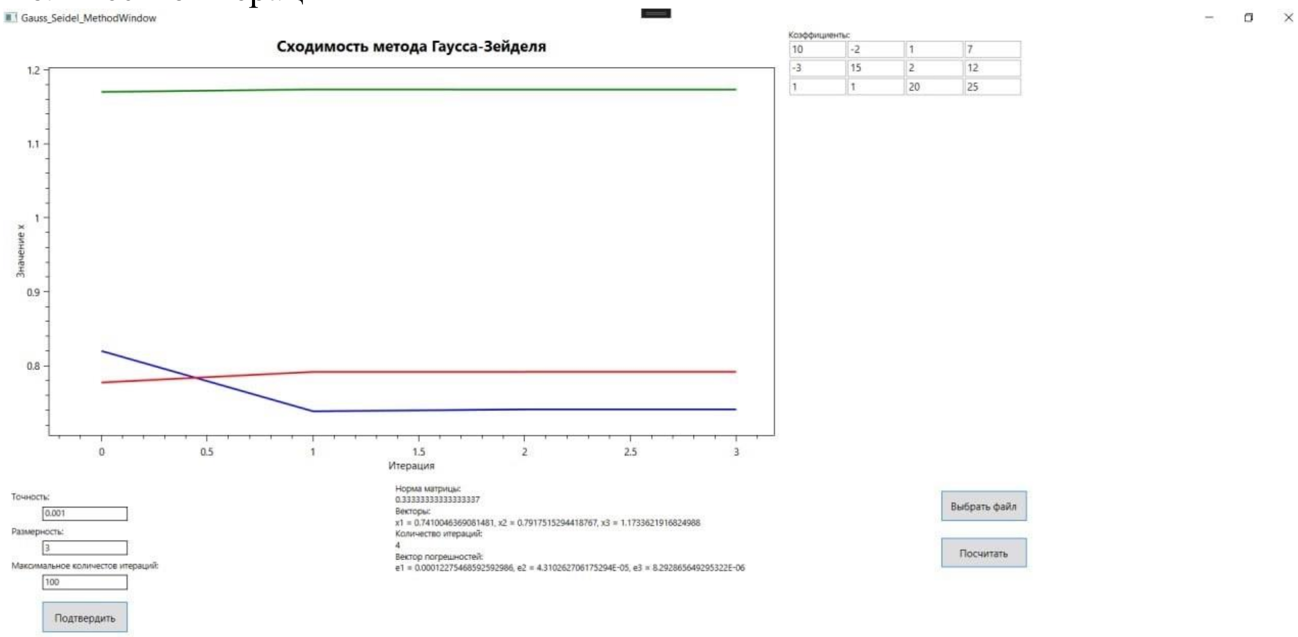
10 -2 1= 7
-3 15 2 = 12
1 1 20 = 15

точность = 0.001

Норма матрицы C: 0.33333333333333337
Итерация 1:

x1 = 0.735, x2 = 0.7803333333333333, x3 = 1.1742333333333332
Погрешности: e1 = 0.035000000000000003, e2 = 0.019666666666666672, e3 = 0.07576666666666676
Итерация 2:
x1 = 0.7386433333333333, x2 = 0.7911642222222223, x3 = 1.1735096222222223
Погрешности: e1 = 0.0036433333333333318, e2 = 0.010830888888888945, e3 = 0.000723711111110914
Итерация 3:
x1 = 0.7408818822222222, x2 = 0.7917084268148149, x3 = 1.173370484548148
Погрешности: e1 = 0.0022385488888888583, e2 = 0.0005442045925926342, e3 = 0.00013913767407425226
Итерация 4:
x1 = 0.7410046369081481, x2 = 0.7917515294418767, x3 = 1.1733621916824988
Погрешности: e1 = 0.00012275468592592986, e2 = 4.310262706175294E-05, e3 = 8.292865649295322E-06

Количество итераций = 4



Выводы

После выполнения лабораторной работы по методу Гаусса–Зейделя можно сделать следующие выводы:

- Применение итерационного метода позволило получить приближённое решение системы линейных уравнений с заданной точностью.
- Экспериментально установлено, что при соблюдении условий сходимости (наличие диагонального преобладания) метод сходится достаточно быстро – число итераций для достижения заданной точности оказалось приемлемым.
- Метод Гаусса–Зейделя продемонстрировал свою практическую ценность для решения СЛАУ, особенно в случаях, когда матрица системы удовлетворяет необходимым условиям.

Таким образом, проведённая работа не только подтвердил эффективность метода Гаусса–Зейделя для решения СЛАУ, но и выявила ключевые моменты, влияющие на скорость сходимости и точность решения, что является важным аспектом при выборе численного метода для решения практических задач.