Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2
Решение СЛАУ

по курсу «Вычислительной математики»

Выполнил:
Студент группы P3230
Пономаренко Алиса Валерьевна

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2024

Описание численного метода

Итерационные методы: Метод Гаусса-Зейделя

Метод Гаусса-Зейделя — это итерационный метод решения систем линейных уравнений, который является модификацией метода Якоби.

Пусть у нас есть набор данных:
Квадратная матрица коэффициентов A:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Вектор b:

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Приближение epsilon ϵ.

Решением СЛАУ является такой вектор , что $AX = b$.

Вектор X:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Основная идея метода Гаусса-Зейделя заключается в следующем:

Начинаем с начального приближения к решению системы уравнений. Используем текущее приближение для вычисления новых значений неизвестных. Обновляем приближение к решению и повторяем процесс, пока не будет достигнута заданная точность epsilon или не будет достигнуто максимальное количество итераций.

Описание алгоритма: Сначала проверим исходную матрицу на диагональное преобладание. Если условие не выполняется - пробуем перестановками строк и столбцов добиться диагонального преобладания.

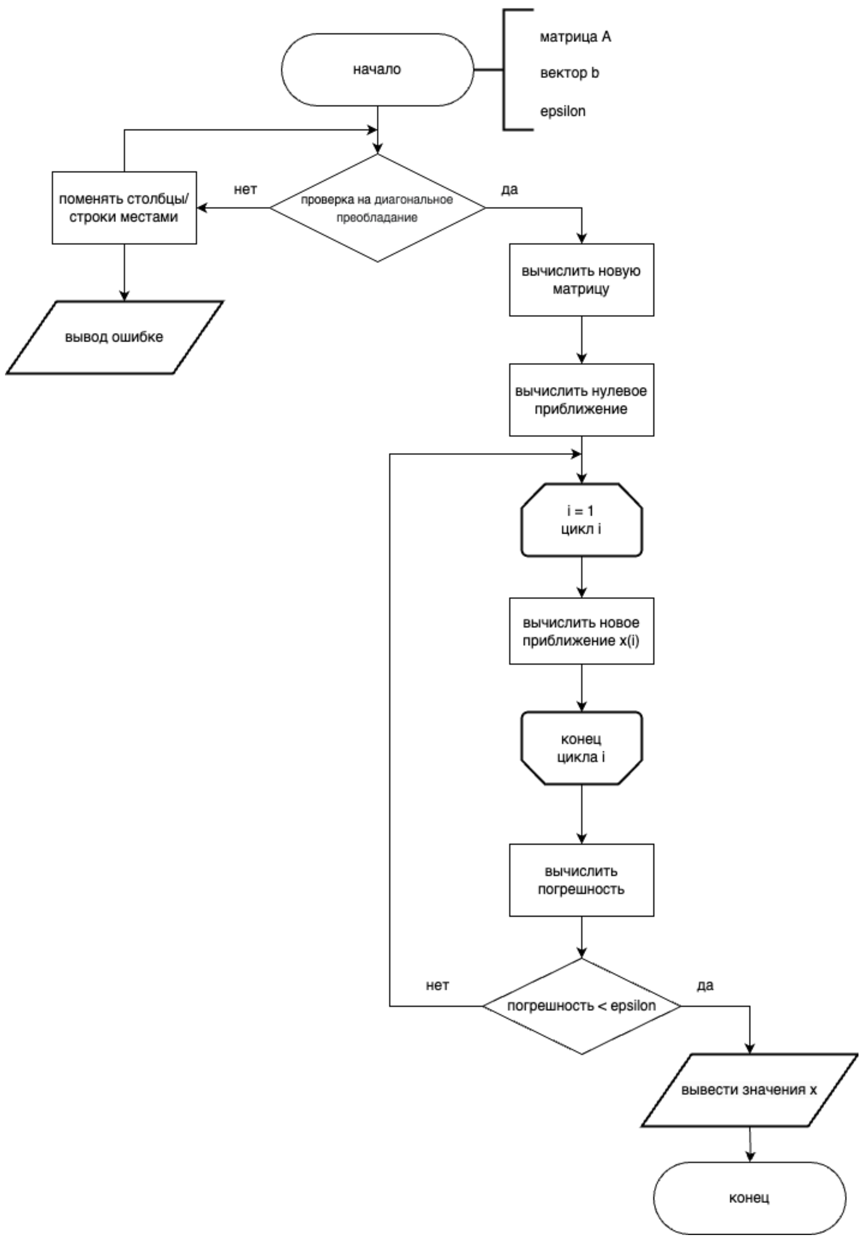
Если условие диагонального преобладания выполнено, то начинаем итерационный алгоритм. 1. Выбераем начальное приближение $x^{(0)}$ и задайте точность ε.
2. Для каждого $i = 1, 2, \ldots, N$:

Для каждого $j = 1, 2, \ldots, n$ вычисляем новое значение $x_j^{(i)}$ следующим образом:

$$x_j^{(i)} = \frac{1}{a_{jj}} \left(b_j - \sum_{\substack{k=1 \\ k \neq j}}^{n} a_{jk} x_k^{(i-1)} \right)$$

3. Если $\|x^{(i)} - x^{(i-1)}\| < \varepsilon$ алгоритм заканчивается. Также в коде стоит добавить максимальное число операций, чтобы искоючить ситуацию с бесконечной работой программы.

Блок-схема по составленному описанию метода



Код численного метода

```java

3 usages  Alice Ponomarenko *
class Result {

    1 usage
    private static final int MAX_ITERATIONS_COUNT = 1000;
    3 usages
    public static boolean isMethodApplicable = true;
    3 usages
    public static String errorMessage;

    /**
     * Complete the 'solveByGaussSeidel' function below.
     * The function is expected to return a DOUBLE_ARRAY.
     * The function accepts following parameters:
     * 1. INTEGER n
     * 2. 2D_DOUBLE_ARRAY matrix
     * 3. INTEGER epsilon
     */
    1 usage  Alice Ponomarenko
    public static boolean validateMatrix(int n, final List<List<Double>> matrix) {
        if (matrix.size() == n) {
            for (int i = 0; i < n; i++) {
                if (!(matrix.get(i).size() == n + 1)) {
                    return false;
                }
            }
        }
        return true;
    }

    1 usage  Alice Ponomarenko *
    public static List<Double> solveByGaussSeidel(int n, List<List<Double>> matrix, double epsilon) {
        if (!validateMatrix(n, matrix)) {
            isMethodApplicable = false;
            errorMessage = "\"The system has no diagonal dominance for this method. " +
                    "Method of the Gauss-Seidel is not applicable.\"";
            return null;
        }
        double[][] massiveMatrix = getQuadraticMassiveOfMatrix(n, matrix);
        double[] bVector = getBVector(n, matrix);
        double[][] diagonalDominanceMatrix = getDiagonalDominanceMatrix(n, massiveMatrix);
        if (diagonalDominanceMatrix == null) {
```

```java
        if (diagonalDominanceMatrix == null) {
            isMethodApplicable = false;
            errorMessage = "\"The system has no diagonal dominance for this method." +
                    " Method of the Gauss-Seidel is not applicable.\"";
            return null; //null value will be ignored by error "isMethodApplicable"
        } else {
            int[] permutationVector = getPermutationVector(massiveMatrix, diagonalDominanceMatrix);
            double[] array = getXVector(n, diagonalDominanceMatrix, bVector, epsilon);
            //permute columns to origin sequence
            double[] permutedBackVector = new double[permutationVector.length];
            for (int i = 0; i < permutationVector.length; i++) {
                permutedBackVector[permutationVector[i]] = array[i];
            }
            return Arrays.stream(permutedBackVector).boxed().collect(Collectors.toList());
        }
    }

    1 usage   Alice Ponomarenko *
    private static double[] getXVector(int n, double[][] massiveMatrix, double[] bVector, double epsilon) {
        double[] firstApproaching = new double[n];
        for (int i = 0; i < n; i++) {
            firstApproaching[i] = bVector[i] / massiveMatrix[i][i];
        }
        double[] approaching = new double[n];
        double[] previousApproaching = new double[n];
        int iterationCount = 0;
        System.arraycopy(firstApproaching, srcPos: 0, approaching, destPos: 0, n);
        do {
            //save x vector values from previous iteration
            System.arraycopy(approaching, srcPos: 0, previousApproaching, destPos: 0, n);
            for (int i = 0; i < n; i++) {
                //sum1 - sum of values from current iteration
                double sum1 = 0;
                //sum2 - sum of values from previous iteration
                double sum2 = 0;
                for (int j = 0; j < i; j++) {
                    sum1 += (massiveMatrix[i][j] * approaching[j]);
                }
                for (int j = i + 1; j < n; j++) {
                    sum2 += (massiveMatrix[i][j] * previousApproaching[j]);
                }
                approaching[i] = (bVector[i] - sum1 - sum2) / massiveMatrix[i][i];
            }
            iterationCount++;
```

```java
    } while (!checkStopPoint(n, massiveMatrix, approaching, bVector, epsilon)
            && iterationCount < MAX_ITERATIONS_COUNT);
    return approaching;
}

/**
 * @return true value only if iterations achieved epsilon approaching
 */
1 usage  Alice Ponomarenko
private static boolean checkStopPoint(int n, double[][] matrix, double[] x, double[] b, double epsilon) {
    boolean stop = false;
    for (int i = 0; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < n; j++) {
            sum += matrix[i][j] * x[i];
        }
        if (Math.abs(sum - b[i]) > epsilon) {
            stop = false;
            break;
        } else {
            stop = true;
        }
    }
    return stop;
}

1 usage  Alice Ponomarenko
private static double[] getBVector(int n, List<List<Double>> matrix) {
    double[] vector = new double[n];
    for (int i = 0; i < n; i++) {
        vector[i] = matrix.get(i).get(n);
    }
    return vector;
}

1 usage  Alice Ponomarenko
private static double[][] getQuadraticMassiveOfMatrix(int n, List<List<Double>> matrix) {
    double[][] newMatrix = new double[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            newMatrix[i][j] = matrix.get(i).get(j);
        }
    }
    return newMatrix;
```

```java
/**
 * @param n      size of quadratic matrix
 * @param matrix initial matrix
 * @return new matrix equal with initial matrix with diagonal dominance or null if one doesn't exist
 */
1 usage  Alice Ponomarenko
private static double[][] getDiagonalDominanceMatrix(int n, double[][] matrix) {
    boolean isDiagonalDominant;
    int countOfPermutations = 1;
    for (int i = 1; i < n + 1; i++) {
        countOfPermutations *= i;
    }
    for (int i = 0; i < countOfPermutations; i++) {
        isDiagonalDominant = true;
        List<double[][]> matrixPermutations = generateColumnPermutations(matrix);
        for (int j = 0; j < n; j++) {
            double sum = 0;
            for (int k = 0; k < n; k++) {
                sum += Math.abs(matrixPermutations.get(i)[j][k]);
            }
            //subtract module of a_ii
            if (sum - Math.abs(matrixPermutations.get(i)[j][j]) > Math.abs(matrixPermutations.get(i)[j][j])) {
                isDiagonalDominant = false;
            }
        }
        if (isDiagonalDominant) {
            return matrixPermutations.get(i);
        }
    }
    //there no matrix permutation which has diagonal permutation
    return null;
}

/**
 * @return list of all permutations of matrix by replacing columns to try to achieve diagonal domination
 */
1 usage  Alice Ponomarenko
public static List<double[][]> generateColumnPermutations(double[][] matrix) {
    List<double[][]> result = new ArrayList<>();
    int rows = matrix.length;
    int cols = matrix[0].length;
    int[] colIndices = new int[cols];

```

```
        for (int i = 0; i < cols; i++) {
            colIndices[i] = i;
        }
        List<List<Integer>> colPermutations = permute(colIndices);

        for (List<Integer> colIndexPermutation : colPermutations) {
            double[][] permutedMatrix = new double[rows][cols];
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    permutedMatrix[i][j] = matrix[i][colIndexPermutation.get(j)];
                }
            }
            result.add(permutedMatrix);
        }
        return result;
    }

    1 usage  Alice Ponomarenko
    private static List<List<Integer>> permute(int[] indices) {
        List<List<Integer>> result = new ArrayList<>();
        permute(indices,  start: 0, result);
        return result;
    }

    2 usages  Alice Ponomarenko
    private static void permute(int[] indices, int start, List<List<Integer>> result) {
        if (start == indices.length) {
            List<Integer> permutation = new ArrayList<>();
            for (int index : indices) {
                permutation.add(index);
            }
            result.add(permutation);
            return;
        }
        for (int i = start; i < indices.length; i++) {
            swap(indices, start, i);
            permute(indices,  start: start + 1, result);
            swap(indices, start, i);
        }
    }

```

```java
    2 usages  Alice Ponomarenko
    private static void swap(int[] indices, int i, int j) {
        int temp = indices[i];
        indices[i] = indices[j];
        indices[j] = temp;
    }

    1 usage  Alice Ponomarenko
    private static int[] getPermutationVector(double[][] startMatrix, double[][] permutMatrix) {
        int[] permutationVector = new int[startMatrix[0].length];
        for (int j = 0; j < permutMatrix[0].length; j++) {
            double[] column = new double[permutMatrix.length];
            for (int i = 0; i < permutMatrix.length; i++) {
                column[i] = permutMatrix[i][j];
            }
            int index = findColumnIndex(startMatrix, column);
            permutationVector[j] = index;
        }
        return permutationVector;
    }


    1 usage  Alice Ponomarenko
    private static int findColumnIndex(double[][] matrix, double[] column) {
        for (int j = 0; j < matrix[0].length; j++) {
            boolean match = true;
            for (int i = 0; i < matrix.length; i++) {
                if (matrix[i][j] != column[i]) {
                    match = false;
                    break;
                }
            }
            if (match) {
                return j;
            }
        }
        return -1; // if column was not found, return -1
    }

}
```

Примеры работы программы

1. Входные данные: (целые значение с диагональным преобладанием)

3

9 2 3 35

2 8 4 22

3 4 12 37

0.01

Выходные данные:

3.0

1.0

2.0

2. Входные данные: (дробные значение с диагональным преобладанием)

3

9.2 2.4 3.9 35

2 8.4 4 22.2

3.1 4 12.04 37.33

0.01

Выходные данные:

2.6575029874579332

1.0210529635699506

2.077037282774137

3. Входные данные: (единственное значение)

1

2 4

0.01

Выходные данные:

2.0

4. Входные данные: (Переставлены строки 1 и 2 примера 1, с диагональным преобладанием) 3

2 8 4 22

9 2 3 35

3 4 12 37

0.01

Выходные данные:

3.0

1.0

2.0

5. Входные данные: (Переставлены столбцы 1 и 2 примера 1, с диагональным преобладанием)

3

2 9 3 35

8 2 4 22

4 3 12 37

0.01

Выходные данные:

1.0

3.0

2.0

6. Входные данные: (Без диагонального преобладания)

3

2 8 4 22

2 9 3 35

3 4 12 37

0.01

Выходные данные:

The system has no diagonal dominance for this method. Method of the Gauss-Seidel is not applicable.

7. Входные данные: (Неправильное количество введенных чисел)

3

1 2 3 4

1 2 3

3 4 5 6

0.0003

Выходные данные:

The system has no diagonal dominance for this method. Method of the Gauss-Seidel is not applicable.

Выводы

Результаты запуска реализованного метода на различных данных
1. Метод правильно отрабатывает как и с целыми входными данными, так и с числами с дробной частью, потому что в коде вычисления производятся в типе double, а в java есть приведение типов из double в int.

2. Метод проверяет матрицу на диагональное преобладание, в случае отсутствия - переставляет строки и/или столбцы местами. Если же диагонального преобладания все же нет, то программа выводит сообщение об ошибке. Если во время исполнения потребовалось переставить строки и/или столбцы местами, то перед выводом ответа программа вернет исходную последовательность переменных и именно в таком порядке выведет в консоль.

3. Программа обрабатывает ситуации неправильного ввода данных ("неправильная матрица") и в таком случае выводит сообщение об ошибке.

4. Программа корректно работает с матрицей-вектором (матрицей 1 на 1).

Сравнение с другими методами интепроляции:

Метод Гаусса Принцип: Прямой метод, использует пошаговое исключение переменных для приведения системы к треугольному виду. Преимущества: Высокая точность, подходит для любой системы, для которой существует решение. Недостатки: Может быть медленным для больших систем (сложность $O(n^3)$), чувствителен к ошибкам округления, требует много памяти.

Метод Гаусса-Зейделя Принцип: Итерационный метод, улучшает приближение решения путем последовательного обновления значений переменных. Преимущества: Прост в реализации, эффективен для больших разреженных систем, требует меньше памяти. Недостатки: Не всегда сходится (требует диагонального доминирования матрицы), точность зависит от количества итераций и критерия остановки.

Метод Якоби Принцип: Итерационный метод, обновляет все переменные одновременно на каждой итерации. Преимущества: Простота параллельной реализации, прост в программировании и понимании. Недостатки: Часто требует больше итераций для сходимости по сравнению с методом Гаусса-Зейделя, не всегда сходится (требует диагонального доминирования матрицы).

Метод сопряженных градиентов Принцип: Итерационный метод, применяется для симметричных положительно определенных матриц, использует направление сопряженных градиентов для нахождения решения. Преимущества: Очень эффективен для больших разреженных систем, хорошая сходимость для положительно определенных матриц. Недостатки: Применим только к симметричным положительно определенным матрицам, более сложен в реализации по сравнению с методом Гаусса-Зейделя и методом Якоби.

Метод LU-разложения Принцип: Прямой метод, разлагает матрицу на произведение нижней и верхней треугольных матриц. Преимущества: Высокая точность, эффективен для многократных решений системы с одной и той же матрицей. Недостатки: Может быть медленным для больших систем (сложность $O(n^3)$), требует дополнительной памяти для хранения матриц L и U.

Сравнение с методом Гаусса-Зейделя: Преимущества метода Гаусса-Зейделя: Простота реализации, эффективность для больших разреженных систем, меньшее требование к памяти. Недостатки метода Гаусса-Зейделя: Не всегда сходится, требует диагонального доминирования матрицы, точность зависит от количества итераций и критерия остановки.

Анализ применимости метода Гаусса-Зейделя:

Метод Гаусса-Зейделя идеально подходит в следующих случаях:
- Разреженные системы: Когда матрица системы имеет много нулевых элементов, метод эффективно использует память и вычислительные ресурсы.
- Диагонально доминирующие матрицы: Если матрица имеет диагональное доминирование, метод обычно сходится быстро и стабильно.
- Большие системы: При решении больших систем, особенно с разреженными матрицами, метод может быть более эффективен по времени и памяти по сравнению с прямыми методами.
- Начальное приближение: Если можно получить хорошее начальное приближение, метод быстро сходится к решению.

Метод Гаусса-Зейделя может быть менее подходящим в следующих случаях:
- Плотные системы: При решении плотных матриц (мало нулевых элементов) метод может быть менее эффективен из-за большого количества операций.
- Слабо диагонально доминирующие или недиагонально доминирующие матрицы: В таких случаях метод может не сходиться или сходиться очень медленно.
- Высокая точность: Если требуется очень высокая точность, метод может потребовать слишком много итераций, особенно если начальное приближение не удачно.
- Системы с плохой обусловленность: Для плохо обусловленных систем метод может быть нестабильным и давать неточные результаты из-за накопления ошибок.

Алгоритмическая сложность метода: $O(n! + n^2)$

Сложность подготовки данных (валидация и перестановка для диагонального доминирования) имеет факториальную сложность $O(n!)$. После этого основной итерационный процесс в худшем случае имеет сложность $O(n^2 * MAX\_ITERATIONS\_COUNT)$.

Итого: $O(n! + n^2)$

Анализ численных ошибок метода интерполяции Ньютона:
1. Ошибки округления
2. Первоначальное приближение
3. Диагональное преобладание

Вариации по уменьшению численных ошибки:

1. Выбор начального приближения:

Хорошее начальное приближение может существенно улучшить сходимость и уменьшить накопление ошибок.

2. Контроль за условием диагонального доминирования:

Проверка и обеспечение диагонального доминирования матрицы, если это возможно, улучшает устойчивость метода.

3. Контроль за числом итераций:

Ограничение максимального числа итераций и использование более строгих критериев остановки помогает контролировать накопление ошибок.