Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
Факультет Программной Инженерии и Компьютерной Техники

Вычислительная математика

Решение системы линейных алгебраических уравнений СЛАУ

Вариант №9

Выполнил: Копытов Никита Эдуардович P3219
Преподаватель: Преподаватель практики Бострикова Дарья Константиновна

Санкт-Петербург 2025

Содержание

Цель

1. Изучить прямые и итерационные методы решения систем линейных алгебраических уравнения (СЛАУ).
2. Выполнить программную реализацию выбранного метода.

Описание метода

В данной лабораторной работе использовался метод простых итераций.

Метод простых итераций — это один из итерационных численных методов, используемых для нахождения решений систем линейных алгебраических уравнений. Этот метод основан на итеративном процессе, который позволяет постепенно приближаться к решению.

Основная идея метода заключается в следующем: дано уравнение $f(x)=0$, где $f(x)$ некоторая функция. Метод Простых итераций предлагает преобразовать это уравнение к виду $x=g(x)$, где $g(x)$ - другая функция. Затем начальное приближение x_0 выбирается произвольным образом, и затем последовательно вычисляются значения $x_1=g(x_0)$, $x_2=g(x_1)$, $x_3=g(x_2)$, и так далее.

Таким образом, получается последовательность приближений x_0 , x_1 , x_2 , и т.д. Условие сходимости метода простых итераций обычно проверяется через расчет нормы матрицы. Для того чтобы метод сходился, необходимо, чтобы модуль нормы был меньше 1 на всей области сходимости.

Итерации заканчиваются в тот момент, когда максимальный из векторов погрешностей станет меньше заданной точности.

Рабочая формула метода простой итерации:

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

Условие диагонального преобладания:

$$|a_{ii}|>\sum_{j\neq i}|a_{ij}|\,,\qquad i=1,2\,,\dots\,,n\,.$$

Листинг программы

Полный код программы на языке Java находится в качестве приложения в репозитории на сайте github.com по ссылке.

Класс LinearSystem, в котором производятся расчеты изначального приблежения `getInitialApproximation()` и расчеты приближенных значений `solve()`.

```
package nikita.math;

import java.util.ArrayList;

import nikita.exception.ExecutionException;
import nikita.output.ConsolePrinter;

public class LinearSystem {
	float absoluteAccuracy;
	Matrix matrix;
```

```
	ArrayList<Approximation> approximations = new ArrayList<Approximation>();
	int iteration = 0;

	public LinearSystem(float absoluteAccuracy, Matrix matrix) {
		this.absoluteAccuracy = absoluteAccuracy;
		this.matrix = matrix;

		if (!this.matrix.isDiagonalDominanceEnforcable()) {
			throw new ExecutionException("No perfect matching found. Diagonal dominance cannot be
enforced.");
		}
		this.matrix.enforceDiagonalDominance();

		ConsolePrinter.println(ConsolePrinter.LINE);
		ConsolePrinter.println(this.toString());
		ConsolePrinter.println(ConsolePrinter.LINE);

		this.getInitialApproximation();
		printApproximation(0, this.approximations.get(0).values, null);
		ConsolePrinter.println(ConsolePrinter.LINE);
		this.solve();
	}

	public void getInitialApproximation() {
		ArrayList<Float> values = new ArrayList<Float>();
		for (int i = 0; i < this.matrix.size; i++) {
			Row currentRow = this.matrix.rows[i];
			values.add(currentRow.result / currentRow.values[i]);
		}
		float[] primitiveValues = new float[values.size()];
		for (int i = 0; i < values.size(); i++) {
			primitiveValues[i] = values.get(i);
		}
		this.approximations.add(new Approximation(primitiveValues));
	}

	public void solve() {
		boolean converged = false;
		int maxIterations = 1000; // Ну чтобы слишком долго не считало если что

		while (!converged && iteration < maxIterations) {
			Approximation prevApproximation = this.approximations.get(this.approximations.size() -
1);
			float[] currentApprox = prevApproximation.values;
			float[] newApprox = new float[currentApprox.length];
			float[] diffVector = new float[currentApprox.length];
			float maxDiff = 0;

			// Новое приближение для каждого x
			for (int i = 0; i < matrix.size; i++) {
				Row currentRow = matrix.rows[i];
				float sum = 0;
				for (int j = 0; j < matrix.size; j++) {
					if (j != i) {
						sum += currentRow.values[j] * currentApprox[j];
					}
				}
				newApprox[i] = (currentRow.result - sum) / currentRow.values[i];

				// Абсолютная погрешность для каждого x
				float diff = Math.abs(newApprox[i] - currentApprox[i]);
				diffVector[i] = diff;
				if (diff > maxDiff) {
					maxDiff = diff;
				}
			}

			iteration++;
			printApproximation(iteration, newApprox, diffVector);
```

```
                        ConsolePrinter.println("Max difference: " + maxDiff);
                        ConsolePrinter.println(ConsolePrinter.LINE);

                        this.approximations.add(new Approximation(newApprox));

                        if (maxDiff < absoluteAccuracy) {
                                converged = true;
                        }
                }

                if (!converged) {
                        throw new ExecutionException("The iterative method did not converge within the maximum
allowed iterations.");
                }
        }

        private void printApproximation(int iter, float[] approx, float[] diffVector) {
                if (iter == 0) {
                        ConsolePrinter.println("Initial Approximation:");
                } else {
                        ConsolePrinter.println("Iteration " + iter + ":");
                }

                StringBuilder approximationStr = new StringBuilder();
                for (int i = 0; i < approx.length; i++) {
                        approximationStr.append(String.format("x%d = %.10f\t", i + 1, approx[i]));
                }
                ConsolePrinter.println(approximationStr.toString());

                if (diffVector != null) {
                        StringBuilder diffsStr = new StringBuilder("");
                        for (int i = 0; i < diffVector.length; i++) {
                                diffsStr.append(String.format("|x%d_diff| = %.10f\t", i + 1, diffVector[i]));
                        }
                        ConsolePrinter.println(diffsStr.toString());
                }
        }
}
```

Результат работы программы

Входной файл test-1.json:

```
{
    "matrix": {
        "size": 3,
        "content": [
            {
                "row": [10, 2, 1],
                "result": 1
            },
            {
                "row": [1, 11, 2],
                "result": 2
            },
            {
                "row": [0, 1, 12],
                "result": 3
            }
        ]
    },
    "absolute-accuracy": 0.01
}
```

Результат:

```
> file test-1.json
================================================================================
```

```
Parsed System of Linear Equations:

Matrix size: 3×3         Euclidian norm: 19.390720

/ 10x₁ + 2x₂ + 1x₃ = 1
| 1x₁ + 11x₂ + 2x₃ = 2
\ 0x₁ + 1x₂ + 12x₃ = 3

Absolute accuracy: 0.010000

================================================================================
Initial Approximation:
x1 = 0.1000000015       x2 = 0.1818181872       x3 = 0.2500000000
================================================================================
Iteration 1:
x1 = 0.0386363640       x2 = 0.1272727251       x3 = 0.2348484844
|x1_diff| = 0.0613636374        |x2_diff| = 0.0545454621        |x3_diff| = 0.0151515156
Max difference: 0.061363637
================================================================================
Iteration 2:
x1 = 0.0510606058       x2 = 0.1356060654       x3 = 0.2393939495
|x1_diff| = 0.0124242418        |x2_diff| = 0.0083333403        |x3_diff| = 0.0045454651
Max difference: 0.012424242
================================================================================
Iteration 3:
x1 = 0.0489393957       x2 = 0.1336501241       x3 = 0.2386994958
|x1_diff| = 0.0021212101        |x2_diff| = 0.0019559413        |x3_diff| = 0.0006944537
Max difference: 0.00212121
================================================================================
```

Вывод

В ходе выполнения лабораторной работы был подробно изучен и реализован метод простых итераций для решения СЛАУ. Данный метод основывается на последовательном уточнении искомого решения через повторяющиеся итерационные процедуры, что позволяет достичь требуемой точности при выполнении достаточного числа итераций. В сравнении с другими методами можно выделить следующие аспекты:

- Обычный метод Гаусса (без перестановок): В отличие от метода простых итераций, метод Гаусса является прямым способом решения СЛАУ, где решение находится за конечное число шагов (при отсутствии особенностей, таких как вырождение системы). При этом метод Гаусса подвержен накоплению ошибок округления при работе с матрицами, содержащими сильно различающиеся по модулю коэффициенты. Метод простых итераций, как правило, не испытывает таких проблем, так как использует повторяющийся процесс уточнения решения, однако его эффективность и точность сильно зависят от условий сходимости, а также от правильного выбора начального приближения.
- Метод Гаусса-Зейделя: Метод Гаусса-Зейделя можно рассматривать как улучшенный вариант метода простых итераций, когда при вычислении каждого приближения используются уже обновлённые значения переменных, что зачастую

ускоряет сходимость. В отличие от метода простых итераций, при котором используется только значение из предыдущей итерации, метод Гаусса-Зейделя использует комбинированный подход, что повышает эффективность при условии выполнения необходимых условий (например, диагональной доминантности). Тем не менее, как и метод простых итераций, метод Гаусса-Зейделя требует тщательного анализа системы для гарантирования сходимости, в то время как прямые методы, например метод Гаусса с выбором главного элемента, не зависят от итерационного характера решения задачи.

Таким образом, метод простых итераций является удобным инструментом для решения СЛАУ в случаях, когда выполнены необходимые условия сходимости и когда требуется обеспечить гибкость подхода при работе с большими, однако его применение требует предварительного анализа характеристик матрицы, что позволяет выбрать оптимальные параметры итерационного процесса.