Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени Н.Э.БАУМАНА
(национальный исследовательский университет)»

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа № 1

«Решение СЛАУ. Метод прогонки»

по дисциплине «Численные методы»

Работу выполнил

студент группы ИУ9-62Б

Жук Дмитрий

Москва, 2022

Цель работы

Целью данной работы является:

- Анализ метода прогонки и решение с помощью него СЛАУ с трёхдиагональной матрицей;
- Сравнение решения СЛАУ, полученного методом прогонки, с решением СЛАУ, полученного с помощью метода Гаусса.

Задание

Дано: $Ax = d$, где $A \in \mathbb{R}^{n\times n}$ и $x, d \in \mathbb{R}^n$, A – трехдиагональная матрица

Найти: Решение СЛАУ с помощью метода прогонки, т.е. найти x при известных A и d.

Теория

Метод прогонки используется для решения систем линейных уравнений вида $Ax = d$, где A – трёхдиагональная матрица. Представляет собой вариант метода последовательного исключения неизвестных.

Пусть массив a – элементы матрицы A под главной диагональю, b – на главной диагонали, c – над главной диагональю.

$$\begin{pmatrix} \mathrm{b}_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_2 & b_3 & c_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & \cdots & \cdots & 0 & a_{n-1} & b_n \end{pmatrix}$$

Соответствующая СЛАУ:

$$\begin{cases} b_1x_1 + c_1x_2 = d_1 \\ a_1x_1 + b_2x_2 + c_2x_3 = d_2 \\ \cdots \\ a_{n-1}x_{n-1} + b_nx_n = d_n \end{cases}$$

x_i вычисляется следующим образом:

$$x_i = \alpha_i x_{i+1} + \beta_i, \qquad i = \overline{n-1, 1}$$

$$x_n = \frac{d_n - a_{n-1}\beta_{n-1}}{a_{n-1}\alpha_{n-1} + b_n}$$

α_i, β_i вычисляются следующим образом:

$$\alpha_i = \frac{-c_i}{a_{i-1}\alpha_{i-1} + b_i}, \qquad i = \overline{2, n-1}$$

$$\beta_i = \frac{d_i - a_{i-1}\beta_{i-1}}{a_{i-1}\alpha_{i-1} + b_i}, \qquad i = \overline{2, n}$$

$$\alpha_1 = \frac{c_1}{b_1}, \qquad \beta_1 = \frac{d_1}{b_1}$$

Вычисление α_i, β_i называется прямым ходом метода прогонки. Вычисление x_i – обратным ходом метода прогонки.

Достаточные условия метода:

1. $|b_i| \geq |a_{i-1}| + |c_i|,\ i = \overline{2, n-1}$,
2. $|d_i| > |c_i|,\ i = \overline{2, n-1}$.

Метод Гаусса заключается в последовательном исключении переменных: с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы.

Пусть дана система:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = d_1 \\ \cdots \\ a_{i1}x_1 + \cdots + a_{in}x_n = d_i \\ \cdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = d_n \end{cases}$$

Приведем матрицу коэффициентов к верхнетреугольному виду: на первом шаге вычитаем из i-ой строки, где $i = \overline{2, n}$, первую строку, домноженную на $\frac{a_{i1}}{a_{11}}$. Далее аналогичным образом вычитаем вторую строку, в итоге получаем систему вида:

$$\begin{cases} a''_{11}x_1 + \cdots + a''_{1n}x_n = d''_1 \\ \cdots \\ a''_{ii}x_i + \cdots + a''_{in}x_n = d''_i \\ \cdots \\ a''_{nn}x_n = d''_n \end{cases}$$

Далее начинается обратной ход метода Гаусса. Вычитаем из i-ой строки, где $i = \overline{1, n-1}$, последнюю, домноженную на $\frac{a''_{in}}{a''_{nn}}$.

Продолжая этот процесс, придём к диагональной матрице:

$$\begin{cases} a'''_{11}x_1 = d'''_1 \\ \cdots \\ a'''_{ii}x_i = d'''_i \\ \cdots \\ a'''_{nn}x_n = d'''_n \end{cases}$$

Разделив i-ую строку на a'''_{ii} получаем решение.

Оценка погрешности для решения СЛАУ при отсутствии точного решения:

Найти вектор-решение x^* с помощью метода прогонки и вектор x с помощью метода Гаусса. При вычислении x^* и x с плавающей точкой возникает погрешность: $\varepsilon = x^* - x$.

Реализация

```
import * as fs from 'fs';

const verbose = (...args: any[]) => {
	if (Number(process.env.VERBOSE)) {
		// eslint-disable-next-line no-console
		console.log(...args);
	}
};

const more = <T, K>(arg: T, callback: (v: T) => K) => (
	Number(process.env.MORE) ? callback(arg) : arg
);

type InputType = {a: number[], b: number[], c: number[], d: number[], n:
number};

const read = (name: string): InputType => {
	const elems = fs.readFileSync(name, { encoding: 'utf8' })
		.split(/[\s\n]+/g)
		.filter(Boolean)
		.map(Number);
	const n = +(elems.shift() || 0);
	const ans = {
		n,
		a: elems.splice(0, n - 1),
		b: elems.splice(0, n),
		c: elems.splice(0, n - 1),
		d: elems.splice(0, n),
	};

	verbose('read:', { name, ans });

	return ans;
};

const solveProgon = ({
	a, b, c, d, n,
}: InputType) => {
	const alpha: number[] = [];
	const beta: number[] = [];
```

```
        for (let i = 0; i < n; i++) {
                const denominator = (a[i - 1] || 0) * (alpha[i - 1] || 0) +
b[i];
                alpha.push(-c[i] / denominator);
                beta.push((d[i] - (a[i - 1] || 0) * (beta[i - 1] || 0)) /
denominator);
        }

        const x: number[] = [];

        for (let i = n - 1; i >= 0; i--) {
                x[i] = (alpha[i] || 0) * (x[i + 1] || 0) + beta[i];
        }

        verbose('solve:', { alpha, beta, x });

        return x;
};

type FormatType = {m: number[][], d: number[], n: number};

const format = ({
        a, b, c, d, n,
}: InputType): FormatType => {
        const m = Array.from({ length: n }, () => Array.from({ length: n }, ()
=> 0));

        a.forEach((v, i) => { m[i + 1][i] = v; });
        b.forEach((v, i) => { m[i][i] = v; });
        c.forEach((v, i) => { m[i][i + 1] = v; });

        const ans = { m, d, n };

        verbose('format:', ans);

        return ans;
};

const clone = <T>(c: T): T => (Array.isArray(c) ? c.map(clone) : c) as T;

const solveGaus = ({ d: d1, m: m1, n }: FormatType) => {
        const d = clone(d1);
        const m = clone(m1);

        for (let j = 0; j < n; j++) {
                d[j] /= m[j][j];
                for (let i = n - 1; i >= j; i--) {
                        m[j][i] /= m[j][j];
                }

                for (let k = j + 1; k < n; k++) {
                        d[k] -= m[k][j] * d[j];
                        for (let i = n - 1; i >= j; i--) {
                                m[k][i] -= m[k][j] * m[j][i];
                        }
                }
```

```
        }

        for (let j = n - 1; j >= 0; j--) {
                for (let i = j - 1; i >= 0; i--) {
                        d[i] -= d[j] * m[i][j];
                }
        }

        verbose('solveGaus:', m, d);

        return d;
};

const makeResult = ({ m, n }: FormatType, x: number[]) => Array
        .from({ length: n }, (_, i) => {
                let di = 0;
                for (let j = 0; j < n; j++) {
                        di += m[i][j] * x[j];
                }
                return di;
        });

const inp = read(process.argv[2]);

const s1 = solveProgon(inp);
const f = format(inp);
const s2 = solveGaus(f);
const r = makeResult(f, s1);

/* eslint-disable no-console */
console.log('прогон:      ', s1);
console.log('Гаус:        ', s2);
console.log('разница:     ', more(s1.map((e, i) => (e - s2[i])), (s) =>
s.map((e) => e.toFixed(20))));
console.log('подстановка:', r);
console.log('разница:     ', more(inp.d.map((e, i) => (e - r[i])), (s) =>
s.map((e) => e.toFixed(20))));
/* eslint-enable no-console */
```

Листинг 1 — Метод прогонки и метод Гаусса для решения СЛАУ с трёхдиагональной матрицей

Результат

Для тестирования полученной программы в качестве трехдиагональной матрицы *A* была выбрана следующая матрица:

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix}$$

В качестве вектора d:

$$d = \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix}$$

В результате работы программы (Листинг 1) получаем значения:

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad x^* = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \varepsilon = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Как видно выше, не всегда вектор ε может содержать погрешность (нулевой вектор ошибок) в связи с использованием типов данных с двойной точностью. Протестируем программу на измененном векторе d:

$$d = \begin{pmatrix} 5 \\ 6 \\ 6 \\ 6 \end{pmatrix}$$

В результате работы программы (Листинг 1) получаем значения:

$$x = \begin{pmatrix} 0.9952153110047848 \\ 1.0191387559808611 \\ 0.9282296650717703 \\ 1.2679425837320575 \end{pmatrix}, \qquad x^* = \begin{pmatrix} 0.9952153110047848 \\ 1.0191387559808611 \\ 0.9282296650717703 \\ 1.2679425837320575 \end{pmatrix}, \qquad \varepsilon = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Так как погрешности нет, то сделаем обратную подстановку. Вычислим Ax и сравним его с d:

$$Ax = \begin{pmatrix} 5 \\ 5.999999999999999 \\ 6 \\ 6 \end{pmatrix}, \qquad d - AX = \begin{pmatrix} 0 \\ 8.881784197001252 \times 10^{-16} \\ 0 \\ 0 \end{pmatrix}$$

Вывод

В ходе выполнения лабораторной работы был изучен метод решения СЛАУ с трёхдиагональной матрицей: метод прогонки. Так же были реализованы методы прогонки и Гаусса на языке программирования JavaScript.

Для метода прогонки можно отметить то, что отсутствует методологическая (логическая) погрешность, но присутствует вычислительная погрешность в связи с использованием чисел с плавающей запятой, ведущая к высокому накоплению вычислительной ошибки.