

**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

**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:** , где и , 𝐴 – трехдиагональная матрица

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

**Теория**

**Метод прогонки** используется для решения систем линейных уравнений вида , где – трёхдиагональная матрица. Представляет собой вариант метода последовательного исключения неизвестных.

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

Соответствующая СЛАУ:

вычисляется следующим образом:

, вычисляются следующим образом:

Вычисление , называется прямым ходом метода прогонки. Вычисление – обратным ходом метода прогонки.

Достаточные условия метода:

1. ,
2. .

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:

Приведем матрицу коэффициентов к верхнетреугольному виду: на первом шаге вычитаем из -ой строки, где , первую строку, домноженную на . Далее аналогичным образом вычитаем вторую строку, в итоге получаем систему вида:

Далее начинается обратной ход метода Гаусса. Вычитаем из -ой строки, где , последнюю, домноженную на .

Продолжая этот процесс, придём к диагональной матрице:

Разделив -ую строку на получаем решение.

**Оценка погрешности для решения СЛАУ при отсутствии точного решения:**

Найти вектор-решение с помощью метода прогонки и вектор с помощью метода Гаусса. При вычислении и с плавающей точкой возникает погрешность: .

**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 была выбрана следующая матрица:

В качестве вектора :

В результате работы программы (Листинг 1) получаем значения:

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

В результате работы программы (Листинг 1) получаем значения:

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

**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.