Университет ИТМО
Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №1
Вариант – метод Гаусса

Выполнила: Екатерина Машина

Группа P3210

Преподаватель: Ольга Вячеславовна Перл

Санкт-Петербург
2020 г.

Цель работы

Реализовать метод Гаусса для решения СЛАУ с единственным набором решений заданной размерности.

Описание использованного метода

Метод Гаусса – это точный метод решения систем линейных алгебраических уравнений, заключающийся в том, чтобы привести исходную матрицу (составленную из коэффициентов при неизвестных и столбца свободных членов) к треугольному виду (нули под или над главной диагональю).

Изначально стоит сказать, что метод Гаусса применим исключительно в случае если $det(A) \neq 0$, иначе система уравнений не может быть решена данным методом.

В случае если определитель матрицы не нулевой алгоритм решения делится на две основные части – прямой и обратный ход.

Прямой ход:

1) Запишем систему уравнений в виде матрицы следующим образом:

$$\left(\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} & b_n \end{array}\right),$$

Далее начнем постепенно исключать из уравнений неизвестные (приводить матрицу к треугольному виду).

Приводить матрицу к треугольному виду следует линейными преобразованиями следующего вида: на $i-$ом шаге производится исключение x_i из уравнений, следующих в матрице за $i-$ым: каждое из этих уравнений (пробегаем по ним циклом по j) домножается на $^{-a_{j,i}}/_{a_{i,i}}$ и прибавляется к $i-$ому.

2) Производя линейные преобразования с матрицей исключим переменную x_1 из всех уравнений кроме первого. (Важно учитывать если на позиции элемента $a_{1,1}$ стоит 0, то необходимо переставить строки в таком порядке, чтобы $a_{1,1} \neq 0$). В результате данных преобразований матрица получит следующий вид:

$$\left(\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ 0 & a_{2,2}-\frac{a_{1,2}\cdot a_{2,1}}{a_{1,1}} & \cdots & a_{2,n}-\frac{a_{1,n}\cdot a_{2,1}}{a_{1,1}} & b_2-\frac{b_1\cdot a_{2,1}}{a_{1,1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n,2}-\frac{a_{1,2}\cdot a_{n,1}}{a_{1,1}} & \cdots & a_{n,n}-\frac{a_{1,n}\cdot a_{n,1}}{a_{1,1}} & b_n-\frac{b_1\cdot a_{n,1}}{a_{1,1}} \end{array}\right),$$

3) Далее пока в $n-$ой строке матрицы не останется единственный член с неизвестным x_n будем на каждом $i-$ом шаге исключать x_i из всех уравнений от $(i+1)-$го до $n-$го (Важно не забывать, что нули на главной диагонали недопустимы и в случае их появления необходимо переставлять строки местами). В итоге матрица будет иметь следующий вид:

$$\left(\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ 0 & a'_{2,2} & \cdots & a'_{2,n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{n,n} & b'_n \end{array}\right)$$

Обратный ход:

1) Двигаясь от последнего уравнения начнем находить корни уравнений. Первым найдем x_n.
2) Далее подставим его в предпоследнее уравнение и так далее.

Следовательно общая формула для нахождения корня будет иметь следующий вид:

$$x_i = \frac{a_{i,i}}{b_i - \sum_{j=i+1}^{n} \frac{x_j}{a_{i,j}}}.$$

Выводы:

Метод Гаусса, являющийся точным методом решения СЛАУ. При решении используются формулы, что позволяет точно сказать какова сложность алгоритма вычисления

Основным преимуществом метода Гаусса (помимо точности вычислений) является то, что он является универсальным и им можно решить практически любую систему линейных алгебраических уравнений.

Недостатки метода Гаусса:

- При большой количестве неизвестных необходимо хранить всю матрицу, что занимает большой объем памяти;
- Несмотря на то, что метод позволяет точно вычислять ответ, но из-за того, что этот ответ надо где-то хранить, то получить мы его можем только с точностью до какого либо типа данных (например с точностью до double) из-за этого продолжая вычисления на каждом шаге мы накапливаем погрешность.

Метод Гаусса нецелесообразно применять в разреженных матрицах, так как придется производить большое количество лишних операций, а также хранить эти нули.

Метод Гаусса с выбором главного элемента – это усовершенствованная версия метода Гаусса. Единственным отличием является то, что в реализации мы должны обеспечивать, чтобы на месте ведущего элемента всегда находился максимальный по модулю, что позволяет уменьшить погрешность в вычислениях (так как при использовании простого метода Гаусса элемент, находящийся на позиции главного может быть бесконечно мал и при делении на него получим большую погрешность).

Рассмотрим метод простых итерации и метод Гаусса-Зейделя, от метода Гаусса они отличаются прежде всего тем, что вычисления происходят следующим образом:

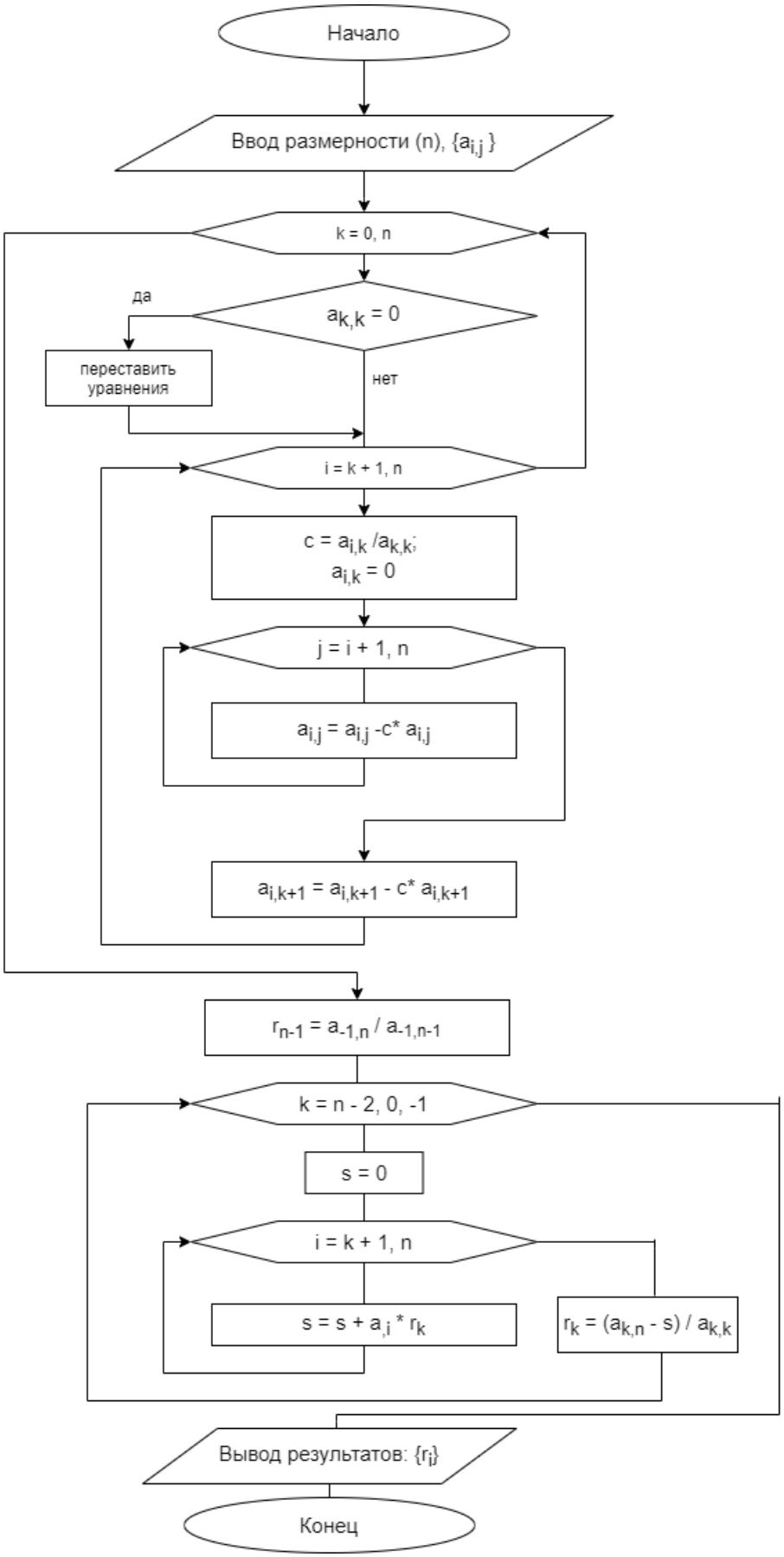
- задается начальное приближение;
- затем производятся итерации – этапы вычисления, в результате которых получают новые приближения, и так далее до тех пор, пока не получим результат с заданной точностью.

В отличии от метода Гаусса, эти методы не требуют хранения всей матрицы, а только нескольких векторов. Однако такие методы являются менее точными, так как в них решение вычисляется как предел последовательности.

Обобщив всё вышесказанное можно сделать следующий вывод:

Метод Гаусса хорошо подходит для матриц небольшой размерности, решение получается точнее, чем при использовании итерационных методов, после приведения матрицы к треугольному виду намного проще найти определитель. Однако метод Гаусса имеет относительно высокую погрешность из-за деления (данная проблема уменьшается в методе Гаусса с выбором главного элемента, хотя решение не сильно усложняется).

Блок-схема


Начало
Ввод размерности (n), {a_i,j }
k = 0, n
да
a_k,k = 0
переставить уравнения
нет
i = k + 1, n
c = a_i,k /a_k,k;
a_i,k = 0
j = i + 1, n
a_i,j = a_i,j -c* a_i,j
a_i,k+1 = a_i,k+1 - c* a_i,k+1
r_n-1 = a_-1,n / a_-1,n-1
k = n - 2, 0, -1
s = 0
i = k + 1, n
s = s + a_,i * r_k
r_k = (a_k,n - s) / a_k,k
Вывод результатов: {r_i}
Конец

Листинг численного метода

```
// straight and back run, getting solutions
List<Double> getAnswers() {
        int n = matrix.size();
        straightRun();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (Math.abs(this.getItem(i, j)) > EPSILON) {
                    System.out.format(colorize("[[RED]]Error: matrix is
not triangular. " +
                            "Item at row %d and column %d " +
                            "is not zero!%n[[WHITE]]"), i + 1, j + 1);
                    return null;
                }
            }
        }
        return backRun();
    }
    private void straightRun(){
        int n = matrix.size();
        boolean isLineOfZeros = false;
        for (int i = 0; i < n; i++) {
            int zeros = 0;
            if (Math.abs(this.getItem(i, i)) < EPSILON) {
                isLineOfZeros = true;

                for (int j = i + 1; j < n; j++)
                    if ((Math.abs(this.getItem(i, j)) < EPSILON))
zeros++;


                if ((zeros == n - i - 1) && Math.abs(this.getItem(i, n))
> EPSILON) {
                    System.out.println(colorize("[[RED]]No
solutions![[WHITE]]"));
                    System.exit(0);
                }

            }
        }
        if (isLineOfZeros == true) {
            System.out.println(colorize("[[RED]]Infinite " +
                    "number of solutions![[WHITE]]"));
            System.exit(0);
        }
    }
    private List<Double> backRun(){
        int n = matrix.size();
        List<Double> unknowns = new
ArrayList<Double>(Collections.nCopies(n, 0.0));
        for (int i = n - 1; i >= 0; i--) {
            Equation curEquation = matrix.get(i);
            Double ans = curEquation.getItem(n);

            for (int j = i + 1; j < n; j++) {
                ans -= curEquation.getItem(j) * unknowns.get(j);
            }
            unknowns.set(i, ans / curEquation.getItem(i));
```

```
        }
        return unknowns;
    }
```

// calculation of the residuals

```
List<Double> getResidual(List<Double> ans) {
    List<Double> residuals = new ArrayList<Double>();
    int n = matrix.size();
    for (int i = 0; i < n; i++) {
        Equation curEquation = matrix.get(i);
        Double res = 0.0;
        for (int j = 0; j < n; j++) {
            res += curEquation.getItem(j) * ans.get(j);
        }
        residuals.add(res - curEquation.getItem(n));
    }
    return residuals;
}
```

//calculation of the triangular matrix and determinant

```
double getTriangularMatrix() {
        int sizeOfMatrix = matrix.size();
        int topRowIndex = 0;
        for (int i = 0; i < sizeOfMatrix; i++) {
            int firstNotZero = -1;
            for (int j = topRowIndex; j < sizeOfMatrix && firstNotZero ==
-1; j++) {
                if (Math.abs(this.getItem(j, i)) > EPSILON) {
                    firstNotZero = j;
                }
            }
            if (firstNotZero == -1) continue;

            if (topRowIndex != firstNotZero) {
                Equation tmp = matrix.get(topRowIndex);
                matrix.set(topRowIndex, matrix.get(firstNotZero));
                matrix.set(firstNotZero, tmp);
                count *= (-1);
            }

            Equation topRow = matrix.get(topRowIndex);
            for (int j = topRowIndex + 1; j < sizeOfMatrix; j++) {
                matrix.set(j, matrix.get(j).summarize(topRow.multiply(-
this.getItem(j, i) / topRow.getItem(i))));
            }
            topRowIndex++;
        }

        double determinant = 1.0;
        for (int i = 0; i < sizeOfMatrix; i++) {
            determinant *= this.getItem(i, i);
        }
        if (Math.abs(determinant)<EPSILON) return 0;
        return determinant * count;
    }
```

Примеры

Изнчально выводимый при старте программы текст:

This is a program that implements the Gauss method.

Input Format:

n is the number of unknowns and the number of equations.
Next, in n lines, put n coefficients for unknowns and the free term of each equation separated by space.
Or type "random" if you want the coefficients to be random.

Would you like to import file?
If yes , type name of text file in format "filename" here.
If no, type "no"

Примеры работы:

Пример 1

```
no
4
random
Your matrix is:

 -6,000  -4,000   2,000  -3,000 |   5,000
  0,000   3,000  -9,000  -6,000 |   2,000
 -1,000 -10,000   8,000  -4,000 |   0,000
 -6,000   5,000  -8,000 -10,000 |  -8,000

det(A) = -2757

Triangular matrix :
 -6,000  -4,000   2,000  -3,000 |   5,000
  0,000   3,000  -9,000  -6,000 |   2,000
  0,000   0,000 -20,333 -22,167 |   5,389
  0,000   0,000   0,000  -7,533 | -14,495

Solutions :
-0,868
-2,573
-2,363
1,924
Residuals :
-1,776 * 10^(-15)
0,000
3,553 * 10^(-15)
0,000
```

Пример 2

```
test1
Your matrix is:

  1,000   3,000   5,000 |   3,000
  1,000   3,000   6,000 |   8,000
  9,000   8,000   5,000 |   6,000

det(A) = 19

Triangular matrix :
  1,000   3,000   5,000 |   3,000
  0,000 -19,000 -40,000 | -21,000
  0,000   0,000   1,000 |   5,000

Solutions :
6,263
-9,421
5,000
Residuals :
0,000
0,000
0,000
```

Пример 3

no
3
1 2 3 4
2 3 4 5
3 4 5 6
Your matrix is:

1,000	2,000	3,000	4,000
2,000	3,000	4,000	5,000
3,000	4,000	5,000	6,000

det(A) = 0

Triangular matrix :

1,000	2,000	3,000	4,000
0,000	-1,000	-2,000	-3,000
0,000	0,000	0,000	0,000

Solutions :
Infinite number of solutions!

Пример 4

no
3
0 0 0 0
0 0 0 0
0 0 0 1
Your matrix is:

0,000	0,000	0,000	0,000
0,000	0,000	0,000	0,000
0,000	0,000	0,000	1,000

det(A) = 0

Triangular matrix :

0,000	0,000	0,000	0,000
0,000	0,000	0,000	0,000
0,000	0,000	0,000	1,000

Solutions :
No solutions!