федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 1
по дисциплине «Вычислительная математика»
Базовые задачи

Выполнил:
Студент группы P32302
Русинов Д. С.
Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2023

Содержание

1 Описание метода, расчетные формулы **2**

2 Блок-схема **3**

3 Листинг исходного кода **3**

4 Примеры работы программы **6**

5 Вывод **9**

1 Описание метода, расчетные формулы

Метод Гаусса позволяет найти решения системы линейных алгебраических уравнений путём приведения матрицы системы к треугольному виду элементарными преобразованиями, которые, как известно из курса линейной алгебры, не меняют линейную зависимость строк и, что то же самое, не меняют решения системы линейных уравнений.

Рассмотрим систему линейных уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

И ее матричный вид:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Будем триангулировать матрицу

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{pmatrix}$$

Для этого найдем ненулевой элемент в первом столбце и переместим его строку наверх (если такого нет, то пропустим этот шаг, ведь первый столбец будет нулевым, что удовлетворяет нашему методу)

НУО ненулевой элемент был в первой строке. Тогда вычтем из каждой строки i ниже первой, домноженную на число a_{i1}/a_{11}.

Получим матрицу:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a'_{22} & a'_{23} & b'_2 \\ 0 & a'_{32} & a'_{33} & b'_3 \end{pmatrix}$$

Далее рассмотрим матрицу, полученную вычеркиванием первой строки и первого столбца из нашей:

$$\begin{pmatrix} a'_{22} & a'_{23} & b'_2 \\ a'_{32} & a'_{33} & b'_3 \end{pmatrix}$$

И совершим тот же шаг. И так далее, пока мы не триангулируем всю матрицу:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ 0 & a^*_{22} & a^*_{23} & b^*_2 \\ 0 & 0 & a^*_{33} & b^*_3 \end{pmatrix}$$

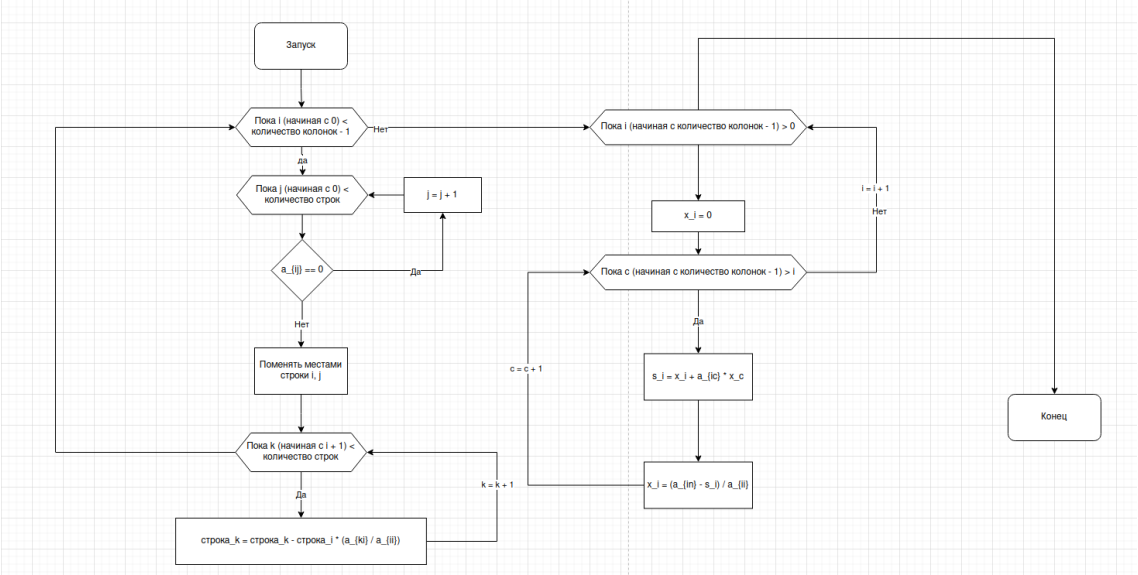
Вернемся в обозначения системы:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ 0 + a^*_{22}x_2 + a^*_{23}x_3 = b^*_2 \\ 0 + 0 + a^*_{33}x_3 = b^*_3 \end{cases}$$

Отсюда видно, что для вычисления x_i достаточо вычислить все $x_j : j \in \{n, \dots, i-1\}$ далее мы подставим их и перенесём в правую часть. Получим линейное уравнение с одной неизвестной. При $a_{ii} \neq 0$ оно всегда имеет решение, а при $a_{ii} = 0$ решение уравнения существует тогда и только тогда, когда $b_i = 0$.

Очевидным образом этот алгоритм обобщается на СЛАУ любых размеров.

2 Блок-схема



3 Листинг исходного кода

Программа была реализована на языке C++.

Листинг 1: main.cpp

```cpp
//
// Created by ruskaof on 17/02/23.
//

#include <optional>
#include "gauss_method.h"
#include "../util/tolerant_equals.h"
#include "../matrix/triangulation/triangulation.h"


static std::optional<std::vector<double>> get_solution(Matrix &matrix);

static double get_determinant_of_triangular(const Matrix &matrix);

std::optional<std::vector<double>> get_residual(Matrix &matrix, std::optional<std::vector<double>> &solution);

GaussResult gauss_method(const Matrix &input) {
    auto copy = Matrix(input);
    size_t rows_swapped_n = make_triangulated(copy);
    double determinant =
            rows_swapped_n % 2 == 0 ? get_determinant_of_triangular(copy) : -get_determinant_of_triangular(copy);
    auto solution = get_solution(copy);
    auto residual = get_residual(copy, solution);
    return GaussResult{determinant, copy, solution, residual};
}

static double get_determinant_of_triangular(const Matrix &matrix) {
    double determinant = 1.0;
    for (size_t diagonal_index = 0; diagonal_index < matrix.rows(); diagonal_index++) {
        determinant *= matrix[diagonal_index][diagonal_index];
    }
    return determinant;
}


static std::optional<std::vector<double>> get_solution(Matrix &matrix) {
    std::vector<double> solution(matrix.rows());
    for (size_t diagonal_index = matrix.rows() - 1;; diagonal_index--) {
        double variable_coeff = matrix[diagonal_index][diagonal_index];
        double free_term = matrix[diagonal_index][matrix.columns() - 1];

        if (tolerantly_equal(variable_coeff, 0)) {
            if (tolerantly_equal(free_term, 0)) {
                solution[diagonal_index] = 0;
                continue;
            } else {
                return std::nullopt;
            }
        }

        double other_row_variables_sum = 0.0;

        for (size_t column_index = diagonal_index + 1; column_index < matrix.columns() - 1; column_index++) {
            other_row_variables_sum += matrix[diagonal_index][column_index] * solution[column_index];
        }
        solution[diagonal_index] = (free_term - other_row_variables_sum) / variable_coeff;


        if (diagonal_index == 0) {
            return solution;
        }
    }
}

std::optional<std::vector<double>> get_residual(Matrix &matrix, std::optional<std::vector<double>> &solution) {
    if (!solution.has_value()) {
        return std::nullopt;
    }

    std::vector<double> evaluated_right(matrix.rows(), 0);
    for (size_t row_index = 0; row_index < matrix.rows(); row_index++) {
        for (size_t column_index = 0; column_index < matrix.rows(); column_index++) {
            evaluated_right[row_index] += matrix[row_index][column_index] * solution.value()[column_index];
        }
    }

    for (size_t i = 0; i < matrix.rows(); i++) {
        evaluated_right[i] -= matrix[i][matrix.columns() - 1];
    }

    return evaluated_right;
}
```

Листинг 2: gauss_method.cpp

```cpp
#include <locale>
#include <chrono>
#include <fstream>

#include "matrix/Matrix.h"
#include "gauss/gauss_method.h"
#include "random/matrix_random.h"
#include "input/matrix_input.h"

static const auto PROJECT_LOCALE = std::locale("ru_RU.utf8");

void print_result(GaussResult &gr) {
    std::cout << "Determinant: " << gr.determinant << '\n';
    std::cout << "Triangulated matrix:" << '\n' << gr.triangular_matrix.to_string() << '\n';
    if (gr.solution.has_value()) {
        std::cout << "Solution:" << '\n';
        for (size_t i = 0; i < gr.solution.value().size(); i++) {
            std::cout << "x" << i + 1 << ": " << gr.solution.value()[i] << '\n';
        }
        std::cout << "Residual:" << '\n';
        for (size_t i = 0; i < gr.solution.value().size(); i++) {
            std::cout << "x" << i + 1 << ": " << gr.residual.value()[i] << '\n';
        }
    } else {
        std::cout << "There is no solution" << '\n';
    }
}

int main() {
    std::cin.imbue(PROJECT_LOCALE);
    std::cout.imbue(PROJECT_LOCALE);

    std::cout << "Press 1 to generate random matrix 20x21, 2 to enter data by hand, 3 to read from file" << std::endl;
    int enter;
    if (!(std::cin >> enter) || (enter != 1 && enter != 2 && enter != 3)) {
        std::cerr << "Could not read your answer" << std::endl;
        return 1;
    }

    Matrix matrix(0, 0);

    if (enter == 1) {
        auto distribution = std::uniform_real_distribution<double>(-3, 3);
        matrix = generate_random_matrix(20, 21, distribution);
    } else if (enter == 2) {
        matrix = read_matrix(std::cin, std::cout);
    } else {
        std::cout << "Enter file path" << std::endl;
        std::string filename;
        std::cin >> filename;

        std::ifstream file(filename);
        file.imbue(PROJECT_LOCALE);

        if (file.is_open()) {
            matrix = read_matrix(file, std::cout);
        } else {
            std::cerr << "Could not read file" << std::endl;
        }
    }

    std::cout << "Rows: " << matrix.rows() << '\n' << "Columns: " << matrix.columns() << '\n';
    std::cout << "You entered matrix: " << '\n' << matrix.to_string() << std::endl;

    auto start_time = std::chrono::high_resolution_clock::now();
    auto result = gauss_method(matrix);
    auto end_time = std::chrono::high_resolution_clock::now();
    auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>(end_time - start_time);
    print_result(result);
    std::cout << "Elapsed time: " << elapsed_time.count() << " ms" << std::endl;
}
```

Листинг 3: trianguation.cpp

```cpp
//
// Created by ruskaof on 19/02/23.
//

#include "triangulation.h"
#include "../../util/tolerant_equals.h"


size_t make_triangulated(Matrix &matrix) {
    size_t swap_times = 0;
    // Iterate through each column to make all the elements below `column_index` zero
    for (size_t diagonal_index = 0; diagonal_index < matrix.columns() - 1; diagonal_index++) {
        // Find a row with a non-zero element on `diagonal_index` index to subtract it from all the others
        for (size_t row_index = diagonal_index; row_index < matrix.rows(); row_index++) {
            if (!tolerantly_equal(matrix[row_index][diagonal_index], 0)) {
                if (row_index != diagonal_index) {
                    std::swap(matrix[row_index], matrix[diagonal_index]); // now our row is on index `diagonal_index`
                    swap_times++;
                }
                for (size_t lower_row_index = diagonal_index + 1; lower_row_index < matrix.rows(); lower_row_index++) {
                    double coeff = matrix[lower_row_index][diagonal_index] / matrix[diagonal_index][diagonal_index];
                    matrix[lower_row_index] = matrix[lower_row_index] - matrix[diagonal_index] * coeff;
                }
                break;
            }
        }
    }
    return swap_times;
}
```

Остальную программу можно найти тут: github

4 Примеры работы программы

Листинг 4: Случайная генерация матрицы

```
/home/ruskaof/labs/numerical_methods/lab1/cmake-build-sanitized/lab1
Press 1 to generate random matrix 5x6, 2 to enter data by hand, 3 to read from file
1
Rows: 5
Columns: 6
You entered matrix:
    -0.63     0.84     1.31    -2.26     2.32    -1.35
     1.18    -2.55     0.76     2.47    -1.45     1.93
    -1.89    -0.72    -2.43    -0.65    -0.66     2.85
    -0.63     2.23    -0.42     1.27    -0.96    -2.15
     1.25     0.70     2.01     1.01     1.83     2.97

Determinant: -49,3217
Triangulated matrix:
    -0.63     0.84     1.31    -2.26     2.32    -1.35
     0.00    -0.97     3.21    -1.77     2.91    -0.61
     0.00     0.00   -17.09    12.02   -17.33     8.93
     0.00     0.00     0.00     3.01    -2.02    -0.17
     0.00     0.00     0.00    -0.00     1.56     5.36

Solution:
x1: 0,0976196
x2: -1,20171
x3: -2,42695
x4: 2,25033
x5: 3,43937
Residual:
x1: 1,33227e-15
x2: 0
x3: -7,10543e-15
x4: -1,11022e-15
x5: 0
Elapsed time: 0 ms

Process finished with exit code 0
```

Листинг 5: Ручной ввод

```
/home/ruskaof/labs/numerical_methods/lab1/cmake-build-sanitized/lab1
Press 1 to generate random matrix 5x6, 2 to enter data by hand, 3 to read from file
2
Please enter dimensions columns of the matrix - an unsigned integer `n`
2
Please enter the matrix data itself - n*n + n double values
1,0 2,0 3,0
4,0 5,0 6,0
Rows: 2
Columns: 3
You entered matrix:
    1.00    2.00    3.00
    4.00    5.00    6.00

Determinant: -3
Triangulated matrix:
    1.00    2.00    3.00
    0.00   -3.00   -6.00

Solution:
x1: -1
x2: 2
Residual:
x1: 0
x2: 0
Elapsed time: 0 ms

Process finished with exit code 0
```

Листинг 6: Ввод из файла

```
Please enter dimensions columns of the matrix - an unsigned integer 'n'
Please enter the matrix data itself - n*n + n double values
Rows: 20
Columns: 21
You entered matrix:

// Matrix cout skipped

Solution:
x1: -2,682
x2: -4,279
x3: -3,591
x4: -6,301
x5: 3,601
x6: 2,911
x7: 9,499
x8: 0,432999
x9: 8,32
x10: 2,945
x11: -9,595
x12: 0,376
x13: 2,112
x14: 7,475
x15: 4,925
x16: 4,953
x17: -7,244
x18: 1,461
x19: -6,841
x20: 8,851
Residual:
x1: 0
x2: 1,42109e-14
x3: -2,84217e-14
x4: -1,13687e-13
x5: 0
x6: 5,68434e-14
x7: 0
x8: 0
x9: 0
x10: -1,13687e-13
x11: 0
x12: 0
x13: 0
x14: 0
x15: 0
x16: 2,84217e-14
x17: -2,27374e-13
x18: 0
x19: -1,35003e-13
x20: -4,12115e-13
Elapsed time: 0 ms

Process finished with exit code 0
```

На больших данных (Матрица 1500×1501) с ключом -O3 программа выполняется около 1500 мс.

5 Вывод

Метод Гаусса эффективен (его асимптотическая сложность O(3)), интуитивно понятен и прост в реализации. Также он позволяет найти аналитическое решение, а не численное, что позволяет ему работать с высокой точностью. Он не накладывает дополнительные ограничения на матрицу системы, как это делают некоторые другие методы. Также в процессе выполнения, он триангулирует матрицу, что позволяет вычислить очень быстро вычислить ее определитель, почти не используя дополнительное время.