Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Методы численного анализа

ОТЧЁТ

к лабораторной работе

на тему

Решение систем линейных алгебраических уравнений (СЛАУ)
методом Гаусса и с помощью его модификаций

Выполнил: студент группы 053506
Слуцкий Никита Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

Оглавление

Вариант 6 (Номер в журнале – 21)

Цели выполнения задания:

- Изучить метод Гаусса для решения СЛАУ и его модификации
- Получить численное решение заданной СЛАУ
- Составить алгоритм решения СЛАУ указанными методами для организации вычислений на ЭВМ
- Составить программу решения по разработанному алгоритму
- Проверить правильность работы программы

Краткие теоретические сведения:

В общем виде систему линейных алгебраических уравнений можно записать в виде:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n = b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \ldots + a_{mn} \cdot x_n = b_m \end{cases}$$

Имея отдельно матрицу коэффициентов A и матрицу свободных членов B это можно компактно записать в виде A * x = B. В случае наличия решений система является совместной, иначе — несовместной.

Всегда можно пробовать систему методом Крамера и даже получить решение, если определитель матрицы не равен нулю. Но при больших размерностях вычисления определителей требуют большого количества вычислений (порядка n!). В этом случае удобнее пользоваться методом Гаусса.

Определю элементарные преобразования над системой (или полной матрицей системы).

- умножение всей строки (уравнения) на число, не равное нулю
- прибавление к одной строке (уравнению) другой строки (уравнения), умноженной на число
- обмен мест двух строк (уравнений) в системе

Метод Гаусса по-другому называется методом последовательного исключения. В общем случае он состоит из прямого и обратного ходов. Рассматриваемые здесь модификации метода привносят модификации в часть прямого хода.

Сначала рассмотрю классическую схему единственного деления. Пусть a_{11} не равняется нулю. Тогда можно исключить из всех последующих строк ([2..n]) переменную x_1. Просто вычту последовательно из 2-го, 3-го .. n-го уравнений системы 1-е, умноженное на a_{i1} / a_{11}. Получу обновлённую матрицу. Теперь коэффициенты в строках [2..n] обновлены, а при x_1 в них теперь вообще стоят нули. Далее пусть a_{22} не равняется нулю. Аналогично исключу x_2 из строк [3..n]. В итоге после n-1 таких шагов я должен получить треугольную матрицу с нулями под главной диагональю. Это был классический метод Гаусса.

Рассмотрю также схему частичного и полного выбора. В классическом методе при исключении переменных из уравнений уравнения сохранялись в исходном порядке.

Метод частичного выбора. Пусть сейчас k-я строка (=> k-й шаг и k-я переменная на очереди). Найду в k-ой колонке под рассматриваемой переменной (в строках i = [k..n]) максимальный элемент a_{ik} и поменяю местами k-ю строку и найденную i-ю. Зачем ? Во избежание сильного

роста коэффициентов. Ведь при вычитании строк и исключении неизвестного мы умножаем строку на множитель a_{ik} / a_{kk}. А при малом $<< 0$ a_{kk} множитель стремится к большим значениям. Поэтому нахожу максимальный элемент в колонке, меняю строки местами и сохраняю “адекватность” множителя.

Метод полного выбора. Если в прошлой модификации я выбирал максимальный элемент в колонке, далее менял строки и делал всё как обычно, то здесь уже буду выбирать максимальный элемент во всей оставшейся (то есть ниже, чем текущая строка) матрице. Предположим, я сейчас на 1-м шаге. Начинаю искать максимальный элемент в матрице (именно матрице главных коэффициентов) в строках [1..n]. Положим, что я нашёл какое-то a_{kl}, k и l – номер строки и столбца. Первым делом я меняю 1-ю строку с k-ой, а далее исключаю из уравнения НЕ 1-ю переменную (как это делалось ранее), а l-ую. Таким образом после процесса под a_{1l} будут нули. Перехожу на следующую строчку, начинаю искать максимальный элемент в части матрицы [2..n], найду его, получу номер строки и столбца, поменяю 2-ю и найденную строки местами и исключу переменную в найденном столбце в уравнениях [2+1 .. n]. Аналогично проделываю n-1 раз. На выходе получится не треугольная матрица (в своём явном виде), но получится матрица, ровно подходящая под дальнейшее вычисление переменных обратным ходом. Например, такая:

```
-0.5300 3.5300 0.8100 1.8700 0.9200 4.2000
2.0404 0.0000 3.6516 1.0908 2.0081 4.8306
1.5563 0.0000 0.0000 2.7106 0.8165 3.0223
0.6572 0.0000 0.0000 0.0000 2.7175 2.4433
3.1199 0.0000 0.0000 0.0000 0.0000 2.5176
```

Обратный ход. После получения треугольной матрицы либо матрицы аналогичной по смыслу (именно для интересуемой нас цели – метода обратного хода) можно приступать к обратному ходу. Начинается он с последней строки матрицы. Там исключены все переменные, кроме какой-то одной. Таким образом, можно сразу её численно выразить. В предпоследней строке исключены все переменные, кроме той же, что была в последней, и ещё одной. Значение последней у меня уже есть. Таким образом вторая переменная также выражается без проблем. Обобщённо – в каждой новой строке (если идти снизу вверх) добавляется новая неизвестная, а остальные из этой строки были вычислены ещё на этапе прошлых строк. В этом и заключается суть метода обратного хода.

Задание

Методом Гаусса (3-мя описанными его разновидностями) найти с точностью 0.0001 численное решение системы A * x = b, где A = k * C + D. Матрицы C, D, b задаются ниже.

$$\text{Вектор } \mathbf{b} = (4{,}2;\ 4{,}2;\ 4{,}2\ ;\ 4{,}2;\ 4{,}2)^{\mathrm{T}},$$

$$C = \begin{bmatrix} 0{,}2 & 0 & 0{,}2 & 0 & 0 \\ 0 & 0{,}2 & 0 & 0{,}2 & 0 \\ 0{,}2 & 0 & 0{,}2 & 0 & 0{,}2 \\ 0 & 0{,}2 & 0 & 0{,}2 & 0 \\ 0 & 0 & 0{,}2 & 0 & 0{,}2 \end{bmatrix}, \quad D = \begin{bmatrix} 2{,}33 & 0{,}81 & 0{,}67 & 0{,}92 & -0{,}53 \\ -0{,}53 & 2{,}33 & 0{,}81 & 0{,}67 & 0{,}92 \\ 0{,}92 & -0{,}53 & 2{,}33 & 0{,}81 & 0{,}67 \\ 0{,}67 & 0{,}92 & -0{,}53 & 2{,}33 & 0{,}81 \\ 0{,}81 & 0{,}67 & 0{,}92 & -0{,}53 & 2{,}33 \end{bmatrix}.$$

Программная реализация:

Ниже можно ознакомиться с программной реализацией главных функций по триангуляции, обратному ходу, а также вызов этого всего из главного файла. Вспомогательные функции и перегруженные операторы (реализации) опущены.

```cpp
std::vector<std::size_t> Triangulate(std::vector<std::vector<double>> &full_matrix,
                                     const GaussSolvingType &solution_type)
{
    const auto subtract_rows{[&](const std::size_t from, const std::size_t which, const double ratio) -> void {
        for (std::size_t col = 0; col < full_matrix.size() + 1; ++col)
            full_matrix[from][col] -= full_matrix[which][col] * ratio;
    }};

    const auto swap_rows{[&](const std::size_t first, const std::size_t second) -> void {
        for (std::size_t col = 0; col < full_matrix.size() + 1; ++col)
            std::swap(full_matrix[first][col], full_matrix[second][col]);
    }};

    const auto find_row_with_max_main_element{[&](const std::size_t from) -> std::size_t {
        auto response{from};
        for (std::size_t row = from + 1; row < full_matrix.size(); ++row)
            if (std::abs(full_matrix[row][from]) > std::abs(full_matrix[response][from]))
                response = row;
        return response;
    }};

    auto find_position_with_max_matrix_element{[&](const std::size_t from) -> std::pair<std::size_t, std::size_t> {
        auto maximum{full_matrix[from][0]};
        for (std::size_t row = from + 1; row < full_matrix.size(); ++row)
            maximum = std::max(maximum, *std::max_element(
                    std::cbegin(full_matrix[row]),
                    --std::cend(full_matrix[row]),
                    [](const auto first, const auto second) -> bool { return std::abs(first) < std::abs(second); }
            ));

        for (std::size_t row = from + 1; row < full_matrix.size(); ++row)
            for (std::size_t col = 0; col < full_matrix.size(); ++col)
                if (full_matrix.at(row).at(col) == maximum) return {row, col};

        return {from, 0};
    }};

    std::vector<std::size_t> variables_excluding_order{};

    switch (solution_type)
    {
        case GaussSolvingType::kSchemeOfTheOnlyDivision:
            for (std::size_t row = 0; row < full_matrix.size(); ++row)
            {
                variables_excluding_order.push_back(row);
                if (IsNear(full_matrix.at(row).at(row), 0.0))
                    throw std::runtime_error("Error: Main diagonal element = 0");
                for (std::size_t lower_row = row + 1; lower_row < full_matrix.size(); ++lower_row)
                {
                    const auto ratio{full_matrix.at(lower_row).at(row) / full_matrix.at(row).at(row)};
                    subtract_rows(lower_row, row, ratio);
                }
            }
            break;
        case GaussSolvingType::kSchemeOfPartialSelection:
            for (std::size_t row = 0; row < full_matrix.size(); ++row)
            {
                const auto row_with_max_first_item{find_row_with_max_main_element(row)};
                swap_rows(row_with_max_first_item, row);
                variables_excluding_order.push_back(row);

                if (IsNear(full_matrix.at(row).at(row), 0.0))
                    throw std::runtime_error("Error: Main diagonal element = 0");

                for (std::size_t lower_row = row + 1; lower_row < full_matrix.size(); ++lower_row)
                {
                    const auto ratio{full_matrix.at(lower_row).at(row) / full_matrix.at(row).at(row)};
                    subtract_rows(lower_row, row, ratio);
                }
            }
            break;
        case GaussSolvingType::kSchemeOfFullSelection:
            for (std::size_t row = 0; row < full_matrix.size(); ++row)
            {
```

```
                const auto[row_with_max, col_with_max]{find_position_with_max_matrix_element(row)};
                variables_excluding_order.push_back(col_with_max);

                swap_rows(row_with_max, row);

                if (IsNear(full_matrix.at(row).at(col_with_max), 0.0))
                    throw std::runtime_error("Error: Main element = 0");

                for (std::size_t lower_row = row + 1; lower_row < full_matrix.size(); ++lower_row)
                {
                    const auto ratio{full_matrix[lower_row][col_with_max] / full_matrix[row][col_with_max]};
                    subtract_rows(lower_row, row, ratio);
                }
            }
            break;
    }

    return variables_excluding_order;
}
```

```
std::vector<double> GetSolutionByBackSubstitution(std::vector<std::vector<double>> &triangulated,
                                                  const std::vector<std::size_t> &variables_counting_order)
{
    std::vector<double> response(triangulated.size());

    std::size_t var_counter{0};

    for (std::size_t current_row = triangulated.size() - 1;
         current_row >= 0; --current_row) // current row in system !! (from last)
    {
        if (var_counter >= variables_counting_order.size()) break;
        // which var we can count on this row
        const auto current_variable_number{variables_counting_order.at(var_counter++)};
        response.at(current_variable_number) = triangulated[current_row].at(triangulated.size()) /
                                               triangulated.at(current_row).at(current_variable_number);

        for (int rest_row = 0; rest_row < current_row; ++rest_row)
            triangulated.at(rest_row).at(triangulated.size()) -= triangulated.at(rest_row).at(current_variable_number)
                                                                 * response.at(current_variable_number);
    }

    return response;
}
```

```
std::vector<double> SolveByGauss(const std::vector<std::vector<double>> &main_coefficients,
                                 const std::vector<double> &free_coefficients,
                                 const GaussSolvingType &solution_type)
{
    auto to_triangulate{GetFullSystemMatrix(main_coefficients, free_coefficients)};

    auto excluding_order{Triangulate(to_triangulate, solution_type)};
    std::reverse(std::begin(excluding_order), std::end(excluding_order));

    std::cout << to_triangulate;

    const auto last_row_variables_count = std::accumulate(
            std::cbegin(to_triangulate.back()),
            --std::cend(to_triangulate.back()),
            0,
            [&](const auto response, const auto current) -> std::size_t {
                return IsNear(current, 0.0) ? response : response + 1;
            });

    if (last_row_variables_count >= 2)
        throw std::runtime_error("System has infinite number of solutions");

    auto response{GetSolutionByBackSubstitution(to_triangulate, excluding_order)};

    return response;
}
```

```
int main()
{
```

```cpp
    const auto accuracy{GetNumberOfSignsAfterDot(kAccuracy)};
    auto coefficients{kMatrixC * kOption + kMatrixD};
    const auto &free_coefficients{kVectorB};


    try
    {
        std::cout << "Scheme of the only division: \n";
        const auto only_sol{SolveByGauss(coefficients, free_coefficients, GaussSolvingType::kSchemeOfTheOnlyDivision)};
        std::cout << std::setprecision(accuracy) << "Roots: " << only_sol << '\n';

        std::cout << "Scheme of partial selection: \n";
        auto part_select{SolveByGauss(coefficients, free_coefficients, GaussSolvingType::kSchemeOfPartialSelection)};
        std::cout << std::setprecision(accuracy) << "Roots: " << part_select << '\n';

        std::cout << "Scheme of full selection: \n";
        const auto full_select{SolveByGauss(coefficients, free_coefficients, GaussSolvingType::kSchemeOfFullSelection)};
        std::cout << std::setprecision(accuracy) << "Roots: " << full_select << '\n';


        coefficients.at(1).at(2) += 0.01;
        coefficients.at(0).at(4) += 0.01;
        coefficients.at(3).at(0) -= 0.01;
        coefficients.at(4).at(2) -= 0.01;

        auto sol_with_error{SolveByGauss(coefficients, free_coefficients, GaussSolvingType::kSchemeOfFullSelection)};
        std::cout << GetNorm(sol_with_error - full_select) << '\n';
    }
    catch (const std::exception &exception)
    {
        std::cout << exception.what();
    }

    return 0;
}
```

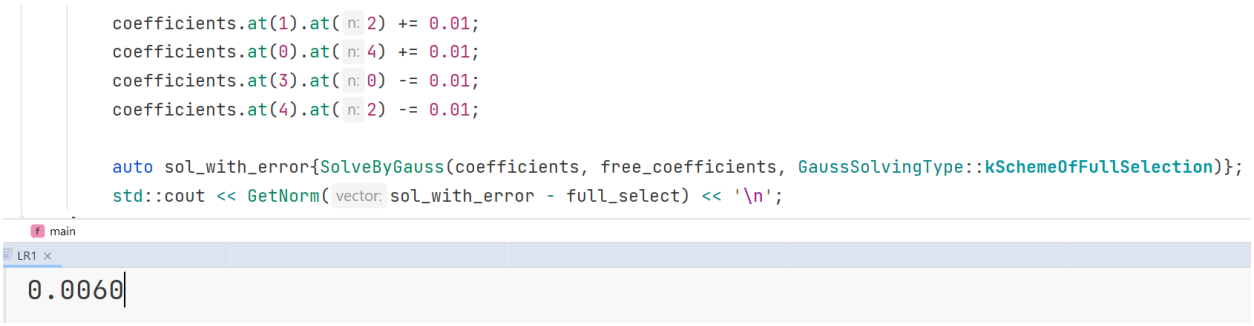
Полученные результаты:

```
"D:\Other\HomeWork\Methods Of Numerical Analysis\LR1\CPP\
Scheme of the only division:
3.5300 0.8100 1.8700 0.9200 -0.5300 4.2000
0.0000 3.6516 1.0908 2.0081 0.8404 4.8306
0.0000 0.0000 2.7106 0.8165 2.4222 3.0223
0.0000 0.0000 0.0000 2.7175 1.7737 2.4433
0.0000 0.0000 -0.0000 0.0000 3.1199 2.1962
Roots: 0.8070 0.8135 0.3535 0.4396 0.7039

Scheme of partial selection:
3.5300 0.8100 1.8700 0.9200 -0.5300 4.2000
0.0000 3.6516 1.0908 2.0081 0.8404 4.8306
0.0000 0.0000 2.7106 0.8165 2.4222 3.0223
0.0000 0.0000 0.0000 2.7175 1.7737 2.4433
0.0000 0.0000 -0.0000 0.0000 3.1199 2.1962
Roots: 0.8070 0.8135 0.3535 0.4396 0.7039

Scheme of full selection:
-0.5300 3.5300 0.8100 1.8700 0.9200 4.2000
2.0404 0.0000 3.6516 1.0908 2.0081 4.8306
1.5563 0.0000 0.0000 2.7106 0.8165 3.0223
0.6572 0.0000 0.0000 0.0000 2.7175 2.4433
3.1199 0.0000 0.0000 0.0000 0.0000 2.5176
Roots: 0.8070 0.8135 0.3535 0.4396 0.7039

Process finished with exit code 0
```

Оценка

```
coefficients.at(1).at( n: 2) += 0.01;
coefficients.at(0).at( n: 4) += 0.01;
coefficients.at(3).at( n: 0) -= 0.01;
coefficients.at(4).at( n: 2) -= 0.01;

auto sol_with_error{SolveByGauss(coefficients, free_coefficients, GaussSolvingType::kSchemeOfFullSelection)};
std::cout << GetNorm( vector: sol_with_error - full_select) << '\n';
```

При изменении случайно выбранных коэффициентов матрицы на значение +- 0.01 и вычислении решений СЛАУ с новыми значениями получил новое решение. Норма (евклидова) разности векторов старого и нового решения отличается не более, чем на 0.01.



Тестовые задания

Настало время протестировать программу. Выше была продемонстрирована работа программы для предложенной по заданию матрицы. Ниже рассмотрю несколько краевых случаев и просто примеров работы моей программы.

Будет прилагаться вид исходной полной матрицы системы и то, как реагирует программа на те или иные входные данные.

Функция для подсчёта вызывалась в следующем виде:

```
solution = SolveByGauss({{1, 2, 3, 4},
                         {2, 1, 2, 3},
                         {3, 2, 1, 2},
                         {4, 3, 2, 1}},
                        {7, 6, 7, 18},
                        GaussSolvingType::kSchemeOfPartialSelection);
```

```
Source full system matrix:
3 2 1
1 4 -3
---------------
Solution: 1 -1
```

```
Source full system matrix:
1 1 1 2
2 -1 -6 -1
3 -2 0 8
---------------
Solution: 2 -1 1
```

```
Source full system matrix:
1 2 3 4 7
2 1 2 3 6
3 2 1 2 7
4 3 2 1 18
--------------
Solution: 2 1 5 -3
```

Теперь касательно каких-то краевых случаев. К ним относится несовместность системы и наличие бесконечного количества решений. В подобных случаях программа выбрасывает исключение. Отловлю его и посмотрю, что происходит (текст и случаи исключения реализованы мной).

```
try
{
    solution = SolveByGauss({{2, 3, 6},
                             {0, 4, 5},
                             {0, 0, 0}},
                            {7, 5, 6},
                            GaussSolvingType::kSchemeOfFullSelection);
    std::cout << "---------------\nSolution: " << solution << '\n';
}
catch (const std::exception &exception)
{
    std::cout << exception.what();
}
```

```
Source full system matrix:
2 3 6 7
0 4 5 5
0 0 0 6
The system is incompatible
```

```
Source full system matrix:
7 3 -2 4 0
-6 -1 -1 1 1
9 7 8 14 2
1 2 -3 5 1
The system is incompatible
```

```
Source full system matrix:
1.00 3.00 -2.00 -2.00 -3.00
-1.00 -2.00 1.00 2.00 2.00
-2.00 -1.00 3.00 1.00 -2.00
-3.00 -2.00 3.00 3.00 -1.00
Triangulated matrix:
-2.00 -1.00 3.00 1.00 -2.00
-0.33 2.33 0.00 -1.33 -4.33
-1.14 0.00 0.00 1.43 -0.86
0.00 0.00 0.00 0.00 0.00
---------------
System has infinite number of solutions
```

В случае с примером, в котором бесконечное число решений, напечатал преобразованную матрицу. Если мне будет необходимо узнать размерность пространства, в котором будут строиться решения, я пойду вверх от последней строки и буду искать первую ненулевую строку, в которой посчитаю количество ненулевых коэффициентов перед переменными. Это количество минус один – и будет размерностью пространства.

Выводы

Метод Гаусса – хороший прямой метод для решения СЛАУ. Тут не надо вычислять много определителей, находить обратную матрицу. А если учесть, что для нахождения определителя или обратной матрицы, собственно, я скорее всего и буду использовать прямой ход Гаусса (приведение к треугольной), то это тем более это хороший способ. Другие методы используют его как часть себя, а сам по себе для решения СЛАУ он самостоятелен и, следовательно, не является таким трудоёмким, как метод Крамера или метод обратной матрицы (которые могут ссылаться к нему для своих подзадач).

В реализованной программе контролируется деление на ноль или на очень малый элемент — поэтому в случае чего будет сформировано исключение.

+ мой вывод для другого (чужого) отчёта:

Для совместных систем (определитель которых не равен и не находится достаточно близко к нулю) реализованный на ЭВМ метод Гаусса помогает найти точное решение (по крайней мере, в рамках той точности, которую гарантирует дробные типы в используемом языке программирования). Метод выбора элемента по всей матрице помогает избежать проблем при наличии нулей на диагонали и т.д., однако требует большего количества

операций (а именно дополнительные прохождения по массиву при поиске максимального элемента).

Касательно определения точности. Для определения погрешности можно придать случайные изменения (0.1, например) некоторым коэффициентам полной матрицы системы, посчитать решение с получившейся матрицей и найти норму разности векторов оригинального решения и нового. Полученное значение можно использовать для попытки охарактеризовать погрешность.