Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчёт по лабораторной работе**

**«Метод Жордана-Гаусса для решения систем линейных алгебраических уравнений»**

**Выполнил**:

студент группы 3823Б1ПМ1

Болтенков С. С.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_gp43zxaetv0f)

[Метод решения 4](#_m8nec4tvjkyw)

[Руководство пользователя 5](#_6hnssn2urt3h)

[Описание программной реализации 6](#_bc5scjxgi6xk)

[Подтверждение корректности 7](#_t9rhc2cojgr0)

[Результаты экспериментов 8](#_69m533yzwv65)

[Заключение 12](#_2keu5v4ast8o)

[Приложение 13](#_w6d7o4cyhtlj)

[Список литературы 15](#_wqa7ewlfwpij)

# **Постановка задачи**

В данной лабораторной работе требовалось:

1. Реализовать метод Жордана-Гаусса для решения систем линейных алгебраических уравнений
2. В процессе реализации применить: наследование, шаблоны, исключения.

# **Метод решения**

Метод Жордана-Гаусса является модификацией метода Гаусса для решения СЛАУ.

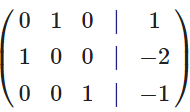
1. На каждом шаге выбирается не нулевой ведущей элемент, в строке и столбце, в которых еще не выбирался такой элемент.
2. Следующим шагом с помощью выбранного элемента зануляются все значения в столбце, с использованием элементарных преобразований, которые не меняют общего решения СЛАУ.

Доказано, что следующие три типа строчечных операций с расширенной матрицей не меняют множество решений:

1. Транспозиция строк.
2. Умножение строки на константу.
3. Прибавление к строке матрицы другой, умноженной на константу.

После работы данного метода получится приведенная матрица ступенчатого вида, из которой легко получить общее решение исходной СЛАУ.

Например, если получилась вот такая матрица:



Её решением будет .

При вычислениях с плавающей точкой может быть большая погрешность при деление на маленькие по модулю значение. Поэтому в реализации данный алгоритм был немного модифицирован. На каждом шаге в не нулевом столбце (есть хотя бы один элемент не равный нулю) выбирается наибольшее по модулю значение в качестве ведущего элемента. Также был реализован класс для работы с “большими” рациональными дробями, при использование которого вектор ошибки будет нулевым.

# **Руководство пользователя**

Взаимодействие с программой осуществляется через консоль. При запуске необходимо выбрать, что нужно вычислить:

1. 0 - найти обратную матрицу.
2. 1 - вычислить определитель матрицы
3. 2 - решить СЛАУ

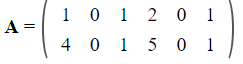
Если был выбран 0 или 1 далее необходимо ввести размерность квадратной матрицы и саму матрицу. Если была выбрана 2, то далее необходимо ввести количество строк и столбцов в матрице и количество столбцов неизвестных, далее ввести матрицу и столбцы.

Если вычисления будут проводиться с использованием длинных дробей, то матрица должна вводиться в следующем виде.

Матрице:



Будет соответствовать матрица (целая часть, числитель и знаменатель):



# **Описание программной реализации**

Программная реализация включает в себя следующие файлы:

1. Файл JordanGauss.cpp содержит реализацию функции main и некоторых других функций для нахождения обратной матрицы и вычисления определителя.
2. Файл LongInt.h содержит описание класса длинных целых чисел.
3. Файл LongInt.cpp содержить реализацию класса длинных целых чисел.
4. Файл LongFraction.h содержит описание класса длинных дробей.
5. Файл LongFraction.cpp содержит реализацию класса длинных дробей.
6. Файл MathVector.h содержит описание и реализацию шаблонного класса математического вектора.
7. Файл Matrix.h содержит описание и реализацию шаблонного класса матрицы.
8. Файл matrixProcessing.h содержит описание и реализацию шаблонного класса для решения СЛАУ.
9. Файл ZERO.h содержит описание класса для сравнения чисел с плавающей точкой и длинных дробей с нулём.
10. Файл ZERO.cpp содержит реализацию класса для сравнения чисел с плавающей точкой и длинных дробей с нулём.
11. Файл exeptionMatrixProc.h содержит описание класса исключений, которые могут возникнуть при работе с матрицами.
12. Файл exeptionMatrixProc.cpp содержит описание класса исключений, которые могут возникнуть при работе с матрицами.

# **Подтверждение корректности**

Для подтверждения корректности реализованного алгоритма были проведены следующие тесты:

1. Процедурно генерировалась матрица полного ранга и столбец неизвестных. После выполнения алгоритма, исходная матрица умножалась на полученный вектор решений. Следующим шагом вычислялся вектор невязки [1] для исходного столбца неизвестных и полученного в результате работы алгоритма. Ошибкой считалось максимальное по модулю значение в векторе невязки. При вычислениях с использованием типа данных double максимальная ошибка составила . При вычислениях с использованием реализованного класса длинных дробей ошибка тождественно равна нулю.
2. Второй тип тестов содержал СЛАУ, которые проверяли, что программа корректно определяет в каких случаях система не имеет решения. Также были проведены тесты с использованием матриц неполного ранга. Также крайние случаи, когда матрица - это строка или столбец. Во всех случаях алгоритм отработал корректно.

# **Результаты экспериментов**

В данной главе рассмотрим точность вычислений в числах с плавающей точкой в различных экспериментах. Также сравним производительность вычислений в стандартных типах данных с реализованным классом длинных дробей.

В первом эксперименте генерируется матрица полного ранга случайным образом. Полученная матрица проходит 1000 итераций на каждой из них ее диагональные элементы умножаются на определенную константу строго большую единицы (но достаточно маленькую, чтобы на 1000 итерации тип данных double не переполнился). Ниже приведены результаты эксперимента в графиках.

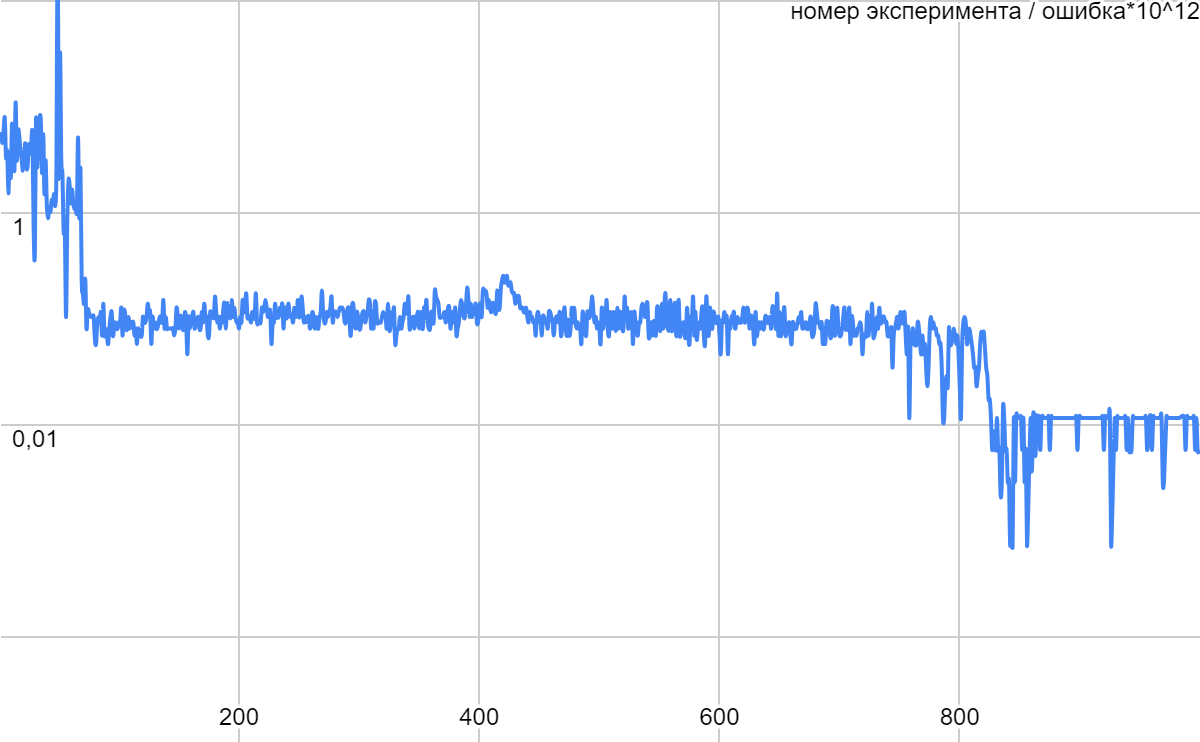


Рисунок 1. График изменения ошибки относительно номера эксперимента. Как видно из графика с ростом диагональных элементов ошибка уменьшается.

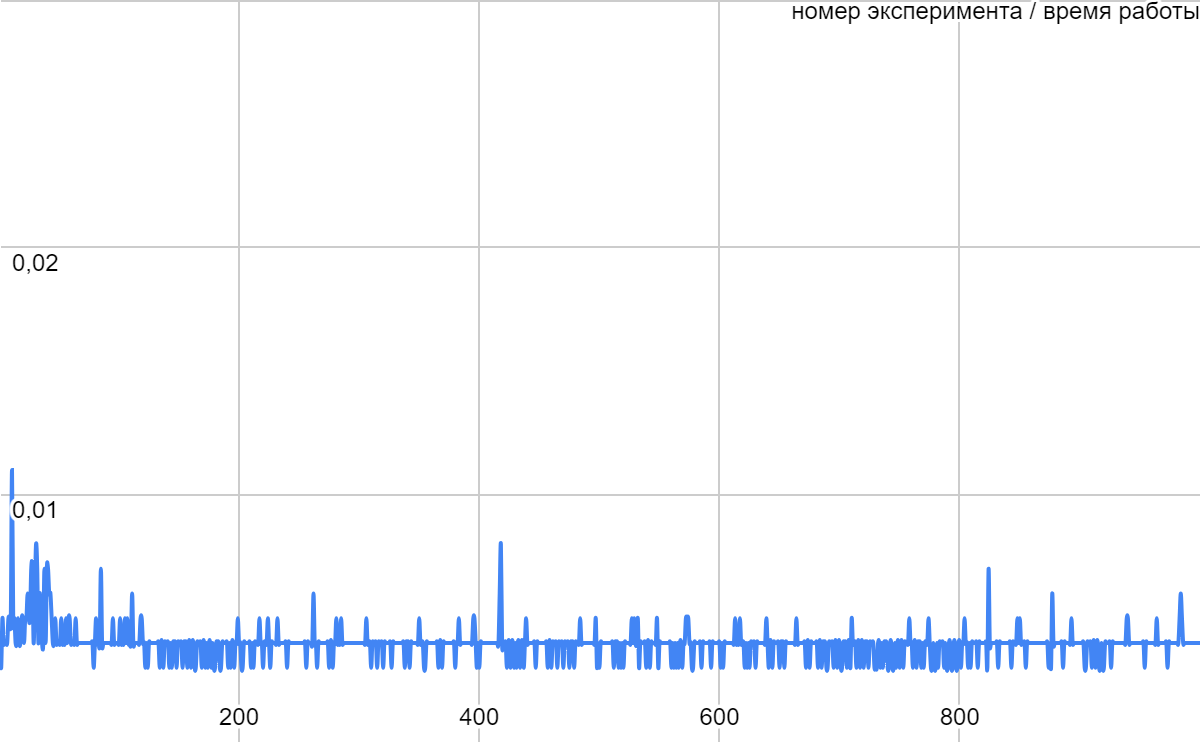


Рисунок 2. Время работы программы относительно номера эксперимента. Как видно из графика время работы программы практически не изменяется при увеличении диагональных элементов.

Теперь аналогичное тестирование проведем для больших дробей.

Как и ожидалось в каждом эксперименте ошибка была тождественно равна нулю.

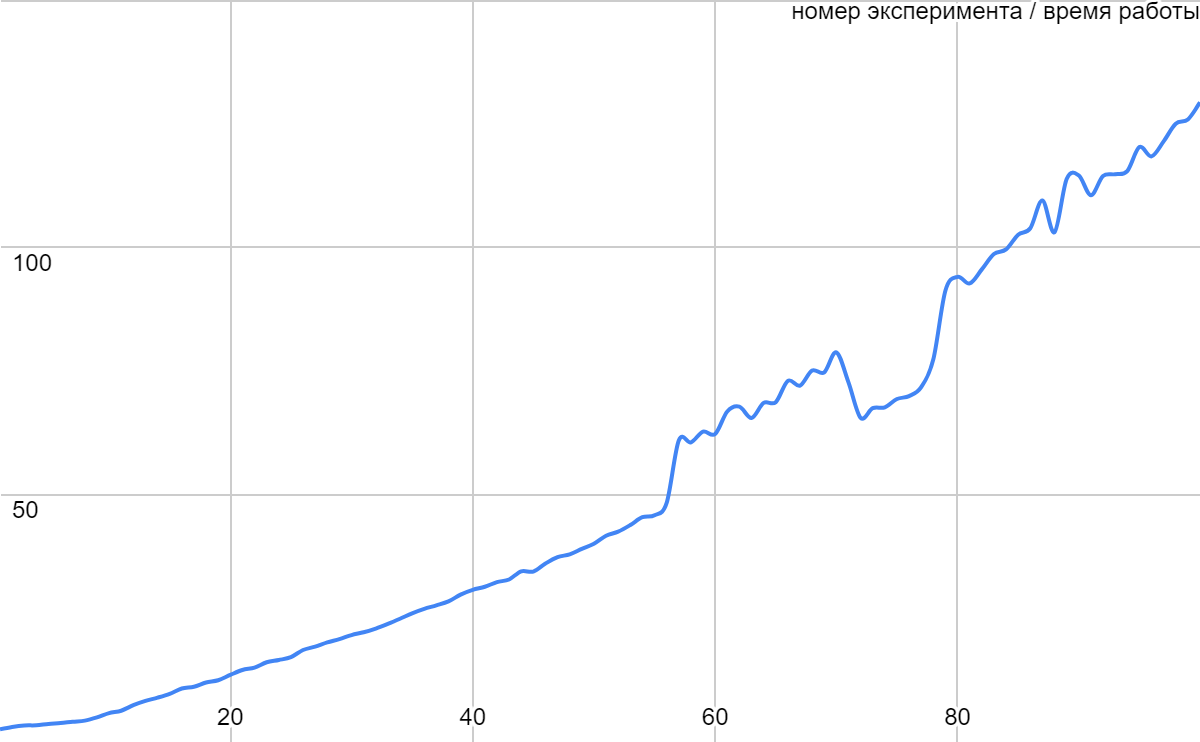


Рисунок 3. Время работы программы относительно номера эксперимента. Как видно из графика при увеличении диагональных элементов, время работы также увеличивается. Это объясняется тем, что в этом типе данных любое значение представимо и, например, если в каком-то из экспериментов вывести вектор ответов, то можно увидеть большие дроби в числителях и знаменателях, которых несколько сотен цифр.

Во втором эксперименте введем в рассмотрение число , которое будет равно разности максимального и минимального по модулю значения в матрице. Посмотрим как будет изменяться ошибка в зависимости от “кучности” чисел.

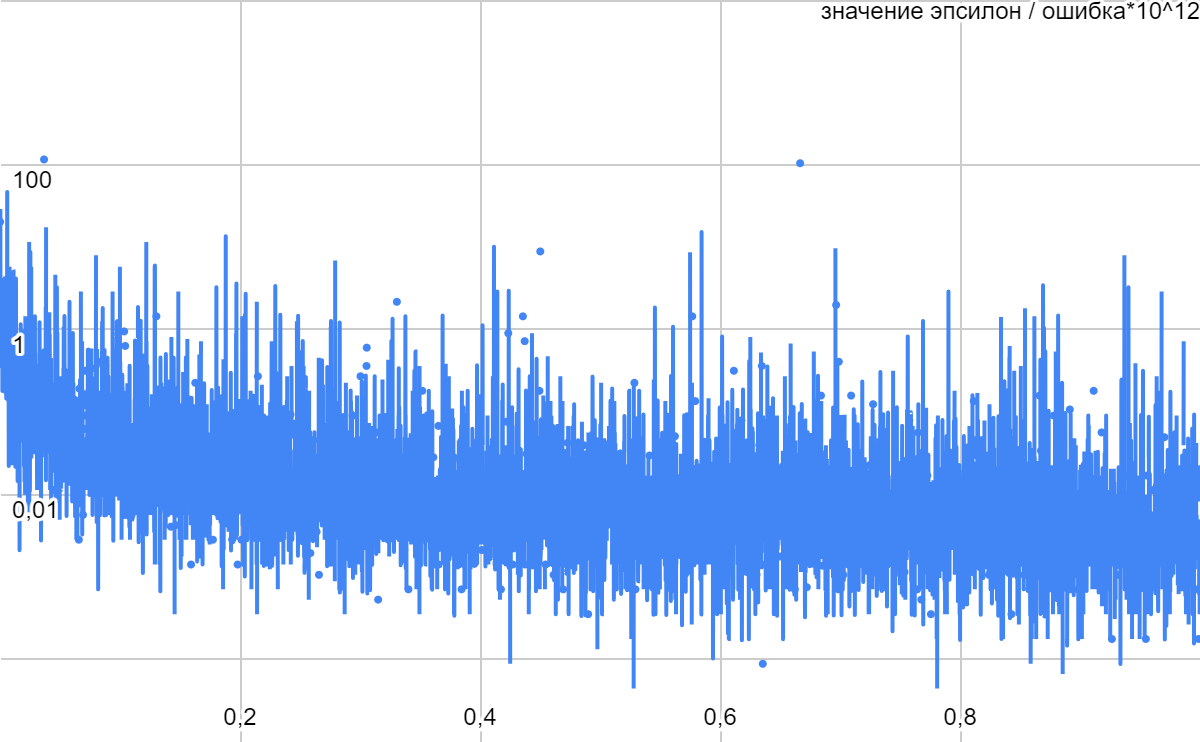


Рисунок 4. Изменение ошибки относительно изменения величины от до с шагом . Как видно из графика, при очень близких значениях ошибка больше, чем, когда близок к .

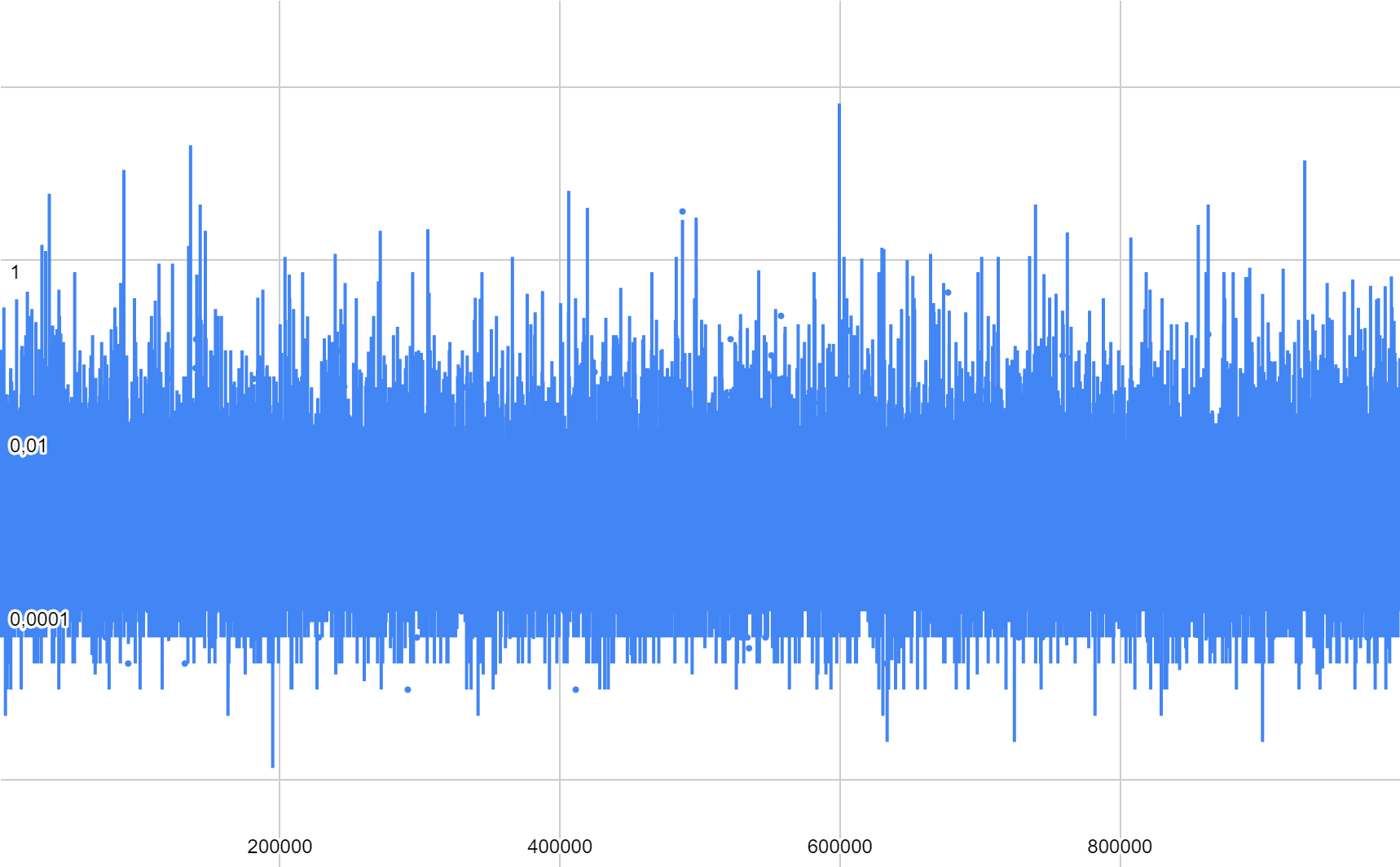


Рисунок 5. Изменение ошибки относительно изменения величины от 1 до с шагом . Как видно из графика, значение ошибки примерно находится в диапазоне от до .

Также были проведены дополнительные эксперименты, которые показали, что при вычислении в больших дробях время работы очень большое даже при сравнительно небольших задачах. Объяснить это можно большой вложенностью циклов. Алгоритм Жордана-Гаусса имеет асимптотику , где n - ранг матрицы, реализованное деление имеет асимптотику , где m - количество разрядов в числе. Уже выходит, что сложность алгоритма снизу оценивается , т.к на практике еще возникают большие константы.

# 

# **Заключение**

Подводя итог, проделанной работе, можно сделать следующие выводы:

1. Алгоритм Жордана-Гаусса работает корректно. Он имеет относительно небольшую погрешность при вычислениях в числах с плавающей точкой, что позволяет применять его на практике.
2. Реализованная длинная арифметика работает корректно. Использовать длинные дроби в алгоритме Жордана-Гаусса стоит только на матрицах небольшого ранга, в том случае, если требуется провести точные вычисления с ошибкой тождественно равной нулю.

# **Приложение**

template<class T>

void matrixProcessing<T>::GaussMethod()

{

for (int i = 0; i < std::min(A.getm(), A.getn()); i++)

{

int ind = -1;

for (int j = i; j < A.getn(); j++)

{

if (A[j][i] != \_\_ZERO && ind == -1 || ind != -1 && abs(A[j][i]) > abs(A[ind][i]))

{

ind = j;

}

}

if (ind == -1) continue;

if (ind != i)

{

A(i, 't', ind);

B(i, 't', ind);

}

for (int j = i + 1; j < A.getn(); j++)

{

T a = (A[j][i] / A[i][i]);

A(j, '-', i, a);

B(j, '-', i, a);

}

}

return;

}

template<class T>

void matrixProcessing<T>::JordanGaussMethod()

{

for (int i = 0; i < std::min(A.getm(), A.getn()); i++)

{

if (A[i][i] == \_\_ZERO)

{

int ind = -1;

for (int j = 0; j < A.getm(); j++)

{

if (A[i][j] != \_\_ZERO)

{

ind = j;

break;

}

}

if (ind == -1) continue;

for (int j = 0; j < A.getn(); j++)

{

if (j == i) continue;

T a = (A[j][ind] / A[i][ind]);

A(j, '-', i, a);

B(j, '-', i, a);

}

B(i, '/', A[i][ind]);

A(i, '/', A[i][ind]);

}

else

{

for (int j = 0; j < i; j++)

{

T a = (A[j][i] / A[i][i]);

A(j, '-', i, a);

B(j, '-', i, a);

}

B(i, '/', A[i][i]);

A(i, '/', A[i][i]);

}

}

checkSolution();

return;

}

# **Список литературы**

1. Ссылка на статью про невязку

<https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0> (дата обращения 05.05.2024)