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

**Отчет по лабораторной работе**

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

**Выполнил**:

студент группы 3823Б1ПМ1-1

Золкин И.А.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 9](#_Toc26962566)

[Результаты экспериментов 11](#_Toc26962567)

[Заключение 13](#_Toc26962568)

[Список литературы 19](#_Toc1)

[Приложение 20](#_Toc26962569)

# Постановка задачи

Программа, выполненная на языке программирования c++, должна корректно решать системы линейных уравнений методом Гаусса-Жордана.

В реализации необходимо использовать наследование, исключения, шаблоны, а также запрещается использовать std::vector<> из стандартной библиотеки.

# Метод решения

Метод Гаусса-Жордана предназначен для решения систем линейных алгебраических уравнений (СЛАУ). Он является модификацией метода Гаусса. Нередко этот метод называют методом Жордана или Жордана-Гаусса.

В данном методе решения СЛАУ мы работаем с расширенной матрицей системы. Преобразования, допустимые в методе Гаусса-Жордана те же, что и в методе Гаусса:

1. Смена мест двух строк.
2. Умножение всех элементов строки на некоторое число, не равное нулю.
3. Прибавление к элементам одной строки соответствующих элементов другой строки, умноженных на любой множитель.
4. Вычеркивание строки, все элементы которой равны нулю.

Можно менять местами и столбцы матрицы системы, но тогда необходимо запоминать новый порядок переменных в уравнениях. Например, смена мест первого и четвертого столбцов матрицы системы означает, что переменные и поменялись местами во всех уравнениях.

Опишем алгоритм, используемый в программе. На вход подается расширенная матрица размером n строк и m столбцов. Сортируем матрицу так, чтобы нулевые столбцы оказались в конце матрицы, запоминая при этом положение . Например:

Запускаем цикл от 0 до по . В каждой итерации цикла ищем в какой строке -го столбца находится максимальный не выбранный ранее элемент и выбираем его в качестве ведущего. Проверяем, не равняются ли все элементы ведущей строки кроме свободного члена нулевыми, в положительном случае система несовместна. Проходим внутренним циклом по не ведущим строкам, высчитываем коэффициент для зануления элементов строк в -м столбце, и прибавляем к элементам текущей строки соответствующие элементы ведущей строки, умноженные на коэффициент. Последним шагом будет обратная перестановка столбцов в правильном порядке. Рассмотрим действие алгоритма для частного случая, представленного выше.

До обратной транспозиции столбцов результаты из решенной Жорданом матрицы помещаются в массив следующим образом. Количество строк массива равняется количеству переменных. Каждый столбец — это столбец коэффициента, умноженного на независимый элемент, последний столбец представляет из себя столбец свободных членов. Для приведенного выше примера, массив выглядит следующим образом.

Вывод для данного примера выглядит следующим образом.

# Руководство пользователя

При запуске программы необходимо ввести цифру, обозначающую тип данных, с которым производятся планируемые вычисления. Всего пользователю представлено три типа данных: 1 – double, 2 – Complex, 3 – float. После выбора типа необходимо ввести количество строк системы n и количество столбцов расширенной матрицы системы m. После ввода n и m, следует вводить через пробел построчно элементы расширенной матрицы, вещественные числа вводятся через точку, для комплексных чисел вводят через пробел сначала действительную часть, потом мнимую (без i). На выходе при успешном выполнении программы пользователь получает решение системы линейных алгебраических уравнений, если же в процессе выполнения выяснилось, что система несовместна или обнаружились другие факторы, не позволяющие ее решить, то будет выведена соответствующее сообщение.

Пример работы с программой:

Необходимо решить систему 2x4 следующего вида.

Программа>

What type are we working with?:

1 - double

2 - complex

3 - float

Answer:

Пользователь>

1

2 4

0 1 3 4

0 2 4 6

Программа>

----------------------

x1 = 1\*t1

x2 = 1

x3 = 1

# Описание программной реализации

Каталог содержит несколько файлов, среди которых два файла “\*.cpp”, девять файлов “\*.hpp” и файл проекта “.sln”.

Программа начинает работу с функции main() в файле «**Jordano\_Gauss.cpp**». Внутри функции main() происходит выбор типа данных и вызов шаблонной функции root(double, double).

В функции *root*() если программа находится в режиме тестирования не корректность, запускаются циклы по m и n, создается объект matrix и заполняется случайными элементами функцией *inputRAND*(vector<vector<\_Mtrx>>&), где \_Mtrx – это выбранный тип данных (vector<vector<\_Mtrx>> дальше обозначен как MX). Создаются вспомогательные контейнеры: trans предназначен для хранения номеров столбцов сделанных транспозиций, indexes предназначен для хранения порядка номеров столбцов, ans нужен для хранения массива ответа и количества ведущих элементов (ранга системы). Вызывается функция решения матрицы *Solver*(MX&,vector<\_max\_el\_cord>,int,int,double,double,vector<std::pair<int,int>>), а после вызывается функция выделения ответа без вывода на экран *output\_s*(MX&, vector<\_max\_el\_cord>,n,m,vector<int>, vector<std::pair<int,int>>). Если в процессе решения системы оказалось, что она не совместна, то итерация цикла пропускается. После получения ответа вызывается функция *correct\_test*(MX&, std::pair<MX, size\_t>,double), проводящая тест на корректность программы.

Если программа не находится в режиме тестирования на корректность, то создаются вышеописанные контейнеры, производится получение данных от пользователя через функцию *input*(MX&), вызывается *Solver*(…) и *output*(…), принимающая те же аргументы, что и output\_s, но выводящая ответ на экран.

Файл **correct\_test.hpp**.

В файле находятся 4 функции: *correct\_test*(…), *inputRAND*(…), аргументы которые описаны выше и их реализации для типа данных Complex.

В *inputRAND*(…) используется std::mt19937 для генерации случайных чисел. Числа записываются в MX в циклах по строкам и столбцам.

Функция *correct\_test*(…) в цикле перемножает столбцы ответа на исходную матрицу, и проверяет результирующий столбец на то, что он нулевой.

Файл **utils.hpp**.

В файле находятся девять функций и одна структура.

Структура *\_max\_el\_cord* хранит номер строки str и номер столбца col.

Функция *print*(MX&) проходит циклом по всем элементам матрицы и выводит их в консоль.

Функция *numeration\_var*(char, int) возвращает строку из входного символа и цифры.

Функция *max\_cords*(MX&,int,vector<bool>&,double) проходит циклом по элементам заданного столбца матрицы и выводит номер строки, где тот максимальный и ранее не используемый.

Функция *Checker*(int,int,vector<\_Mtrx>&,double) проходит по заданной строке и возвращает есть ли там ненулевой элемент.

Функция *input*(MX&) считывает данные пользователя в MX&.

Функция *swap*(\_Mtrx&,\_Mtrx&) меняет элементы местами.

Функция *isNull\_col*(MX&,int,double) проходит циклом по элементам заданного столбца и определяет является ли столбец нулевым.

Функция *transpos*(MX&, double, vector<int>&) циклом ищет нулевые столбцы, и выполняет транспозицию их с последними ненулевыми.

Функция *distranspos*(MX&, vector<std::pair<int,int>>&) выполняет обратную транспозицию столбцов.

Файл **solver.hpp** содержит в себе одну функцию *Solver*(…). Сначала мы выполняем транспозицию нулевых столбцов и не нулевых, т.е. “сортируем” матрицу так, чтобы нулевые столбцы были в конце. Дальше в основном цикле мы проверяем систему на совместность и находим ведущий элемент. И если он не нулевой, то построчно высчитываем коэффициент для зануления элемента того же столбца, что и ведущего, но не той же строки и складываем элементы ведущей строки, умноженные на коэффициент с соответствующими элементами текущей строки.

Файл **outputter.hpp** содержит одну функцию *output*(…). В ней идем по номерам переменных, находим коэффициент для нормализации ведущего элемента (делим минус единицу на ведущий элемент) и помещаем в массив ответа элемент матрицы, умноженный на коэффициент. После проходим по массиву ответа и выводим соответствующие столбцам ответа элементы.

Файл **shadow\_outputter.hpp** имеет одну функцию *output\_s*(…). Реализация функции совпадает с реализацией функции output(…), за исключением вывода информации на экран, который здесь отсутствует.

Файлы **j\_error.cpp** и **j\_error.hpp** представляют собой реализацию и объявление класса собственного объекта ошибки, наследуется от std::exception. Имеет два конструктора, деструктор и переменную-указатель на массив char.

Файл **DEFINES.hpp** представляет собой файл с установленными значениями для подстановки препроцессором и одну extern переменную.

Файл **complex.hpp** содержит реализацию класса комплексных чисел *Complex*. Комплексное число представляется двумя переменными для действительной части double *real* и мнимой double *img*. Реализованы четыре конструктора, деструктор и перегружены следующие операции: +, - , /, \*, ==, >, <, >=, <=, !=, =. Также реализованы сетеры и гетеры на *real* и *img*, функция *module*(), возвращающая модуль, функция *angle*(), возвращающая угол. Также перегружены ввод и вывод комплексного числа и функция *abs*(..), возвращающая модуль.

# Подтверждение корректности

Для подтверждения корректности использовалась утверждение о том, что при перемножении любого столбца, задающего многообразие M(A,b), на матрицу системы получается нулевой столбец. Утверждение следует из другого схожего утверждения:

Рассмотрим на частном примере. Возьмем матрицу из примера выше.

Матрица с столбцами общего решения у нее выглядит следующим образом:

Проверим на корректность используя утверждение выше. Перемножим столбец и матрицу.

Получили нулевой столбец, следовательно, согласно утверждению, найденное решение верно.

Чтобы перевести программу в режим проверки корректности нужно изменить флаг CORRECT\_TEST в файле DEFINES.hpp с false на true. Проверку осуществляет функция correct\_test(…) в файле correct\_test.hpp путем поочередного умножения столбцов на матрицу системы.

Тестирование проводилось на числах из промежутка [-100000; 100000] для double и для действительной и мнимой части Comple, для float [-500; 500].

# Результаты экспериментов

Проводился эксперимент по выявлению точности. В качестве исследуемой величины была зависимость отклонения элементов в столбце в проверке корректности от нуля от количества переменных в системе.

Причина выбора исследуемой величины – при проверке метода решения СЛАУ на корректность производится определенное количество неточных вычислений, таких как произведение, сумма сильно различающихся чисел. В связи с этим возникает предположение, что, чем больше переменных в системе, тем больше дает погрешность проверка на корректность.

Ниже представлены результаты замеров. На графиках изображена зависимость ошибки от x, где x – количество переменных в системе.

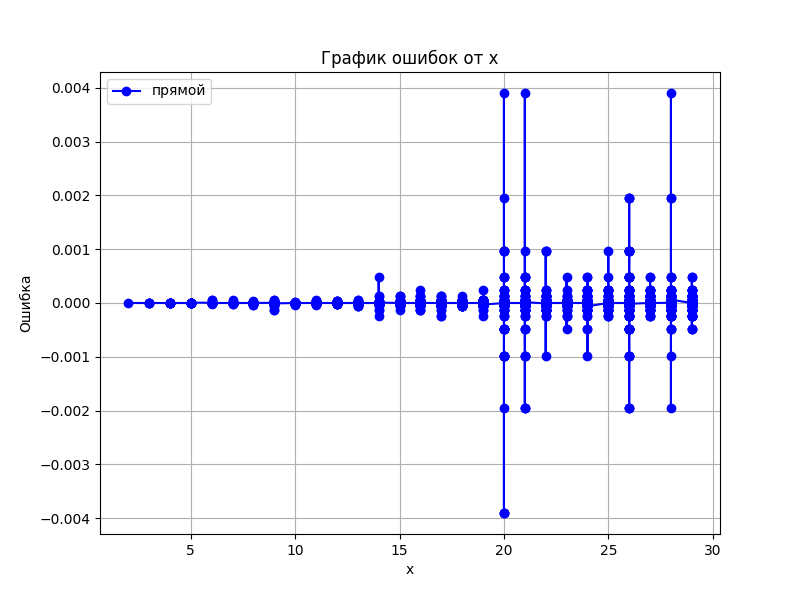


Рис. 1 - ошибка от x при float

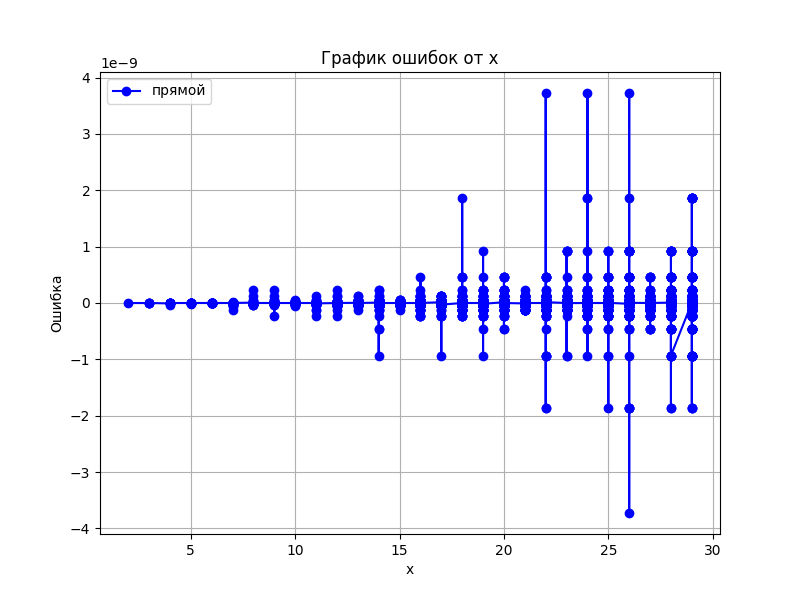


Рис. 2 - ошибка от x при double

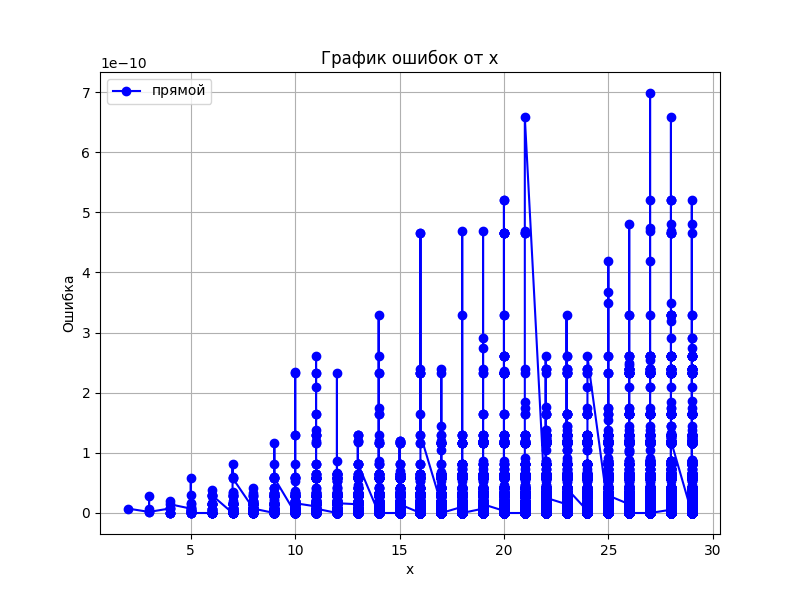


Рис. 3 - ошибка от x при Complex

Как видно из графиков, отклонение элементов от нуля растет с увеличением количества переменных в системе. Это обусловлено увеличением количества неточных операций не только при решении системы, но и при проверке корректности и, как следствие, увеличение погрешности вычислений элементов нулевого вектора. Опираясь на полученные результаты, проанализируем данные на промежутках .

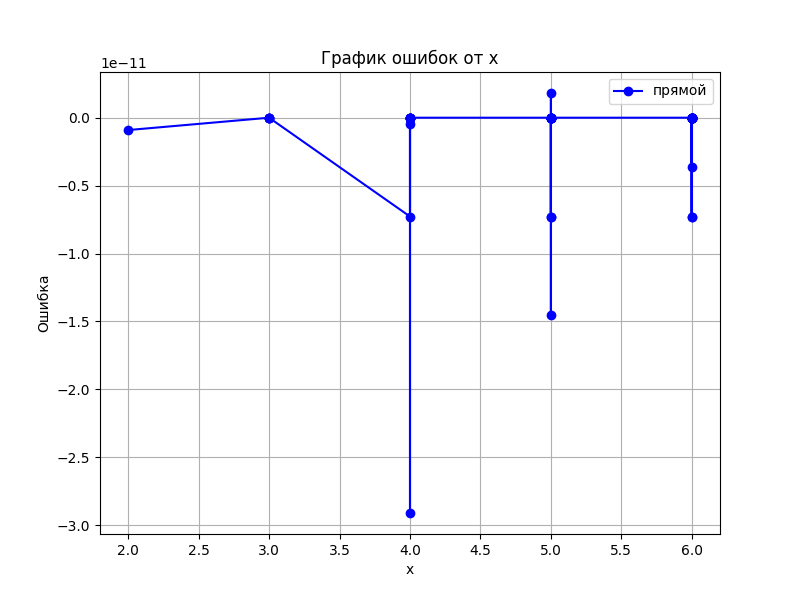


Рис. 4 - ошибка от x (0;7) на double

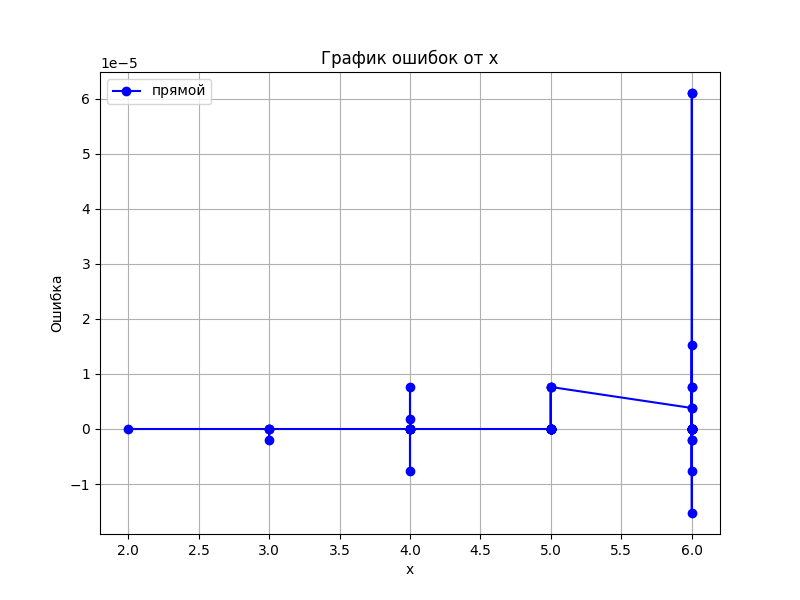


Рис. 5 - ошибка от x (0,7) на float

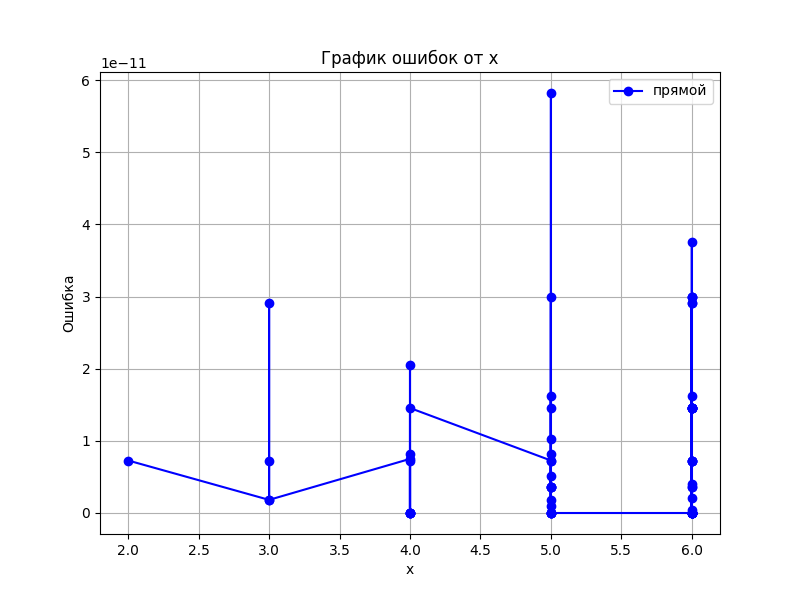


Рис. 6 - ошибка от x (0;7) на Complex

Из результатов замеров на промежутке (0; 7) видно значительное улучшение точности. По полученным данным, к сожалению, реальную точность на тестируемых данных определить нельзя, т.к. при большом x на полученную точность оказывает большое влияние сам метод проверки, а при малом x мы не можем в достаточной мере предсказать погрешности при больших x.

Исходя из этого, делаем вывод, что для исследуемых промежутков и значений можем определить приближенную верхнюю границу и нижнюю границу погрешности вычислений:

* Double – от 3e-11 до 4e-9.
* Float – от 6e-5 до 4e-3.
* Complex – от 6e-11 до 7e-10.

# Заключение

Реализованный метод решения систем линейных алгебраических уравнений Жордана-Гаусса работает корректно в пределах погрешности. Комплексные числа при вычислениях дают верные результаты в пределах погрешности используемого для представления мнимой и действительной части типа double. Метод Жордана можно использовать при не большом количестве переменных x < 7, где x – кол-во переменных, следовательно, при небольшом ранге. Были получены верхние и нижние границы погрешности для вычислений до 30 переменных. Определить точную погрешность метода не удалось, т.к. на вектор проверки оказывает влияние неточности вычислений самого вектора проверки.

# Список литературы

1. Любимцев О.В., Золотых Н. Ю. Необходимые требования к успешному освоению дисциплины “Алгебра” (минимально необходимый уровень) : учебно-методическое пособие / ННГУ им. Н. И. Лобачевского. – Нижний Новгород : Изд-во ННГУ, 2019. – 86 с. – Текст : электронный.

# Приложение

template<class \_Mtrx>

vector<int> Solver(vector<vector<\_Mtrx>>& matrix, vector<\_max\_el\_cord>& main\_elements, int n, int m, double EPS,double INF,vector<std::pair<int,int>>& trans) {

try {

\_Mtrx NULL\_EL = \_Mtrx(0);

\_Mtrx coef;

vector<bool> used(n, false);

vector<int> indexes(m - 1);

for (int i = 0; i < m - 1; i++) indexes[i] = i;

trans = transpos(matrix,EPS,indexes);

for (int i = 0; i < n && i < m; ++i) {

int max\_el\_pos = max\_cords(matrix, i, used,INF);

if (max\_el\_pos == -1) continue;

if ((!Checker(0, m - 1, matrix[max\_el\_pos], EPS)) && abs(matrix[max\_el\_pos][m - 1]) > \_Mtrx(EPS)) {

CORRECT\_TEST\_NEED = false;

throw j\_error("Inconsistent system");

}

if (abs(matrix[max\_el\_pos][i]) < \_Mtrx(EPS)) continue;

main\_elements.push\_back({ max\_el\_pos,i });

for (int k = 0; k < n; k++) { //Идем по строкам

if (k == max\_el\_pos) continue;

coef = -matrix[k][i] / matrix[max\_el\_pos][i];

matrix[k][i] = NULL\_EL;

for (int j = i + 1; j < m; j++) { //считаем текущую строку

matrix[k][j] = matrix[k][j] + matrix[max\_el\_pos][j] \* coef;

if (abs(matrix[k][j]) < \_Mtrx(EPS)) matrix[k][j] = NULL\_EL;

}

if ((!Checker(0, m - 1, matrix[k],EPS)) && abs(matrix[k][m - 1]) > \_Mtrx(EPS)) {

CORRECT\_TEST\_NEED = false;

throw j\_error("Inconsistent system");

}

}

}

for (int i = std::min(n, m); i < n; i++) {

if ((!Checker(0, m - 1, matrix[i],EPS)) && abs(matrix[i][m - 1]) > \_Mtrx(EPS)) {

CORRECT\_TEST\_NEED = false;

throw j\_error("Inconsistent system");

}

}

return indexes;

}

catch (std::exception& e) {

if (!CORRECT\_TEST) { std::cout << e.what() << '\n'; exit(ERR\_SYS\_1); }

}

catch (...) {

std::cout << "unnamed error\n";

exit(ERR\_UNNAMED);

}

};