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 Жордана-Гаусса»**

**Выполнил**:

студент группы 3823Б1ПМ1-1

Алемаев М.В.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 3](#_Toc165918736)

[Метод решения 4](#_Toc165918737)

[Руководство пользователя 6](#_Toc165918738)

[Описание программной реализации 7](#_Toc165918739)

[Подтверждение корректности 9](#_Toc165918740)

[Результаты экспериментов 10](#_Toc165918741)

[Заключение 13](#_Toc165918742)

[Приложение 14](#_Toc165918743)

[Литература 17](#_Toc165918744)

# Постановка задачи

В рамках лабораторной работы требовалось написать метод Жордана-Гаусса для матриц произвольного размера.

Необходимо в рамках лабораторной работы:

1. Написать реализацию метода Жордана-Гаусса
2. Включить в программу следующие принципы: шаблоны, наследование и исключение
3. Проверить метод на корректность работы

# Метод решения

* **Алгоритм Жордана-Гаусса**

1. Выбираем первый слева столбец матрицы, в котором есть хоть одно отличное от нуля значение.
2. Находим в нем максимальный по модулю элемент и меняем местами строку, в которой он на находится, и первую. Такой элемент будем называть ведущим.
3. Из остальных строк вычитают первую строку, умноженную отношение первого элемента текущей строки к ведущему.
4. Далее проводим такую же процедуру с матрицей, получающейся из исходной матрицы после вычёркивания первой строки и первого столбца, и так далее, пока не наткнемся на последний столбец или строку.
5. В каждой строке делим её элементы на ведущий.

В качестве ведущего будем брать максимальный по модулю элемент данного столбца текущей подматрицы, в целях минимизировать погрешность.

Нормализовать матрицу будем лишь в конце, дабы не совершать лишних операций и не увеличивать погрешность.

В результате работы алгоритма мы получим матрицу простейшего (или ступенчато-приведенного вида), где \* - произвольное число, значение которого нас не интересует.

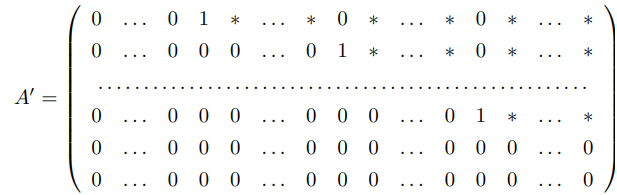


Рис. . Простейший вид матрицы

Предположительно асимптотическая сложность такого алгоритма – O(n\*m\*min(n,m)), т.к. в каждой итерации алгоритм проходится по каждой строке и каждому столбцу, а всего итераций будет min(n,m).

n – число строк, m – число столбцов.

* **Вывод ответа**

Ответ выводится в следующем формате:

*x1 = a1 + k1,pt1 + k1,p+1t2 + … + k1,ptm-n+1*

*x2 = a2 + k2,pt1 + k2,p+1t2 + … + k2,ptm-n+1*

*…*

*xr = ar + kr,pt1 + kr,p+1t2 + … + kr,ptm-n+1* ,

где *ai –* значение элемента, столбца свободных элементов расширенной матрицы; *t1*, …, *tn-m+1* – свободные коэффициенты; *ki,j* – коэффициент стоящий на пересечении i-ой строки и j-го столбца (не обязательно идут подряд).

# Руководство пользователя

Класс *Sovler* содержит несколько методов, которые могут быть использованы для работы с матрицей:

1. GordanGauss(): Приводит матрицу к ступенчато-приведенному виду.
2. answer(): Выводит ответ.
3. check\_ans(): Проверяет ответ на корректность.
4. get\_max(int str, int col): находит максимальный элемент столбца *col*, начиная со строки *str* если же все элементы равны нулю, выводит -1.
5. сonversion(int str, int col): Вычитает из элементов остальных строк элементы строки *str*, начиная со столбца *col*.
6. Normalize(int str, int col): делит все элементы строки *str*, на элемент в столбце *col*.
7. Check\_coop(): Проверяет совместность получившейся СЛОУ.

Методы 4-7 являются по большей части вспомогательными для первых трех, но также могут быть использованы в качестве методов класса.

# Описание программной реализации

Создаем массив указателей на массив - *mat*. Помимо этого, создаем его копию - *orig*. Чтобы потом подставив значения из *mas* в *orig* проверить программу на корректность.

Двумерный массив *mat* представляет из себя расширенную матрицу, где последний столбец является столбцом свободных коэффициентов.

Также мы создаем два динамических массива *free* и *depend*, где будут находится индексы свободных и зависимых переменных соответственно.

Опишем программную реализацию трех основных методов.

* **JordanGauss**

Полный алгоритм работы метода был описан ранее (см. Методы решения)

GordanGauss()

1. for (i=0, j=0; i<n && j<m-1; j++)

2. str = get\_max(I,j)

3. if (str==-1) j++

4. else

5. swap(mat[str], mat[i])

6. i++

7. //Прописываем normalize для всех ведущих элементов

* **Answer**

Находим ведущий элемент каждой строки и выводим его в виде суммы свободных коэффициентов.

Answer()

1. for (i=0; i<depend.size(); i++)

2. // Выводим «x{i} = {mat[i][m-1]}»

3. for (j=0; j<free.size(); j++)

4. if (abs(mas[i][free[j]]>0)

5. print(« + {соответствующий коэффициент} \* x{free[j]}»)

* **Check\_ans**

Вручную вводятся значения свободных коэффициентов, затем исходя из них присваиваются значения зависимых. Подставляем их значения в массив *values*. И считаем разницу с соответствующим значением последнего столбца.

Check\_ans()

1. for (i = 0; i < free.size(); i++)

2. // Вводим значения

3. for (i = 0; i < depend.size(); i++)

4. values[depend[i]] = mat[i][m-1]

5. for (j = 0; j < free.size(); j++)

6. values[depend[i]] -= mat[i][free[j]]\*values[free[j]]

7. dif = 0

8. for (i = 0; i < n; i++)

9. s = 0

10. for (j=0;j<m-1;j++)

11. s+= values[j] \* orig[i][j]

12. dif = abs(s – orig[i][m-1])

# Подтверждение корректности

Чтобы проверить ответ на корректность нужно поставить полученные значения в исходную матрицу. Тогда значения свободных коэффициентов мы выбираем произвольно, а значения зависимых будут меняться исходя из значений свободных. Что будет равносильно умножению на столбец частных решений Ax**’ =** 0.

Будем вычитать из последнего столбца исходной матрицы полученное при подстановке значение. Если СЛОУ решена верно, то погрешность будет на уровне погрешности вычисления используемого типа данных.

При этом не стоит забывать что система может быть не совместна и в таком случае последние два пункта не имеют смысла. Система является несовместной, если строка, в которой все элементы нулевые равна ненулевому элементу столбца свободных коэффициентов.

# Результаты экспериментов

Чтобы подтвердить точность асимптотики количество тактов процессора за время выполнения метода *JordanGauss* было поделено на его предполагаемую асимптотическую сложность – O(n\*m\*min(n,m)), т.к. во всех случаях мы получаем константу, то предположительная сложность оказалась верна.

Рассмотрим различные конфигурации:

1. *n –* фиксировано, m .
2. *m –* фиксировано, n .
3. *m ,* *n .*

*n* – число строк, *m* – число столбцов.

При этом константа в данных конфигурациях будет различна, т. к. при проходе по вертикали мы просто обновляем максимальный элемент, а когда идем по горизонтали, то совершаем много арифметических действий. И во втором случае константа, стоит полагать, будет больше.

Для подтверждения данных слов ниже приведены графики каждой из конфигураций.

В первом случае *n* фиксировано и равно 10, во втором аналогично *m.*

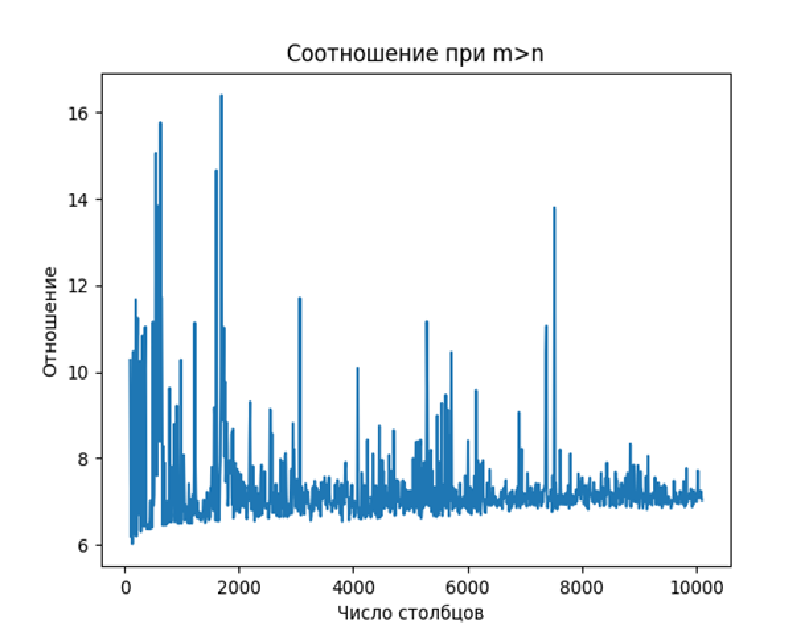
Исходя из графиков можно сделать следующие выводы:

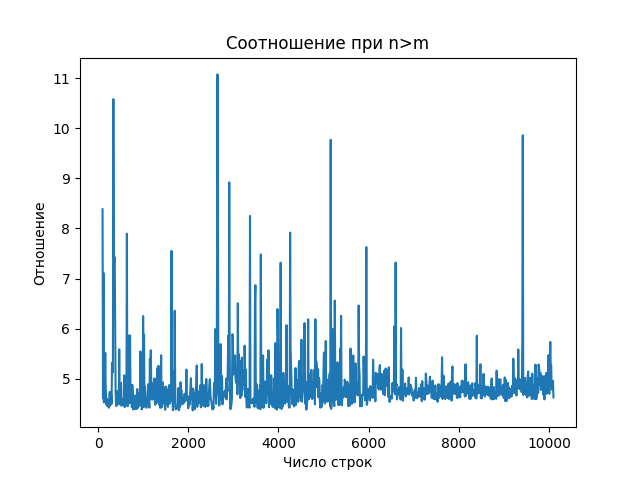
* Константа для *(1)* действительно самая большая. C = 7.5
* Для *(2)* C = 5
* Для *(3)* константа оказалась самой маленькой. С = 3.8

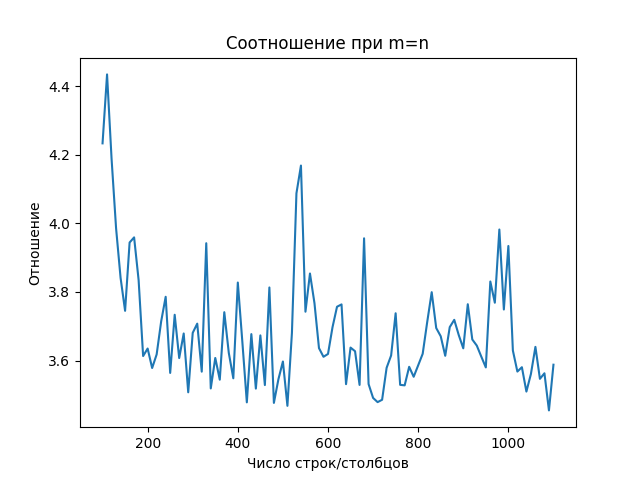
При этом понятно, что если в первых двух случаях изменить вторую фиксированную переменную, то константа измениться.

Также случаи 1-2 и 3 не эквивалентны, т.к. n3 ≠ O(n\*m\*min(n,m)), при фиксированном *m* поэтому сравнивать их бессмысленно.

*= =*







# Заключение

В рамках лабораторной работы:

* был создан класс Матриц
* в классе был реализован метод Жордана-Гаусса
* был описан алгоритм работы и подтверждено предположение об асимптотической сложности
* были использованы: наследования, шаблоны, исключения
* был написан метод для проверки корректности работы алгоритма.

# Приложение

template <class t>

class Solver {

const t eps = pow(10, -10);

int n, m;

bool coop;

Vector<int> depend;

Vector<int> free;

Matrix<t> orig;

Matrix<t> mat;

public:

Solver(int n, int m) : depend(), free(), mat(n, m), orig(n, m) {

this->n = n;

this->m = m;

}

t& operator() (int i, int j) {

return mat(i, j);

}

int get\_max(int str, int col) {

t mx = eps;

int ind = -1;

for (int i = str; i < n; i++) {

if (abs(mat(i,col)) > mx) {

mx = abs(mat(i, col));

ind = i;

}

}

return ind;

}

void conversion(int str, int col) {

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

t c = mat(i,col) / mat(str,col);

if (i != str && abs(c) > eps)

for (int j = col; j < m; j++)

mat(i,j) = mat(i,j) - mat(str,j) \* c;

}

}

void normalize(int str, int col) {

t div = mat(str,col);

for (int j = col; j < m; j++)

mat(str,j) = mat(str,j) / div;

}

void check\_coop() {

coop = 1;

for (int i = depend.getsize(); i < n; i++)

if (abs(mat(i,m - 1)) > eps) {

mat.print\_mat();

throw Cooperative\_exception<t>("Система несовместна", i + 1, mat(i,m - 1));

}

}

void GordanGauss() {

depend.clean();

free.clean();

orig = mat;

int j = 0;

for (int i = 0; j < m - 1 && i < n; j++) {

int str = get\_max(i, j);

if (str != -1) {

mat.swap(str, i);

conversion(i, j);

depend.push\_back(j);

i++;

}

else

free.push\_back(j);

}

for (; j < m - 1; j++)

free.push\_back(j);

for (int i = 0; i < depend.getsize(); i++)

if (mat(i,depend[i]) != 0)

normalize(i, depend[i]);

check\_coop();

}

void answer() {

if (depend.getsize() + free.getsize() == 0)

throw Reduce\_exception("Система не приведена");

for (int i = 0; i < depend.getsize(); i++) {

std::cout << 'x' << depend[i] + 1 << " = " << mat(i,m - 1);

for (int j = 0; j < free.getsize(); j++) {

int c = free[j];

if (abs(mat(i, c)) > eps) {

std::cout << " + " << mat(i, c) << " \* t" << j + 1;

}

}

std::cout << '\n';

}

for (int i = 0; i < free.getsize(); i++) {

std::cout << 'x' << free[i] + 1 << " = " << 't' << i+1 << '\n';

}

}

void check\_ans() {

if (depend.getsize() + free.getsize() == 0)

throw Reduce\_exception("Система не приведена");

Vector<t> values(free.getsize() + depend.getsize());

if (free.getsize() > 0) {

std::cout << "Введите значения свободных переменных\n";

for (int i = 0; i < free.getsize(); i++) {

std::cout << 't' << i + 1 << " = ";

int ind = free[i];

std::cin >> values[ind];

}

}

for (int i = 0; i < depend.getsize(); i++) {

values[depend[i]] = mat(i, m - 1);

for (int j = 0; j < free.getsize(); j++) {

int c = free[j];

values[depend[i]] -= mat(i, c) \* values[c];

}

}

t dif = 0;

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

t s = 0;

for (int j = 0; j < values.getsize(); j++) {

t val = orig(i, j);

s += values[j] \* val;

}

dif += abs(s - orig(i, m - 1));

}

std::cout << "Погрешность вычисления - " << dif << '\n';

}

};

# Литература

1. http://www.uic.unn.ru/~zny/algebra/lectures/lectures/10\_LinearSystems.pdf