**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №10

«Вычисление обратной матрицы методом Гаусса-Жордана.»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-13Б |  | преподаватель каф. ИУ5 |
| Поляков Данила |  | Аксёнова М.В. |
| Подпись и дата |  | Подпись и дата: |

Москва, 2021 г.

**Постановка задачи**

Создать функцию для вычисления обратной матрицы по методу Гаусса-Жордана. Размер матрицы передавать в функцию в качестве параметра. Для упрощения алгоритма следует присоединить единичную матрицу справа к исходной и выполнять все преобразования над объединенной матрицей размером N\*2N. Обратная матрица получится на месте единичной в столбцах N…2N, а на месте исходной матрицы в столбцах 0…(N-1) должна получиться единичная матрица.

Включить в алгоритм проверку на существование обратной матрицы. Для этого в в прямом ходе перед делением выполнить проверку на ноль элементов главной диагонали исходной матрицы. Если элемент равен 0, то нужно поменять местами текущую строку с одной из нижележащих строк, в которой элемент в соответствующем столбце не равен 0.

Если таких строк нет, то выдать сообщение: «Обратная матрица не существует».

Применить функцию для решения системы линейных алгебраических уравнений.

Шаги выполнения задания.

Часть 1.

* 1. Разработать схему алгоритма для вычисления обратной матрицы методом Гаусса-Жордана.
  2. Разработать программу, реализующую метод Гаусса-Жордана. Для ускорения процесса разработки программы на этом шаге следует сконцентрироваться на реализации алгоритма и не использовать динамическую память для хранения матриц и не оформлять реализацию алгоритма в виде функции.
  3. Отладить программу, используя контрольный пример.
  4. Оформить алгоритм в виде функции для вычисления обратной матрицы по методу Гаусса-Жордана. Размер матрицы передавать в функцию в качестве параметра.

Часть 2.

* 1. Выполнить тестирование программы. Проверку правильности результатов вычислений выполнять путем умножения полученной обратной матрицы на исходную (в результате должна получиться единичная матрица). Для реализации контроля разработайте функцию, реализующую произведение прямоугольных матриц. В процессе выполнения программы на экран должы выводиться следующие данные: исходная матрица, промежуточные результаты после прямого и обратного ходов, обратная матрица и результат умножения прямой матрицы на обратную. Для печати матрицы использовать функцию из лабораторной работы 10.
  2. Доработать алгоритм для случая, когда на главной диагонали появляется ноль. Включить в алгоритм проверку на существование обратной матрицы.
  3. Применить функцию вычисления обратной матрицы для решения системы линейных алгебраических уравнений.

**Разработка алгоритма**

**Описание алгоритма**

Данный алгоритм находит обратную матрицу для заданной квадратной методом Гаусса-Жордана. После присоединения справа единичной матрицы и обработки получившейся новой расширенной матрицы, получаем по правую сторону от середины искомую обратную изначальной матрицы, а слева – единичную матрицу. После алгоритм перемножает получившуюся матрицу с изначальной, убеждаясь в правильности расчёта, при которой в результате получится единичная матрица.

**Описание используемых переменных**

SIZE – константа, обозначающая размер всех используемых квадратных матриц

double \*\*originalMatrix – изначальная матрица, заполненная случайными числами из отрезка [0,10].

double \*\*matrix – скопированная изначальная матрица, с которой потом будут производиться манипуляции. Используется для того, чтобы не испортить изначальную, поскольку в будущем она будет перемножаться для проверки результата.

double \*\*reversedMatrix – переменная для хранения обратной матрицы, получившейся после использования заданного метода.

double \*\*resultMatrix – матрица, хранящая результат перемножения изначальной и найденной обратной матрицы.

void printMatrix(double \*\*matrix, size\_t N) – функция вывода матрицы размером NxN на экран.

bool isMatrixHasReversed(double \*\*matrix, size\_t N) – функция, определяющая, существует ли у данной матрицы обратная. В случае нахождении на основной диагонали нуля, меняет местами строку с нижестоящей, той, у которой в искомой позиции находится число, отличное от нуля. Если такое невозможно, возвращает значение false.

bool gaussJordan(double \*\*matrix, double \*\*reversed, size\_t N) – функция нахождения обратной матрицы методом Гаусса-Жордана, сохраняет получившуюся матрицу в переменную reversed. Возвращает значение в зависимости от возможности нахождения обратной матрицы.

void multiplyMatrixes(double \*\*matrix1, double \*\*matrix2, double \*\*resultMatrix, size\_t N) – функция перемножения двух квадратных матриц размером NxN, результат записывается в переменную resultMatrix.

**Текст программы**

**Листинг кода программы:**

**main.cpp**

#include "functions.hpp"

int main() {

double \*\*matrixOriginal = new double \*[SIZE];

for (size\_t i = 0; i < SIZE; ++i) {

matrixOriginal[i] = new double[SIZE];

}

generateRandom(matrixOriginal, SIZE);

double \*\*matrix = new double \*[SIZE];

for (size\_t i = 0; i < SIZE; ++i) {

matrix[i] = new double[SIZE];

}

double \*\*reversedMatrix = new double \*[SIZE];

for (size\_t i = 0; i < SIZE; ++i) {

reversedMatrix[i] = new double[SIZE];

}

double \*\*resultMatrix = new double \*[SIZE];

for (size\_t i = 0; i < SIZE; ++i) {

resultMatrix[i] = new double[SIZE];

}

for (size\_t i = 0; i < SIZE; ++i) {

for (size\_t j = 0; j < SIZE; ++j) {

matrix[i][j] = matrixOriginal[i][j];

}

}

printMatrix(matrixOriginal, SIZE);

if (!gaussJordan(matrix, reversedMatrix, SIZE)) {

std::cout << "Обратная матрица не существует";

} else {

multiplyMatrixes(matrixOriginal, reversedMatrix, resultMatrix, SIZE);

printMatrix(reversedMatrix, SIZE);

printMatrix(resultMatrix, SIZE);

}

for (size\_t i = 0; i < SIZE; ++i) {

delete[] matrix[i];

delete[] reversedMatrix[i];

delete[] matrixOriginal[i];

delete[] resultMatrix[i];

}

delete[] matrix;

delete[] reversedMatrix;

delete[] matrixOriginal;

delete[] resultMatrix;

return 0;

}

**functions.hpp**

#ifndef CODE\_FUNCTIONS\_HPP

#define CODE\_FUNCTIONS\_HPP

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <stdio.h>

#include <time.h>

#include <math.h>

#define SIZE 4

bool isMatrixHasReversed(double \*\*, size\_t);

bool gaussJordan(double \*\*, double \*\*, size\_t);

void generateRandom(double \*\*, size\_t);

void printMatrix(double \*\*, size\_t);

void multiplyMatrixes(double \*\*, double \*\*, double \*\*, size\_t);

#endif //CODE\_FUNCTIONS\_HPP

**functions.cpp**

// Copyright 2021 qqq <polyakovdd@student.bmstu.ru>

#include "functions.hpp"

void printMatrix(double \*\*matrix, size\_t N) {

for (size\_t i = 0; i < N; ++i) {

for (size\_t j = 0; j < N; ++j) {

std::cout << std::setw(12) << std::setprecision(5) << std::fixed << matrix[i][j];

}

std::cout << std::endl;

std::cout << std::endl;

}

std::cout << std::endl;

std::cout << std::endl;

}

void multiplyMatrixes(double \*\*matrix1, double \*\*matrix2, double \*\*resultMatrix, size\_t N) {

for (size\_t i = 0; i < N; ++i) {

for (size\_t j = 0; j < N; ++j) {

resultMatrix[i][j] = 0;

}

}

for (size\_t i = 0; i < N; ++i) {

for (size\_t j = 0; j < N; ++j) {

for (size\_t k = 0; k < N; ++k) {

resultMatrix[i][j] += matrix1[i][k] \* matrix2[k][j];

}

resultMatrix[i][j] = abs(resultMatrix[i][j]);

}

}

}

bool isMatrixHasReversed(double \*\*matrix, size\_t N) {

bool result = true;

for (size\_t i = 0; i < N; ++i) {

if (matrix[i][i] == 0) {

result = false;

for (size\_t j = i; j < N; ++j) {

if (matrix[j][i] != 0) {

for (size\_t k = 0; k < N; ++k) {

std::swap(matrix[j][k], matrix[i][k]);

}

result = true;

continue;

}

}

if (!result) {

return result;

}

}

}

return true;

}

void generateRandom(double \*\*matrix, size\_t N) {

srand(time(nullptr));

for (size\_t i = 0; i < N; ++i) {

for (size\_t j = 0; j < N; ++j) {

matrix[i][j] = rand() % 10;

}

}

}

bool gaussJordan(double \*\*matrix, double \*\*reversed, size\_t N) {

if (!isMatrixHasReversed(matrix, N)) {

return false;

} else {

double \*\*unoMatrix = new double \*[N];

for (size\_t i = 0; i < N; ++i) {

unoMatrix[i] = new double[N];

}

//Единичная матрица (искомая обратная матрица)

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

for (size\_t j = 0; j < N; ++j) {

if (i == j) {

unoMatrix[i][j] = 1;

} else {

unoMatrix[i][j] = 0;

}

}

double \*\*bigMatrix = new double \*[N];

for (size\_t i = 0; i < N; ++i) {

bigMatrix[i] = new double[N \* 2];

}

//Общая матрица, получаемая скреплением Начальной матрицы и единичной

for (size\_t i = 0; i < N; ++i) {

for (size\_t j = 0; j < N; ++j) {

bigMatrix[i][j] = matrix[i][j];

bigMatrix[i][j + N] = unoMatrix[i][j];

}

}

for (size\_t k = 0; k < N; k++) //k-номер строки

{

for (size\_t i = 0; i < 2 \* N; i++) //i-номер столбца

bigMatrix[k][i] = bigMatrix[k][i] /

matrix[k][k]; //Деление k-строки на первый член !=0 для преобразования его в единицу

for (size\_t i = k + 1; i < N; i++) //i-номер следующей строки после k

{

double q = bigMatrix[i][k] / bigMatrix[k][k]; //Коэффициент

for (size\_t j = 0; j < 2 \* N; j++) //j-номер столбца следующей строки после k

bigMatrix[i][j] = bigMatrix[i][j] - bigMatrix[k][j] \*

q; //Зануление элементов матрицы ниже первого члена, преобразованного в единицу

}

for (size\_t i = 0; i < N; i++) //Обновление, внесение изменений в начальную матрицу

for (size\_t j = 0; j < N; j++)

matrix[i][j] = bigMatrix[i][j];

}

for (int k = N - 1; k > -1; k--) //k-номер строки

{

for (int i = 2 \* N - 1; i > -1; i--) //i-номер столбца

bigMatrix[k][i] = bigMatrix[k][i] / matrix[k][k];

for (int i = k - 1; i > -1; i--) //i-номер следующей строки после k

{

double q = bigMatrix[i][k] / bigMatrix[k][k];

for (int j = 2 \* N - 1; j > -1; j--) //j-номер столбца следующей строки после k

bigMatrix[i][j] = bigMatrix[i][j] - bigMatrix[k][j] \* q;

}

}

for (size\_t i = 0; i < N; i++)

for (size\_t j = 0; j < N; j++)

reversed[i][j] = bigMatrix[i][j + N];

for (size\_t i = 0; i < N; ++i) {

delete[] bigMatrix[i];

delete[] unoMatrix[i];

}

delete[] bigMatrix;

delete[] unoMatrix;

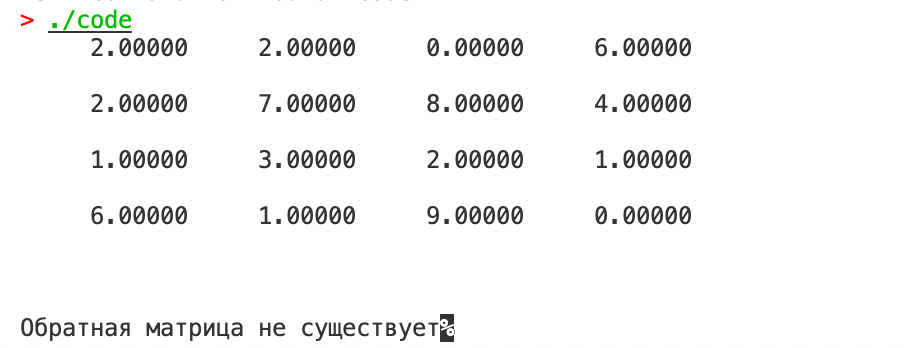
return true;

}

}

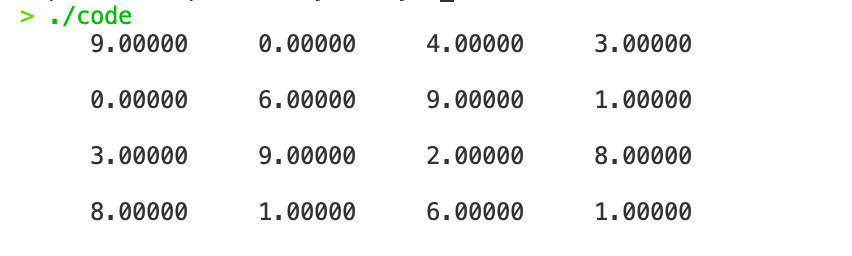
**Анализ результатов**

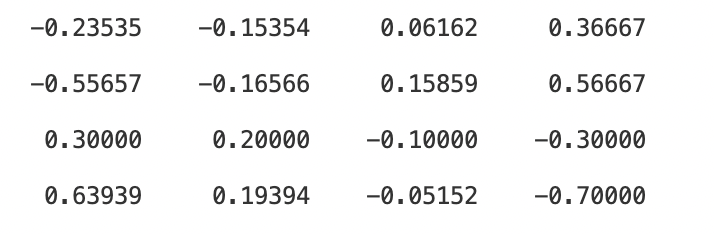
**Случай, когда обратной матрицы не существует**



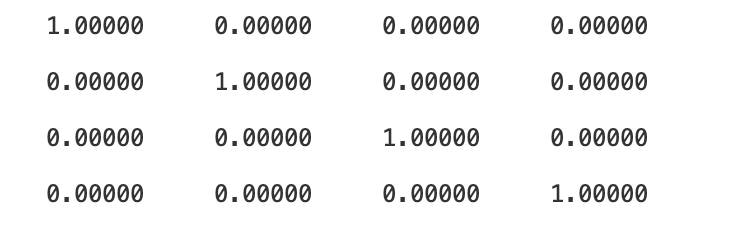
**Стандартный случай**

Сгенерированная матрица:



Полученная обратная матрица:  


Результат перемножения:



Единичная матрица, следствие верной работы алгоритма.