**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №10

«**Вычисление обратной матрицы методом Гаусса-Жордана**»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнил:** | |  | **Принял:** | |
| ФИО: | \_Цыпышев Т. А.\_\_\_\_\_ |  | ФИО: | \_**Самохвалов А. Э.**\_\_\_ |
| Группа: | \_ИУ5-11Б\_\_\_\_\_\_\_\_\_\_ |  | Должность: | \_Преподаватель\_\_\_\_ |
| Дата: | \_28.11.2022\_\_\_\_\_\_\_\_\_ |  | Дата: | \_28.11.2022\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2022 г.

**Постановка задачи**

**Цели работы:**

- разработка программы, реализующей алгоритм вычисления обратной матрицы методом Гаусса-Жордана

- использование разработанных функций для решения систем линейных уравнений

**Задание:**

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 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: «The matrix does not have an inverse matrix».

Применить функцию для решения системы линейных алгебраических уравнений.

**Разработка алгоритма**

**Шаги выполнения задания.**

* 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.

**Входные данные:**

* int N – размер матрицы
* double\*\* A - матрица

**Выходные данные:**

* double det – определитель матрица A
* double\*\* A\_reverse – обратная матрица по отношению к A

**Функции:**

* double \*\*CreateIdentityMatrix(int N) – возвращает единичную матрицу размером N\*N
* double\*\* CreateZeroMatrix(int N) – возвращает нулевую матрицу размером N\*N
* double \*\*CloneMatrix(double \*\*A, int N) – создаёт копию матрицы А
* void PrintMatrix(double \*\*arr, int N) – выводит матрицу в консоль
* double \*\*GenerateRandomMatrix(int N) – создает случайную матрицу размером N\*N
* double \*\*ReadMatrix(const char \*filename, int N) – считывает матрицу из файла
* double \*\*FragmentMatrix(double \*\*A, int N, int i, int j) – создаёт копию матрицу А без строки i и столбца j
* double Determinant(double \*\*arr, int N) – считает определитель матрицы методом разложения по строке
* double \*\*Inverse(double \*\*A, int N) – создает обратную матрицу
* double \*\*Multiple(double \*\*A, double \*\*B, int N) – умножает матрицу A на матрицу B

**Текст программы**

**Main.cpp**

/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>

#include <iostream>

#include <cmath>

#include "Header.h"

using namespace std;

double \*\*FragmentMatrix(double \*\*A, int N, int i, int j) { // i and j - row and column to be excluded from the new matrix

double \*\*B = CreateZeroMatrix(N);

for (int ki = 0, di = 0; ki < N - 1; ki++) {

if (ki == i)di = 1; // if the number is equal to the number of the row to be excluded, we shift it by 1 to ignore it

for (int kj = 0, dj = 0; kj < N - 1; kj++) {

if (kj == j) dj = 1; // similar to the lines

B[ki][kj] = A[ki + di][kj + dj];

}

}

return B;

}

double Determinant(double \*\*arr, int N) {

int det = 0, k = 1; //det - determinant, k - (-1)^i

if (N < 1) cout << "\nThe determinant cannot be calculated!\n";

else if (N == 1) det = arr[0][0]; // matrix 1x1

else if (N == 2) det = arr[0][0] \* arr[1][1] - (arr[1][0] \* arr[0][1]); // matrix 2x2

else { // matrix nxn, where n>=3

for (int i = 0; i < N; i++) { // calculation of the determinant by the method of line decomposition

/\* det B = a31 \* (-1)^(3+1) \* det arr(without 3 row and 1 columns) + ... \*/

det += k \* arr[i][0] \* Determinant(FragmentMatrix(arr, N, i, 0), N - 1);

k = -k;

}

}

return det;

}

double \*\*Inverse(double \*\*A, int N) {

double temp;

double \*\*E = CreateIdentityMatrix(N); // E - identity matrix

double \*\*B = CloneMatrix(A, N); // B - copy of matrix A

for (int i = 0; i < N - 1; ++i) { // swapping the lines

if (B[i][0] < B[i + 1][0]) {

swap(B[i], B[i + 1]); // in the matrix A

swap(E[i], E[i + 1]); // in the unit matrix E

}

}

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

for (int j = 0; j < N; j++) {

if (j != i) {

temp = B[j][i] / B[i][i];

for (int k = 0; k < N; ++k) {

B[j][k] -= B[i][k] \* temp;

E[j][k] -= E[i][k] \* temp;

}

}

}

}

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

temp = B[i][i];

for (int j = 0; j < N; j++) {

B[i][j] /= temp;

E[i][j] /= temp;

}

}

return E;

}

double \*\*Multiple(double \*\*A, double \*\*B, int N) {

double \*\*C = CreateZeroMatrix(N);

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

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

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

C[i][j] += A[i][k] \* B[k][j];

return C;

}

int main() {

srand((unsigned int) time(NULL));

int N = 3;

// double \*\*A = GenerateRandomMatrix(N);

double \*\*A = ReadMatrix("../matrix.txt", N);

cout << "\nMatrix A\n";

PrintMatrix(A, N);

double det = Determinant(A, N);

cout << "\nDeterminant: " << det << "\n\n\n";

if (det == 0) cout << "The matrix does not have an inverse matrix\n";

else {

double \*\*A\_inverse = Inverse(A, N);

cout << "Inverse matrix A^(-1)\n";

PrintMatrix(A\_inverse, N);

cout << "\n\nCheck: A \* A^(-1) = E" << "\n";

PrintMatrix(Multiple(A, A\_inverse, N), N);

Clear(A\_inverse, N);

}

Clear(A, N);

return 0;

}

**Other\_functions.cpp**

/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>

#include <iostream>

#include <fstream>

#include <cmath>

#include "Header.h"

using namespace std;

void Clear(double \*\*A, int N) {

for (int i = 1; i < N; i++)

delete[] A[i];

delete[] A;

}

double \*\*CreateIdentityMatrix(int N) {

double \*\*E = new double \*[N + 1];

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

E[i] = new double[N + 1];

for (int j = 0; j < N; ++j) {

E[i][j] = 0;

}

E[i][i] = 1;

}

return E;

}

double \*\*CreateZeroMatrix(int N) {

double \*\*E = new double \*[N + 1];

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

E[i] = new double[N + 1];

for (int j = 0; j < N; ++j) {

E[i][j] = 0;

}

}

return E;

}

double \*\*CloneMatrix(double \*\*A, int N) {

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

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

B[i] = new double[N];

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

B[i][j] = A[i][j];

}

return B;

}

void PrintMatrix(double \*\*arr, int N) {

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

for (int j = 0; j < N; j++) {

printf("%.3f\t", arr[i][j]);

}

cout << endl;

}

return;

}

double \*\*GenerateRandomMatrix(int N) {

double \*\*A = new double \*[N + 1];

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

A[i] = new double[N + 1];

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

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

}

return A;

}

double \*\*ReadMatrix(const char \*filename, int N) {

double \*\*A = CreateZeroMatrix(N);

ifstream file(filename, ios\_base::in);

if (!file) cout << "\nError of file1 opening.\n";

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

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

file >> A[i][j];

file.close();

return A;

}

**Header.h**

/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>

#ifndef INF\_LAB\_10\_HEADER\_H

#define INF\_LAB\_10\_HEADER\_H

void Clear(double \*\*A, int N);

double \*\*CreateIdentityMatrix(int N);

double \*\*CreateZeroMatrix(int N);

double \*\*CloneMatrix(double \*\*A, int N);

void PrintMatrix(double \*\*arr, int N);

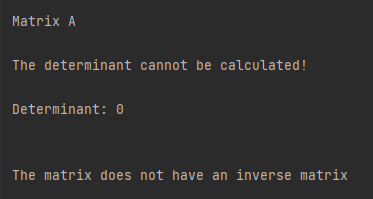
double \*\*GenerateRandomMatrix(int N);

double \*\*ReadMatrix(const char \*filename, int N);

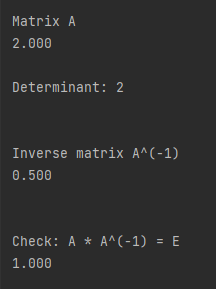
#endif //INF\_LAB\_10\_HEADER\_H

**Анализ результатов**

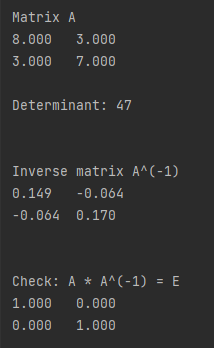
**Создание случайной матрицы размером 0\*0:**

****

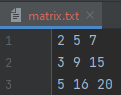
**Создание случайной матрицы размером 1\*1:**

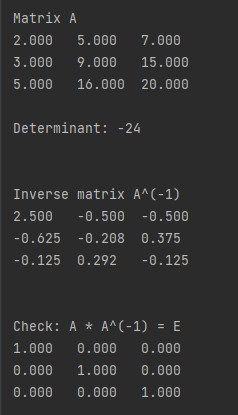
****

**Создание случайной матрицы размером 2\*2:**

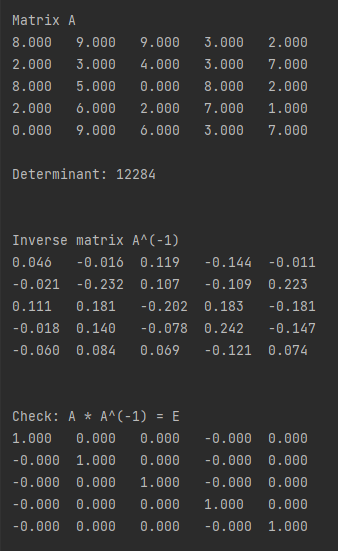
****

**Чтение матрицы из файла:**

****

****

**Создание случайной матрицы размером 5\*5:**

****

**Вывод**

**Я научился:**

* Работать с двумерными динамическими массивами
* Обрабатывать матрицы
* Находить определитель матрицы способом разложения по строке (или столбцу)
* Перемножать матрицы
* Находить обратную матрицу методом Гаусса-Жордана