Московский государственный технический

университет им. Н.Э. Баумана

Курс «Основы информатики»

Отчёт по лабораторной работе №10

Вычислении обратной матрицы методом Гаусса-Жордана

Проверил:

Преподаватель каф. ИУ5

Аксёнова М.В.

Подпись и дата:

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Выполнил:

Студент группы ИУ5-13Б

Пермяков Дмитрий Кириллович

Подпись и дата:

**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 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.

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 должны 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 8.
  2. Доработать алгоритм для случая, когда на главной диагонали появляется ноль. Включить в алгоритм проверку на существование обратной матрицы.

Переменные:

size\_t m – длина массива, n- ширина массива, precision-точность при выводе;

bool typeMatrix-тип матрицы при выводе(scientific or fixed);

long double\*\* arr = new long double\* [m]-начальная матрица

long double\*\* arr1 = new long double\* [m]-присоединённая матрица(единичная)

long double\*\* arr\_old = new long double\* [m]-копия начальной матрицы

bool ok- переменная определяет, существует матрица или нет

bool putting- определяет, ручной ввод матрицы или заполнение rand()%100

long double\*\* c = new long double\* [m]- новая матрица при умножении матрица arr\_old на arr1(преобразованную)

Функции:

void Fill(long double\*\* arr, size\_t m, size\_t n) – заполнение начальной матрицы

void FillUnit(long double\*\* arr, size\_t m, size\_t n) – заполнение присоединённой(единичной) матрицы

void PrintArr(long double\*\* arr, size\_t rows, size\_t cols, bool typeMatrix, size\_t precision)- печать матрицы из лаб.работы 8

void Multiplication(long double\*\* arr\_old, long double\*\* arr, size\_t m, size\_t n, bool typeMatrix, size\_t precision)- умножение матрицы на матрицу

bool StraightCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision)- прямой ход матрицы

void BackCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision)-обратный ход матрицы

size\_t GetDataInt()-ввод целых чисел с проверкой, что вводим именно цифры

**Код программы**

**Заголовки:**

#include <iostream>

#include <iomanip>

#include <cmath>

#include <ctime>

using namespace std;

void Fill(long double\*\* arr, size\_t m, size\_t n);

void FillUnit(long double\*\* arr, size\_t m, size\_t n);

void PrintArr(long double\*\* arr, size\_t rows, size\_t cols, bool typeMatrix, size\_t precision);

void Multiplication(long double\*\* arr\_old, long double\*\* arr, size\_t m, size\_t n, bool typeMatrix, size\_t precision);

bool StraightCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision);

void BackCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision);

size\_t GetDataInt();

**Функции:**

#include "Header.h"

void Fill(long double\*\* arr, size\_t m, size\_t n)

{

//int a[] = {2, 5, 7, 3, 9, 15, 5, 16, 20};

bool putting;

cout << "Ручной ввод или рандом (1|0): ";

cout << "\x1b[35m";

cin >> putting;

cout << "\x1b[0m";

size\_t t = 0;

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

{

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

{

if (!putting)

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

else

{

cout << " array [" << i << "][" << j << "] = ";

cout << "\x1b[35m";

cin >> arr[i][j];

cout << "\x1b[0m";

}

// задача из ворда для проверки

//arr[i][j] = a[t++];

}

}

}

bool StraightCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision)

{

cout << "\x1b[34m============== Прямой ход ==============\x1b[0m\n\n";

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

{

long double k = arr[i][i];

if (k == 0)

{

// если k=0 на диагонале, идём по столбцу в поисках k!=0

size\_t help = 0;

size\_t cheating = 0;

while (arr[help][i] == 0 && help < m)

{

if (help == m - 1)

{

cout << "\x1b[31mОбратная матрица не существует\x1b[0m\n";

return 0;

}

++help;

++cheating;

}

if (cheating == 0)

{

cout << "\x1b[31mОбратная матрица не существует\x1b[0m\n";

return 0;

}

// если находим k != 0, меняем эту строчку со строкой где k=0 на диагонале

for (size\_t i2 = 0; i2 < m; i2++)

{

swap(arr[i][i2], arr[help][i2]);

swap(arr1[i][i2], arr1[help][i2]);

}

PrintArr(arr, m, n, typeMatrix, precision);

k = arr[i][i];

}

// делим строку на arr[i][i]

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

{

arr[i][j] /= k;

arr1[i][j] /= k;

}

// складываем строки

for (size\_t z = i + 1; z < m; z++)

{

long double k = arr[z][i];

for (size\_t ind = 0; ind < n; ind++)

{

arr[z][ind] += k \* (-1) \* arr[i][ind];

arr1[z][ind] += k \* (-1) \* arr1[i][ind];

}

}

PrintArr(arr, m, n, typeMatrix, precision);

}

cout << endl;

return 1;

}

void BackCourse(long double\*\* arr, long double\*\* arr1, size\_t m, size\_t n, bool typeMatrix, size\_t precision)

{

cout << "\x1b[34m============== Обратный ход ==============\x1b[0m\n\n";

for (size\_t i = m - 1; i > 0; i--)

{

long double k = arr[i][i];

for (size\_t j = n - 1; j > 0; j--)

{

arr[i][j] /= k;

arr1[i][j] /= k;

}

for (int z = i - 1; z > -1; z--)

{

long double k = arr[z][i];

for (int ind = n - 1; ind > -1; ind--)

{

arr[z][ind] += k \* (-1) \* arr[i][ind];

arr1[z][ind] += k \* (-1) \* arr1[i][ind];

}

}

PrintArr(arr, m, n, typeMatrix, precision);

}

cout << endl;

}

void FillUnit(long double\*\* arr, size\_t m, size\_t n)

{

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

{

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

{

if (i == j)

arr[i][j] = 1;

else

arr[i][j] = 0;

}

}

}

void Multiplication(long double\*\* arr\_old, long double\*\* arr, size\_t m, size\_t n, bool typeMatrix, size\_t precision)

{

cout << "\n\n\x1b[34m\*\*\*\*\*\*\*\*\*\*\*\*\* Умножение матриц \*\*\*\*\*\*\*\*\*\*\*\*\*\x1b[0m\n\n";

long double\*\* c = new long double\* [m];

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

{

c[i] = new long double[n];

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

{

c[i][j] = 0;

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

c[i][j] += arr\_old[i][k] \* arr[k][j];

}

}

PrintArr(c, m, n, typeMatrix, precision);

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

delete[] c[i];

delete[] c;

}

void PrintArr(long double\*\* arr, size\_t rows, size\_t cols, bool typeMatrix, size\_t precision)

{

size\_t length\_row;

if (typeMatrix)

length\_row = precision + 8;

else if (!typeMatrix)

length\_row = precision + 3;

size\_t cols\_new = 81 / length\_row;

size\_t printed\_cols = 0;

while (printed\_cols < cols)

{

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

{

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

{

if (typeMatrix)

{

if (arr[i][j + printed\_cols] < 0)

cout << scientific << setprecision(precision) << arr[i][j + printed\_cols] << " ";

else

cout << scientific << setprecision(precision) << ' ' << arr[i][j + printed\_cols] << " ";

if (printed\_cols + j == cols - 1)

break;

}

else if (!typeMatrix)

{

if (arr[i][j + printed\_cols] < 0)

cout << fixed << setprecision(precision) << arr[i][j + printed\_cols] << " ";

else

cout << fixed << setprecision(precision) << ' ' << arr[i][j + printed\_cols] << " ";

if (printed\_cols + j == cols - 1)

break;

}

}

cout << endl;

}

printed\_cols += cols\_new;

for (size\_t i = 0; i < 83; i++)

cout << '\_';

cout << endl << endl;

}

}

size\_t GetDataInt()

{

// провенрка введённых данных

while (true)

{

size\_t a;

cout << "\x1b[35m";

cin >> a;

cout << "\x1b[0m";

if (cin.fail())

{

cin.clear();

cin.ignore(32767, '\n');

cout << "\x1b[31mПроверьте введённые данные: \x1b[0m";

}

else

{

cin.ignore(32767, '\n');

return a;

}

}

}

**MAIN:**

#include "Header.h"

using namespace std;

int main()

{

srand(time(NULL));

setlocale(0, "rus");

size\_t m, n, precision;

cout << "Введите m = ";

m = GetDataInt();

cout << "Введите n = ";

n = GetDataInt();

bool typeMatrix;

cout << "Тип матрицы: scientific or fixed (1/0): \x1b[35m";

cin >> typeMatrix;

cout << "\x1b[0mВведите точность. Precision = ";

precision = GetDataInt();

if (m != n)

{

cout << "\x1b[31mm != n, матрица не существует! \x1b[0m\n";

return 0;

}

// создаем двумерные массивы

long double\*\* arr = new long double\* [m];

long double\*\* arr1 = new long double\* [m];

long double\*\* arr\_old = new long double\* [m];

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

{

arr[i] = new long double[n];

arr1[i] = new long double[n];

arr\_old[i] = new long double[n];

}

Fill(arr, m, n);

FillUnit(arr1, m, n);

cout << "\x1b[31m\n\n\*-\*-\*-\*-\*-\*-\*- Начальная матрица -\*-\*-\*-\*-\*-\*-\*\x1b[0m\n\n";

PrintArr(arr, m, n, typeMatrix, precision);

// копируем начальную матрицу в новый массив

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

{

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

arr\_old[i][j] = arr[i][j];

}

cout << endl;

bool ok = StraightCourse(arr, arr1, m, n, typeMatrix, precision);

// если матрица не существует, прекращаем работу

if (!ok)

return 0;

BackCourse(arr, arr1, m, n, typeMatrix, precision);

cout << "\x1b[31m\n\n\*-\*-\*-\*-\*-\*-\*- Обратная матрица -\*-\*-\*-\*-\*-\*-\*\x1b[0m\n\n";

PrintArr(arr1, m, n, typeMatrix, precision);

// проверка

Multiplication(arr\_old, arr1, m, n, typeMatrix, precision);

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

{

delete[] arr[i];

delete[] arr1[i];

delete[] arr\_old[i];

}

delete[] arr;

delete[] arr1;

delete[] arr\_old;

return 0;

}

**Тестирование:**

  

 