**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 ИУ5-15Б | | преподаватель каф. ИУ5 |
| Расулов А. Н. | | Аксенова М. В. |
| Подпись и дата: | | 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 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».

# Разработка алгоритма

1. `double** createMatrix(int n)` -- функция для создания квадратной матрицы размером n. Внутри функции определена переменная `double** matrix` – указатель на созданную матрицу.
2. `double** multiplyMatrixes(double** A, double** B, int n)` – функция для умножения квадратных матриц A и B с размерами n. Возвращает `double** C` – указатель на квадратную матрицу, которая = A * B и имеет такой же размер n.
3. `void findNewLineAndSwap(double** matrix,int n, int c, bool& reverseMatrixExists)` – функция для нахождения новой строки, если на главной диагонали нашелся элемент = 0 и последующей замены местами новой строки и старой, где располагался элемент, который был на главной диагонали = 0.
   a) matrix – матрица, в которой осуществляется поиск
   b) n – ее размерам
   c) c – индекс строки с нулевым элементом на главной диагонали
   d) reverseMatrixExists – возвращаемое функцией значение. Хранит в себе значение того, существует ли обратная матрица для матрицы matrix.
   e) Внутри функции: `int ind` – индекс найденной строки с ненулевым элементом на главной диагонали.
4. `void printMatrix(double** matrix, int n)` – функция для печати квадратной матрицы matrix с размером n в консоль.
5. `void straightWay(double** matrix, int n, bool& reverseMatrixExists)` – функция для выполнения прямого хода над квадратной матрицей matrix с размером n. reverseMatrixExists – возвращаемое функцией значение. Хранит в себе значение того, существует ли обратная матрица для матрицы matrix.
   Внутри функции определены:
   a) `int c` – индекс элемента главной диагонали
   b) `double k` – элемент главной диагонали, коэффициент, чтобы получить нули ниже главной диагонали
6. `void reverseWay(double** matrix, int n)` – функция для выполнения обратного хода над квадратной матрицой matrix с размером n.
   Внутри функции определены:
   1. `int c` – индекс элемента главной диагонали
   2. `double k` – элемент главной диагонали, коэффициент, чтобы получить нули выше главной диагонали
7. `void printExtendedMatrix(double** matrix, int n)` -- функция для вывода расширенной матрицы matrix с размером n строк и 2*n столбцов
8. `void deleteMatrix(double** matrix, int n)` – функция для освобождения памяти, выделенной под матрицу matrix с размером n.
9. В функции main:
   a) `int n` – порядок матрицы
   b) `double** originalMatrix` – исходная матрица, вводимая пользователем
   c) `double** extendedMatrix` – объединенная с единичной исходная матрица
   d) `bool reverseMatrixExists` – отвечает за то, есть ли у матрицы originalMatrix обратная к ней
   e) `double** inverseMatrix` – обратная матрица к матрице originalMatrix
   f) `double** mul` – произведение матриц inverseMatrix и originalMatrix

# Текст программы

header.h

```
#include <iostream>
#include <iomanip>
using namespace std;
void printMatrix(double** matrix, int n);
double** createMatrix(int n);
double** multiplyMatrixes(double** A, double** B, int n);
void findNewLineAndSwap(double** matrix, int n, int c, bool& reverseMatrixExists);
void straightWay(double** matrix, int n, bool& reverseMatrixExists);
void reverseWay(double** matrix, int n);
void printExtendedMatrix(double** matrix, int n);
void deleteMatrix(double** matrix, int n);
```

source.cpp

```
#include "header.h"
double** createMatrix(int n) {
    double** matrix = new double* [n];
    for (int i = 0; i < n; ++i)
        matrix[i] = new double[n];
    return matrix;
}
void findNewLineAndSwap(double** matrix, int n, int c, bool& reverseMatrixExists) {
    int ind = 0;
    reverseMatrixExists = false;
    for (int i = c + 1; i < n; ++i) {
        if (matrix[i][c] != 0) {
            ind = i;
            reverseMatrixExists = true;
        }
    }
    if (reverseMatrixExists) {
        for (int i = 0; i < 2 * n; ++i)
            swap(matrix[c][i], matrix[ind][i]);
    }
}
void straightWay(double** matrix, int n, bool& reverseMatrixExists) {
    int c = 0; double k;
    while (c < n) {
        k = matrix[c][c];
        if (k == 0) {
            findNewLineAndSwap(matrix, n, c, reverseMatrixExists);
            if (!reverseMatrixExists)
                return;
        }
        k = matrix[c][c];

        for (int i = 0; i < 2 * n; ++i)
            matrix[c][i] /= k;

        for (int i = c + 1; i < n; ++i) {
            k = matrix[i][c];
            for (int j = 0; j < 2 * n; ++j)
                matrix[i][j] -= k * matrix[c][j];
        }
        ++c;
    }
}
void reverseWay(double** matrix, int n) {
    double k; int c = 1;
    while (c < n) {
        for (int i = 0; i < n - c; ++i) {
            k = matrix[i][n - c];
            for (int j = 0; j < 2 * n; ++j)
                matrix[i][j] -= k * matrix[n - c][j];
        }
        ++c;
    }
}
void printMatrix(double** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            cout << setw(8) << setprecision(2) << matrix[i][j];
        cout << endl;
    }
}
void printExtendedMatrix(double** matrix, int n) {
    for (int i = 0; i < n; ++i) {
```

```
        for (int j = 0; j < 2 * n; ++j)
            cout << setw(8) << setprecision(2) << matrix[i][j];
        cout << endl;
    }
}
double** multiplyMatrixes(double** A, double** B, int n) {
    double** C = createMatrix(n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] = 0;
    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;
}
void deleteMatrix(double** matrix, int n) {
    for (int i = 0; i < n; ++i)
        delete[] matrix[i];
    delete[] matrix;
}
```

main.cpp

```
#include "header.h"
int main() {
    cout << "Введите порядок матрицы: ";
    int n;
    cin >> n;
    double** originalMatrix = createMatrix(n);
    // Заполнение входной матрицы
    for (int i = 0; i < n; ++i) {
        cout << "Введите значения элементов " << i + 1 << " строки: ";
        for (int j = 0; j < n; ++j)
            cin >> originalMatrix[i][j];
    }
    cout << "Исходная матрица:\n";
    printMatrix(originalMatrix, n);
    //  Заполнение расширенной матрицы
    double** extendedMatrix = createMatrix(2 * n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            extendedMatrix[i][j] = originalMatrix[i][j];
        for (int j = n; j < 2 * n; ++j) {
            if (i + n == j)
                extendedMatrix[i][j] = 1;
            else
                extendedMatrix[i][j] = 0;
        }
    }
    bool reverseMatrixExists;
    straightWay(extendedMatrix, n, reverseMatrixExists);
    if (!reverseMatrixExists) {
        cout << "Обратная матрица не существует!\n";

        deleteMatrix(originalMatrix, n);
        deleteMatrix(extendedMatrix, 2 * n);
        return 0;
    }
    cout << "После прямого хода:\n\n";
    printExtendedMatrix(extendedMatrix, n);

    reverseWay(extendedMatrix, n);
    cout << "После обратного хода:\n\n";
    printExtendedMatrix(extendedMatrix, n);
    double** inverseMatrix = createMatrix(n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            inverseMatrix[i][j] = extendedMatrix[i][n + j];
    cout << "Обратная матрица:\n\n";
    printMatrix(inverseMatrix, n);
    // произведение начальной и обратной к ней матриц
    cout << "Произведение начальной матрицы на ей обратную: " << endl;
    double** mul = multiplyMatrixes(originalMatrix, inverseMatrix, n);
    printMatrix(mul, n);
    // Free allocated memory
    deleteMatrix(originalMatrix, n); deleteMatrix(extendedMatrix, 2 * n);
    deleteMatrix(mul, n); deleteMatrix(inverseMatrix, n);
}
```

# Анализ результатов

```
Введите порядок матрицы: 3
Введите значения элементов 1 строки: 1 2 3
Введите значения элементов 2 строки: 4 5 6
Введите значения элементов 3 строки: 7 8 9
Исходная матрица:
       1       2       3
       4       5       6
       7       8       9
Обратная матрица не существует!
```

```
Введите порядок матрицы: 3
Введите значения элементов 1 строки: 2 5 7
Введите значения элементов 2 строки: 3 9 15
Введите значения элементов 3 строки: 5 16 20
Исходная матрица:
       2       5       7
       3       9      15
       5      16      20
После прямого хода:

       1     2.5     3.5     0.5       0       0
       0       1       3      -1    0.67       0
      -0      -0       1   -0.12    0.29   -0.12
После обратного хода:

       1       0       0     2.5    -0.5    -0.5
       0       1       0   -0.62   -0.21    0.38
      -0      -0       1   -0.12    0.29   -0.12
Обратная матрица:

     2.5    -0.5    -0.5
   -0.62   -0.21    0.38
   -0.12    0.29   -0.12
Произведение начальной матрицы на ей обратную:
       1       0       0
       0       1       0
       0       0       1
```

```
Введите порядок матрицы: 4
Введите значения элементов 1 строки: 0 0 2 0
Введите значения элементов 2 строки: 5 0 2 1
Введите значения элементов 3 строки: 1 5 0 0
Введите значения элементов 4 строки: 1 0 0 0
Исходная матрица:
       0       0       2       0
       5       0       2       1
       1       5       0       0
       1       0       0       0
После прямого хода:

       1       0       0       0       0       0       0       1
       0       1       0       0       0       0     0.2    -0.2
       0       0       1     0.5       0     0.5       0    -2.5
      -0      -0      -0       1      -1       1      -0      -5
После обратного хода:

       1       0       0       0       0       0       0       1
       0       1       0       0       0       0     0.2    -0.2
       0       0       1       0     0.5       0       0       0
      -0      -0      -0       1      -1       1      -0      -5
Обратная матрица:

       0       0       0       1
       0       0     0.2    -0.2
     0.5       0       0       0
      -1       1      -0      -5
Произведение начальной матрицы на ей обратную:
       1       0       0       0
       0       1       0       0
       0       0       1       0
       0       0       0       1
```