

ВВЕДЕНИЕ

Системы линейных алгебраических уравнений широко применяются во многих областях, включая математику, физику, инженерию, экономику и компьютерные науки.

Решение систем линейных уравнений используется в электротехнике, механике, строительстве и других инженерных дисциплинах для моделирования и анализа различных систем, таких как электрические цепи, механические конструкции и тепловые процессы.

В физике СЛАУ используются для моделирования физических систем, например, при решении задач механики, электродинамики, квантовой физики и др.

В экономике СЛАУ применяются для анализа экономических систем, оптимизации распределения ресурсов, моделирования финансовых потоков и других задач, связанных с экономическими и финансовыми моделями.

В компьютерных науках СЛАУ используются в различных областях, включая компьютерную графику, машинное обучение, численные методы, анализ данных и другие области, где требуется решение линейных систем или аппроксимация данных.

Решение СЛАУ имеет широкий спектр применений и является важным инструментом для моделирования, анализа и оптимизации различных систем и процессов во многих областях науки и техники.

В данной лабораторной работе необходимо реализовать метод Гаусса для решения систем линейных алгебраических уравнений.

Для достижения поставленных целей лабораторной работы необходимо выполнить следующие задачи:

- рассмотреть понятие СЛАУ и изучить метод Гаусса для решения СЛАУ;
- реализовать решение СЛАУ методом Гаусса через массив массивов и одномерный массив с разным выделением памяти;
- составить блок-схему метода Гаусса;
- рассчитать трудоемкость метода Гаусса;
- экспериментально проверить работу метода Гаусса;

- сравнить разные варианты адресаций в матрицах;
- сравнить разные варианты выделения памяти под матрицу;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Система линейных алгебраических уравнений

Система линейных алгебраических уравнений (СЛАУ) представляет собой совокупность нескольких линейных уравнений, в которых неизвестными являются переменные. Каждое уравнение в системе имеет вид, представленный следующим образом:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2, \\ &\dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m, \end{aligned} \tag{1}$$

где x_1, x_2, \dots, x_n - неизвестные переменные, $a_{11}, a_{12}, \dots, a_{mn}$ - коэффициенты, b_1, b_2, \dots, b_m - известные константы, а m и n - количество уравнений и переменных соответственно [1, с.33].

СЛАУ имеет широкое применение в различных областях, включая математику, физику, инженерию, экономику и компьютерные науки. Решение систем линейных уравнений позволяет найти значения переменных, которые удовлетворяют заданным условиям, и может быть полезным инструментом для моделирования и анализа различных явлений и процессов.

Решение СЛАУ представляет собой набор значений переменных x_1, x_2, \dots, x_n , которые удовлетворяют всем уравнениям системы. Решение может быть единственным, когда существует одно конкретное значение для каждой переменной, либо может иметь множество решений, когда существует бесконечное количество комбинаций переменных, удовлетворяющих системе.

Существуют различные методы решения, такие как метод Гаусса-Жордана, метод Гаусса или метод матричных операций. В данной работе рассматривается решение методом Гаусса, который позволяет эффективно находить решения СЛАУ и использовать их для решения конкретных задач.

1.2 Метод Гаусса для решения СЛАУ

Метод Гаусса является одним из наиболее распространенных методов решения систем линейных алгебраических уравнений. Он основан на последовательном применении элементарных преобразований над уравнениями системы с целью привести ее к эквивалентной системе, в которой решение становится очевидным [1, с.39].

Для решения СЛАУ методом Гаусса можно выделить следующие шаги:

1. Записать систему уравнений в расширенной форме, включающей коэффициенты и правые части уравнений в матрицу;
2. Применить элементарные преобразования над уравнениями системы с целью привести матрицу к ступенчатому виду или диагональному виду, где каждая строка матрицы содержит все нули слева от главной диагонали. Элементарные преобразования включают перестановку уравнений, умножение уравнения на ненулевое число и сложение уравнений. Данный этап часто называют прямым ходом Гаусса;
3. Обратный ход: обратиться к последнему уравнению и найти значение переменной x_n , где n - количество переменных в СЛАУ. Используя найденное значение x_n , выразить x_{n-1} из предпоследнего уравнения. Продолжать этот процесс до тех пор, пока не найдутся все значения переменных x_1, x_2, \dots, x_n . Эти значения будут составлять решение СЛАУ.

Метод Гаусса обеспечивает эффективный способ нахождения решений СЛАУ, особенно когда количество уравнений и переменных велико.

1.2 Представление СЛАУ

Для достижения поставленных целей необходимо определенным образом представить СЛАУ. Для этого в данной работе будет использоваться матрица, которую можно по-разному представить.

Представить матрицу можно в виде одномерного массива. В одномерном массиве элементы матрицы располагаются последовательно, без учета строк и столбцов. Для доступа к элементу матрицы по координатам (i, j) в одномерном массиве используется формула:

$$\text{Element} = \text{Matrix} [i * n + j], \quad (2)$$

где i – номер строки, j – номер столбца, а n – количество столбцов в матрице [4, с.100].

Представление матрицы в виде одномерного массива обычно используется для экономии памяти и упрощения операций над матрицами.

Существует также представление матрицы в виде двумерного массива – в нем элементы матрицы организуются в виде строк и столбцов. Каждая строка матрицы является внутренним массивом, а сама матрица представляет собой массив массивов. В этом способе каждый элемент имеет два индекса: индекс строки и индекс столбца. Матрица размером $m \times n$ хранится в двумерном массиве размером $m \times n$ [3, с.343].

Для доступа к элементу матрицы по координатам (i, j) в двумерном массиве используется следующая формула:

$$\text{Element} = \text{Matrix} [i][j], \quad (3)$$

где i – номер строки от 0 до $m - 1$, j – индекс столбца от 0 до $n - 1$ [3, с.345].

Представление в виде двумерного массива более естественно для человеческого восприятия, особенно при выполнении операций над

матрицами. Однако, данное представление требует большего объема памяти для хранения данных, особенно при работе с большими матрицами.

Вывод

На данном этапе была изучена и разобрана необходимая теория по СЛАУ и методу Гаусса: объяснены базовые понятия, описаны идея и шаги решения СЛАУ методом Гаусса.

Показано то, как можно представить матрицу с помощью двух вариантов адресаций, таких как массив массивов и одномерный массив.

2 Конструкторская часть

2.1 Разработка алгоритма

На рисунке 1 приведена блок-схема алгоритма решения СЛАУ методом Гаусса.

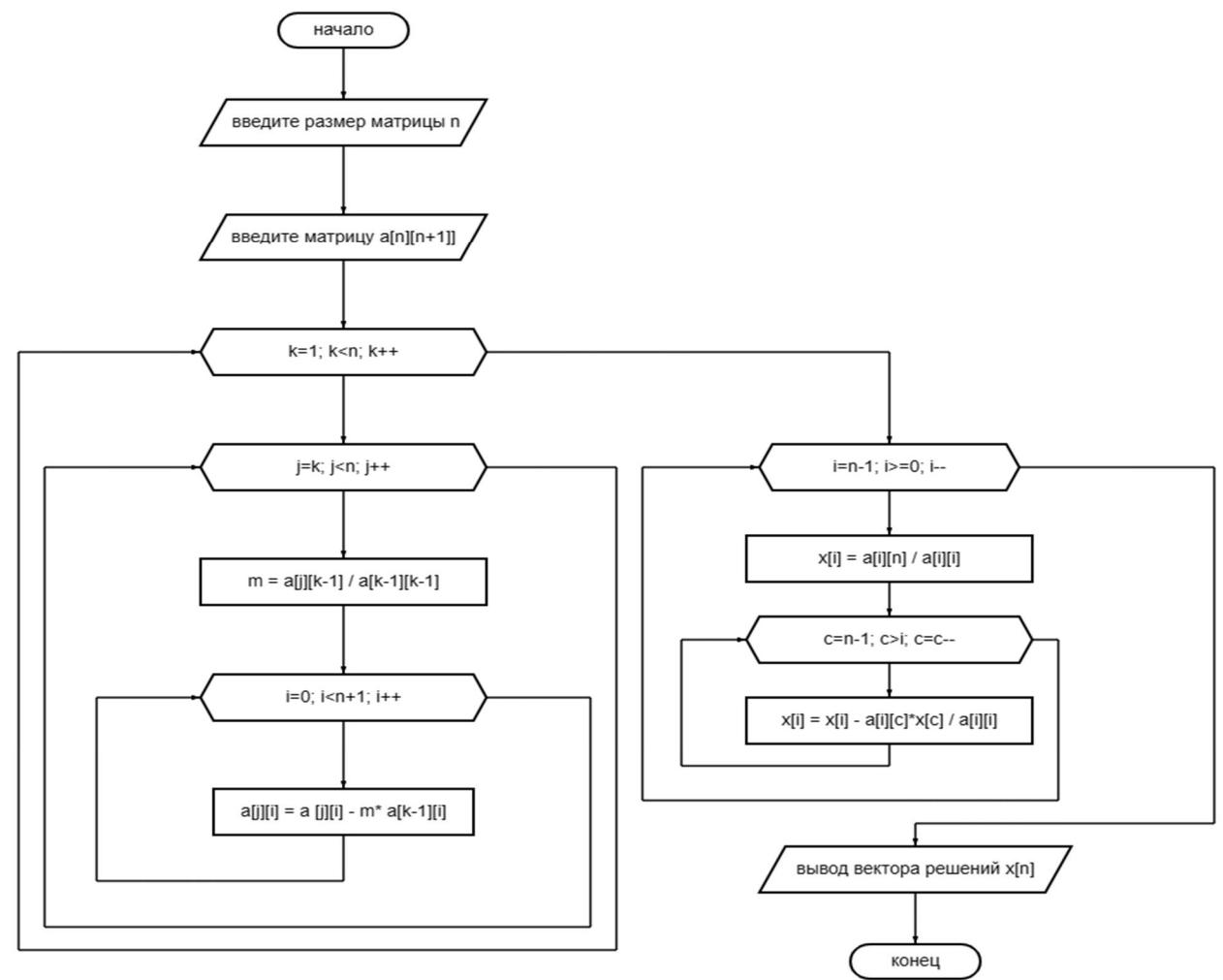


Рисунок 1 – Блок-схема метода Гаусса для решения СЛАУ

2.2 Трудоемкость алгоритма

Для последующего вычисления трудоемкости алгоритмов необходимо ввести модель вычислений:

1) Пусть: +, -, /, %, ==, !=, <, >, <=, >=, [], *, ++, -- – трудоемкость 1.

2) Трудоемкость оператора выбора if условие then A else B рассчитывается как на рисунке 2:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$

Рисунок 2 - Трудоемкость оператора выбора if условие then A else B

3) Трудоемкость цикла for рассчитывается как на рисунке 3:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$$

Рисунок 3 - Трудоемкость оператора for

4) Трудоемкость вызова функции равна 0.

Произведём вычисление трудоёмкости для алгоритма. В таблице 1 записана трудоемкость для метода Гаусса:

Таблица 1 – Расчёт трудоемкости метода Гаусса

Временная сложность Метода Гаусса	
Худший случай	Лучший случай
$O(n^3)$	$O(n^k)$, где $k < 3$

Стоит отметить, что $O(n^3)$ – это оценка трудоемкости в худшем случае. Чаще всего фактическая трудоемкость может быть ниже, особенно если матрица имеет определенные свойства, такие как разреженность или структурная особенность.

Вывод

На основе теоретических данных, полученных из аналитического раздела, была построена блок-схема алгоритма метода Гаусса для решения СЛАУ. Проведена теоретическая оценка трудоемкости метода Гаусса.

3 Экспериментальная часть

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив расширенной матрицы СЛАУ из исключительно целых чисел;
- на выходе — найденный с помощью метода Гаусса массив решения СЛАУ, состоящий из переменных типа double.

3.2 Технические характеристики

Технические характеристики устройства, на котором происходило тестирование:

- Операционная система: Windows 10 Pro 64 бит
- Память: 16 ГБ
- Процессор: Intel Core Pentium G4560 3.5 ГГц

Тестирование проводилось на ПК, включенном в сеть электропитания. Во время тестирования ПК был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

3.3 Тестирование программ

В таблице 2 приведены функциональные тесты для программ с методом Гаусса для одномерного и двумерного массивов – все тесты были пройдены успешно.

Таблица 2 – Тестирование метода Гаусса

Входной массив	Результат	Ожидаемый результат
1 1 1 1	1	1
2 1 1 2	0	0
3 2 1 3	0	0
{Пустой массив}	ENTERED EMPTY MATRIX	ENTERED EMPTY MATRIX
1 1 2 1	0	0
1 3 7 1 1	0.0136986	0.0136986
13 1 2 1	0.493151	0.493151
1 2	2	2

В таблице 3 приведено среднее время выполнения программ для разных представлений матриц.

Таблица 3 – Время выполнения программ

Представление матрицы	Размер расширенной матрицы $n \times (n+1)$	Среднее время работы, мс
Одномерный	3	1.0
Двумерный	3	1.1

Одномерный	4	3.4
Двумерный	4	3.9
Одномерный	5	4.5
Двумерный	5	6.7
Одномерный	10	7.5
Двумерный	10	11.1

Вывод

По результатам экспериментальной части можно сказать, что оба способа представления матриц имеют свои преимущества, и выбор зависит от конкретной задачи и требований программы. Важно учитывать эффективность использования памяти и удобство доступа к элементам матрицы при выборе соответствующего представления.

Так, двумерный массив обеспечивает легкость в работе с индексами строк и столбцов, а также лучшую читаемость кода и понимание алгоритма.

По результатам сравнения времени выполнения программ видно, что разница между двумерным и одномерным представлением матрицы практически незначительная, поскольку оба представления требуют одинакового количества операций в методе Гаусса и в выводе матрицы решений.

По результатам сравнения памяти можно сделать вывод, что обычно операции с одномерными массивами выполняются быстрее, чем

с массивами массивов. Вероятно, это связано с локальностью данных в одномерном массиве, ибо элементы одномерного массива расположены последовательно в памяти, что обеспечивает лучшую локализацию данных и улучшает производительность кэшей процессора. В массиве массивов же элементы могут быть разбросаны по памяти, что может ухудшить производительность по памяти из-за частых обращений к различным областям памяти. К тому же в одномерном массиве элементы хранятся без дополнительных структур данных. В массиве массивов каждый элемент представляет отдельный массив, и требуется дополнительная память для хранения ссылок на каждый из этих массивов. Таким образом, выходит, что общий объем памяти, занимаемый массивом массивов, будет больше, чем одномерного массива с тем же количеством элементов.

ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы были проведены анализ, подробное изучение, реализация метода Гаусса для решений СЛАУ. Проведено сравнение между статически и динамическим выделением памяти, а также между вариантами адресаций.

Были выполнены следующие задачи лабораторной работы:

- рассмотрено понятие СЛАУ, и изучен метод Гаусса для решения СЛАУ;
- реализовано решение СЛАУ методом Гаусса через массив массивов и одномерный массив с разным выделением памяти;
- составлена блок-схема метода Гаусса;
- рассчитана трудоемкость метода Гаусса;
- экспериментально проверена работа метода Гаусса;
- проведено сравнение разных вариантов адресаций в матрицах;
- проведено сравнение разных вариантов выделения памяти под матрицу;
- на основании проделанной работы сформулирован вывод.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вычислительная линейная алгебра с примерами на MATLAB. В.И. Горбаченко, 2011. – 37 с.
2. Роберт Седжвик. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных. Algorithms in C++. — М.: «Вильямс», 2011. — 87 с.
3. Д. Либерти. Освой самостоятельно C++ за 21 день, 2010. – 211 с.
4. Полный справочник по C++, 4-е издание C++: The Complete Reference, 4th Edition. — М.: «Вильямс», 2011. — 279 с.
5. С.Пратта. Язык программирования C++. Лекции и упражнения, 2012. – 306 с.

ПРИЛОЖЕНИЕ А

Листинг программы

Листинг 1 – Одномерный массив со статическим выделением

```
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

//Одномерный массив + статическое выделение

const int MAX_SIZE = 100; //Максимальный размер матрицы

//Свообразные errno
int errf = 0;
int errm = 0;

//Функция для чтения матрицы из текстового файла
void readMatrixFromFile(const std::string& filename, double matrix[], int& rows, int& cols) {
    std::ifstream file(filename);

    if (file.is_open()) {
        std::string line;
        int row = 0;
        int col = 0;
        while (std::getline(file, line)) {
            std::istringstream iss(line);
            double value;
            while (iss >> value) {
                matrix[row * cols + col] = value;
                col++;
            }
            if (cols == 0) {
                cols = col;
            }
            row++;
            col = 0;
        }
        rows = row;
        file.close();
    } else {
        std::cout << "UNABLE TO OPEN FILE: " << filename << std::endl;
        errf = 1;
    }
}

//Функция для вывода матрицы
void printMatrix(const double matrix[], int rows, int cols) {
    if ((rows == 0) || (cols == 0)) {
        std::cout << "ENTERED EMPTY MATRIX" << std::endl;
    }
```

```

        errm = 1;
    } else{
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                std::cout << matrix[i * cols + j] << " ";
            }
            std::cout << std::endl;
        }
    }
}

// Метод Гаусса
void solveLinearSystem(double matrix[], double solution[], int size) {
    //Прямой ход метода Гаусса
    for (int k = 0; k < size; k++) {
        //Поиск максимального элемента в столбце k
        int maxRow = k;
        for (int i = k + 1; i < size; i++) {
            if (std::abs(matrix[i * (size + 1) + k]) > std::abs(matrix[maxRow *
* (size + 1) + k])) {
                maxRow = i;
            }
        }

        //Перестановка строк
        if (maxRow != k) {
            for (int j = k; j <= size; j++) {
                std::swap(matrix[k * (size + 1) + j], matrix[maxRow * (size +
1) + j]);
            }
        }

        //Приведение матрицы к треугольному виду
        for (int i = k + 1; i < size; i++) {
            double factor = matrix[i * (size + 1) + k] / matrix[k * (size +
1) + k];
            for (int j = k; j <= size; j++) {
                matrix[i * (size + 1) + j] -= factor * matrix[k * (size + 1) +
j];
            }
        }
    }

    //Обратный ход метода Гаусса
    for (int i = size - 1; i >= 0; i--) {
        solution[i] = matrix[i * (size + 1) + size];
        for (int j = i + 1; j < size; j++) {
            solution[i] -= matrix[i * (size + 1) + j] * solution[j];
        }
        solution[i] /= matrix[i * (size + 1) + i];
    }
    printMatrix(solution, size, 1);
}

int main() {
    //Задаем файл
    std::string filename = "matrix.txt";
    double matrix[MAX_SIZE * (MAX_SIZE+1)];
    double solution[MAX_SIZE];
    int rows = 0;
    int cols = 0;
}

```

```

//Чтение матрицы из файла
readMatrixFromFile(filename, matrix, rows, cols);

if (errf == 0) {
    //Вывод исходной матрицы
    std::cout << "A|B original extended matrix:" << std::endl;
    printMatrix(matrix, rows, cols);

    int size = rows;

    if (errm == 0) {
        //Решение системы линейных алгебраических уравнений
        std::cout << "X solution matrix:" << std::endl;
        solveLinearSystem(matrix, solution, size);
    }
}

return 0;
}

```

Листинг 2 – Одномерный массив с динамическим выделением

```

#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

//Одномерный массив + динамическое выделение

const int MAX_SIZE = 100; //Максимальный размер матрицы

//Своеобразные errno
int errf = 0;
int errm = 0;

//Функция для чтения матрицы из текстового файла
void readMatrixFromFile(const std::string& filename, double matrix[], int& rows, int& cols) {
    std::ifstream file(filename);

    if (file.is_open()) {
        std::string line;
        int row = 0;
        int col = 0;
        while (std::getline(file, line)) {
            std::istringstream iss(line);
            double value;
            while (iss >> value) {
                matrix[row * cols + col] = value;
                col++;
            }
            if (cols == 0) {
                cols = col;
            }
            row++;
            col = 0;
        }
    }
}

```

```

        rows = row;
        file.close();
    } else {
        std::cout << "UNABLE TO OPEN FILE: " << filename << std::endl;
        errf = 1;
    }
}

//Функция для вывода матрицы
void printMatrix(const double matrix[], int rows, int cols) {
    if((rows == 0) || (cols == 0)) {
        std::cout << "ENTERED EMPTY MATRIX" << std::endl;
        errm = 1;
    } else{
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                std::cout << matrix[i * cols + j] << " ";
            }
            std::cout << std::endl;
        }
    }
}

void solveLinearSystem(double matrix[], double solution[], int size) {
    //Прямой ход метода Гаусса
    for (int k = 0; k < size; k++) {
        //Поиск максимального элемента в столбце k
        int maxRow = k;
        for (int i = k + 1; i < size; i++) {
            if (std::abs(matrix[i * (size + 1) + k]) > std::abs(matrix[maxRow * (size + 1) + k])) {
                maxRow = i;
            }
        }

        //Перестановка строк
        if (maxRow != k) {
            for (int j = k; j <= size; j++) {
                std::swap(matrix[k * (size + 1) + j], matrix[maxRow * (size + 1) + j]);
            }
        }

        //Приведение матрицы к треугольному виду
        for (int i = k + 1; i < size; i++) {
            double factor = matrix[i * (size + 1) + k] / matrix[k * (size + 1) + k];
            for (int j = k; j <= size; j++) {
                matrix[i * (size + 1) + j] -= factor * matrix[k * (size + 1) + j];
            }
        }
    }

    //Обратный ход метода Гаусса
    for (int i = size - 1; i >= 0; i--) {
        solution[i] = matrix[i * (size + 1) + size];
        for (int j = i + 1; j < size; j++) {
            solution[i] -= matrix[i * (size + 1) + j] * solution[j];
        }
        solution[i] /= matrix[i * (size + 1) + i];
    }
}

```

```

        }
        printMatrix(solution, size, 1);
    }

int main() {
    std::string filename = "matrix.txt";
    double solution[MAX_SIZE];
    int rows = 0;
    int cols = 0;

    std::cout << "Enter the number of rows A|B matrix: ";
    std::cin >> rows;
    std::cout << "Enter the number of columns A|B matrix: ";
    std::cin >> cols;

    //Выделение
    double *matrix = new double [rows*cols];

    //Чтение матрицы из файла
    readMatrixFromFile(filename, matrix, rows, cols);

    if (errf == 0) {
        //Вывод исходной матрицы
        std::cout << "A|B original extended matrix:" << std::endl;
        printMatrix(matrix, rows, cols);

        int size = rows;

        if (errm == 0) {
            //Решение системы линейных алгебраических уравнений
            std::cout << "X solution matrix:" << std::endl;
            solveLinearSystem(matrix, solution, size);
        }
    }

    //Очистка
    delete [] matrix;

    return 0;
}

```

Листинг 3 – Двумерный массив со статическим выделением

```

#include <iostream>
#include <fstream>
#include <sstream>

//Массив массивов + статическое выделение

const int MAX_SIZE = 100; //Максимальный размер матрицы

//Своеборазные errno

```

```

int errf = 0;
int errm = 0;

//Функция для чтения матрицы из текстового файла
void readMatrixFromFile(const std::string& filename, double matrix[MAX_SIZE][MAX_SIZE+1], int& size) {
    std::ifstream file(filename);

    if (file.is_open()) {
        std::string line;
        int row = 0;
        while (std::getline(file, line)) {
            std::istringstream iss(line);
            int col = 0;
            double value;
            while (iss >> value) {
                matrix[row][col] = value;
                col++;
            }
            row++;
        }
        size = row; //Запоминаем размер матрицы
        file.close();
    } else {
        std::cout << "UNABLE TO OPEN FILE: " << filename << std::endl;
        errf = 1;
    }
}

//Функция для вывода матрицы
void printMatrix(double matrix[MAX_SIZE][MAX_SIZE+1], int row, int col) {
    if (row == 0) {
        std::cout << "ENTERED EMPTY MATRIX" << std::endl;
        errm = 1;
    } else {
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                std::cout << matrix[i][j] << " ";
            }
            std::cout << std::endl;
        }
    }
}

void solveLinearSystem(double matrix[][MAX_SIZE + 1], double solution[MAX_SIZE], int size) {
    //Прямой ход метода Гаусса
    for (int k = 0; k < size; k++) {
        //Поиск максимального элемента в столбце k
        int maxRow = k;
        for (int i = k + 1; i < size; i++) {
            if (std::abs(matrix[i][k]) > std::abs(matrix[maxRow][k])) {
                maxRow = i;
            }
        }

        //Перестановка строк
        if (maxRow != k) {
            for (int j = k; j <= size; j++) {
                std::swap(matrix[k][j], matrix[maxRow][j]);
            }
        }
    }
}

```

```

    }

    //Приведение матрицы к треугольному виду
    for (int i = k + 1; i < size; i++) {
        double factor = matrix[i][k] / matrix[k][k];
        for (int j = k; j <= size; j++) {
            matrix[i][j] -= factor * matrix[k][j];
        }
    }

    //Обратный ход метода Гаусса
    for (int i = size - 1; i >= 0; i--) {
        solution[i] = matrix[i][size];
        for (int j = i + 1; j < size; j++) {
            solution[i] -= matrix[i][j] * solution[j];
        }
        solution[i] /= matrix[i][i];
    }
}

int main() {
    std::string filename = "matrix.txt";
    double matrix[MAX_SIZE][MAX_SIZE+1];
    double solution[MAX_SIZE];
    int size = 0;

    //Чтение матрицы из файла
    readMatrixFromFile(filename, matrix, size);

    if (errf == 0) {
        //Вывод исходной матрицы
        std::cout << "A|B original extended matrix:" << std::endl;
        printMatrix(matrix, size, size + 1);

        if (errm == 0) {
            //Решение системы линейных алгебраических уравнений
            solveLinearSystem(matrix, solution, size);

            //Вывод решения
            std::cout << "X solution matrix:" << std::endl;
            for (int i = 0; i < size; i++) {
                std::cout << solution[i] << std::endl;
            }
        }
    }

    return 0;
}

```

Листинг 4 – Двумерный массив с динамическим выделением

```
#include <iostream>
```

```

using namespace std;

//Массив массивов + динамические выделение

//Функция для освобождения памяти от двумерного динамического массива
void deleteMatrix(double** matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;
}

//Функция для решения системы линейных алгебраических уравнений методом
Гаусса
double* solveSystemOfEquations(double** matrix, int size) {
    //Приведение к треугольному виду
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            double ratio = matrix[j][i] / matrix[i][i];
            for (int k = i; k < size + 1; k++) {
                matrix[j][k] -= ratio * matrix[i][k];
            }
        }
    }

    //Обратный ход
    double* solution = new double[size];
    for (int i = size - 1; i >= 0; i--) {
        double sum = 0;
        for (int j = i + 1; j < size; j++) {
            sum += matrix[i][j] * solution[j];
        }
        solution[i] = (matrix[i][size] - sum) / matrix[i][i];
    }

    return solution;
}

int main() {
    int size;

    std::cout << "Enter the number of rows (size) of the system: ";
    std::cin >> size;

    //Выделение памяти под расширенную матрицу
    double** matrix = new double*[size];
    for (int i = 0; i < size; i++) {
        matrix[i] = new double[size + 1];
    }

    //Ввод расширенной матрицы
    std::cout << "A|B original extended matrix:\n";
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size + 1; j++) {
            std::cin >> matrix[i][j];
        }
    }

    //Решение системы уравнений
    double* solution = solveSystemOfEquations(matrix, size);
}

```

```
//Вывод решения
std::cout << "X solution matrix:\n";
for (int i = 0; i < size; i++) {
    std::cout << "x" << i + 1 << " = " << solution[i] << std::endl;
}

//Освобождение памяти
deleteMatrix(matrix, size);
delete[] solution;

return 0;
}
```