**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет программной инженерии и компьютерной техники

Вычислительная математика

**Лабораторная работа №1**

Вариант 6

**Студент**: Закусов К.Я.

**Группа:** P3108

**Преподаватель**: Клименков С.В.

САНКТ-ПЕТЕРБУРГ

2077

Оглавление

[Введение. 3](#_Toc116642580)

[Создать дерево каталогов и файлов с содержимым 3](#_Toc116642581)

[Создание дерева 3](#_Toc116642582)

[Заполнение файлов 3](#_Toc116642583)

[Установить согласно заданию права на файлы и каталоги 4](#_Toc116642584)

[Фрагмент файла lab0\_opd.sh 4](#_Toc116642585)

[Файл lab0\_opd\_permissions.sh 4](#_Toc116642586)

[Скопировать часть дерева и создать ссылки внутри дерева 5](#_Toc116642587)

[Выполнить команду ls -lR 5](#_Toc116642588)

[Фрагмент файла lab0\_opd.sh 5](#_Toc116642589)

[Вывод: 5](#_Toc116642590)

[Выполнить поиск и фильтрацию файлов, каталогов и содержащихся в них данных 7](#_Toc116642591)

[Удаление файлов и директорий 7](#_Toc116642592)

[Вывод 7](#_Toc116642593)

[Используемая литература 7](#_Toc116642594)

# Введение.

Исходный код лабораторной работы и результаты находится здесь:  
<https://github.com/username918r818/comp-math-lab1>

Читабельная версия:

<https://pastebin.com/B9pY374N>

# Цели работы

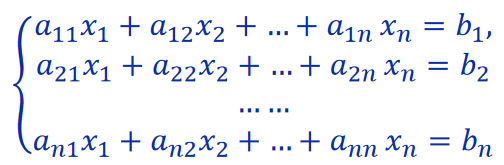
Написать программу, которая методом простых итераций находит решение СЛАУ.

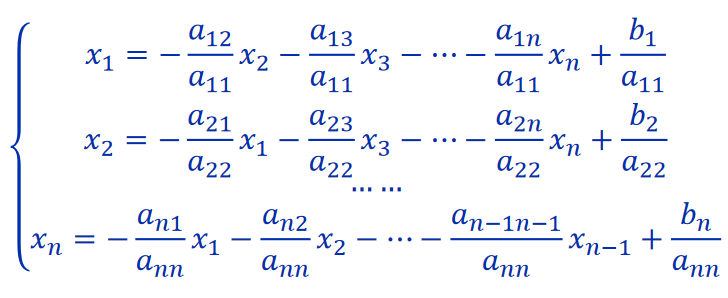
# Описание метода

## Проверка

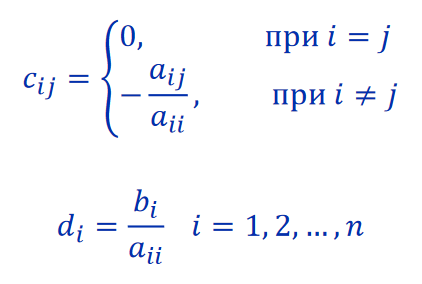
Хотя мы предполагаем, что данные корректны, проверяется условие преобладания диагональных элементов, которое равносильно условию сходимости

## Выражаем новую матрицу

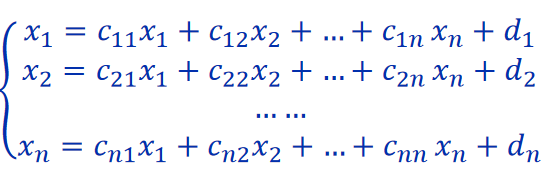




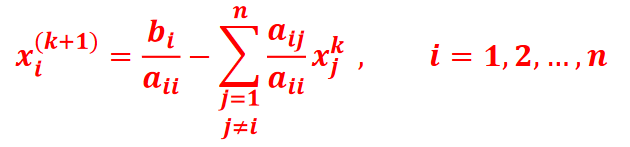
## Обозначим:



## Тогда получим:



## Итоговая формула:



# Листинг программы

#include <iostream>

#include <vector>

void makeDiagMatrix(std::vector<std::vector<double>>& a, std::vector<int>& indX, std::vector<std::vector<double>>& b);

void printMatrix(const std::vector<std::vector<double>>& a);

bool check(const std::vector<std::vector<double>>& a);

std::vector<double> calculate(const std::vector<std::vector<double>>& c, const std::vector<std::vector<double>>& d, const std::vector<std::vector<double>>& x);

double checkPrecision(const std::vector<std::vector<double>>& x);

double abs(double a) {

return a < 0 ? -a : a;

}

int main() {

char read\_from\_file;

std::cout << "Read from file? (Y/n)\n";

std::cin >> read\_from\_file;

if (read\_from\_file == 'Y' || read\_from\_file == 'y') {

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

}

double precision;

std::cout << "Enter precision\n";

std::cin >> precision;

int n;

std::cout << "Enter n\n";

std::cin >> n;

std::vector<std::vector<double>> a(n, std::vector<double>(n));

std::vector<std::vector<double>> b(1, std::vector<double>(n));

std::cout << "Enter matrix A\n";

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

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

std::cin >> a[j][i];

}

}

std::cout << "Enter matrix B\n";

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

std::cin >> b[0][i];

}

std::vector<int> indX(n);

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

indX[i] = i;

}

makeDiagMatrix(a, indX, b);

bool breakFlag = !check(a);

if (breakFlag) {

std::cout << "no diagonal dominance\n";

return 0;

}

std::vector<std::vector<double>> c(n, std::vector<double>(n));

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

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

if (i == j) {

c[i][j] = 0;

continue;

}

c[i][j] = -a[i][j] / a[i][i];

}

}

std::vector<std::vector<double>> d(1, std::vector<double>(n));

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

d[0][i] = b[0][i] / a[i][i];

}

std::vector<std::vector<double>> x;

int counter = 0;

double max = 0;

x.push\_back(d[0]);

do {

x.push\_back(calculate(c, d, x));

counter++;

max = checkPrecision(x);

} while (max > precision);

std::cout << "Result: x\n";

std::vector<double> result(n);

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

result[indX[i]] = x[counter][i];

}

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

std::cout << result[i] << " ";

}

std::cout << std::endl;

std::cout << "Iterations: " << counter << "\n";

std::cout << "Precision: " << max << "\n";

#ifdef DEBUG

std::cout << "DEBUGINFO\n";

std::cout << "x\_n\n";

for (int i = 0; i < x.size(); i++) {

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

std::cout << x[i][j] << " ";

}

std::cout << std::endl;

}

std::cout << "indX\n";

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

std::cout << indX[i] << " ";

}

std::cout << std::endl;

std::cout << "c\n";

printMatrix(c);

std::cout << "d\n";

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

std::cout << d[0][i] << " ";

}

std::cout << std::endl;

#endif

return 0;

}

void makeDiagMatrix(std::vector<std::vector<double>>& a, std::vector<int>& indX, std::vector<std::vector<double>>& b) {

int n = a.size();

for (int i = 0; i < n; i++) { // проходим по строкам

double max = a[i][i];

int maxIndex = i;

for (int j = 0; j < n; j++) { // по столбцам

if (a[j][i] > max) {

max = a[j][i];

maxIndex = j;

}

}

if (maxIndex != i) {

int tmp = indX[i];

indX[i] = indX[maxIndex];

indX[maxIndex] = tmp;

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

double tmp = a[maxIndex][j];

a[maxIndex][j] = a[i][j];

a[i][j] = tmp;

}

}

}

}

void printMatrix(const std::vector<std::vector<double>>& a) {

int n = a.size();

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

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

std::cout << a[j][i] << " ";

}

std::cout << std::endl;

}

}

bool check(const std::vector<std::vector<double>>& a) {

int n = a.size();

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

double sum = -a[i][i];

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

sum += a[j][i];

}

if (a[i][i] < sum) {

return false;

}

}

return true;

}

std::vector<double> calculate(const std::vector<std::vector<double>>& c, const std::vector<std::vector<double>>& d, const std::vector<std::vector<double>>& x) {

int n = c.size();

int counter = x.size();

std::vector<double> newX(n);

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

newX[i] = d[0][i];

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

newX[i] += c[j][i] \* x[counter - 1][j];

}

}

return newX;

}

double checkPrecision(const std::vector<std::vector<double>>& x) {

int n\_x = x.size();

int n = x[0].size();

double max = 0;

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

if (abs(x[n\_x - 1][i] - x[n\_x - 2][i]) > max) {

max = abs(x[n\_x - 1][i] - x[n\_x - 2][i]);

}

}

return max;

}

std::vector<int> get\_reverse\_permutation(const std::vector<int>& permutation) {

int n = permutation.size();

std::vector<int> reverse\_permutation(n);

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

// permutation[i]

}

return reverse\_permutation;

}

// есть перестановка 2 0 1

// обратная перестановка 1 2 0

// 2 3 1 0

// 3 2 0 1

// 2

# Пример программы

## Входные данные

0.01

3

2 2 10

10 1 1

2 10 1

14 12 13

## Результат

Enter precision

Enter n

Enter matrix A

Enter matrix B

Result: x

0.999568 0.99946 0.999316

Iterations: 5

Precision: 0.003084

# DEBUG версия

Если скомпилировать с флагом -D=DEBUG, то будет более детальный вывод.

# Вывод

Я сравнил свой численный метод с другими, мой метод в отличие от прямых позволяет быстро вычислить результат с нужной точностью, не требует хранения в оперативной памяти всех массивов на протяжении всей работы (всего лишь хранение свежего и прошлого). При этом он не такой трудоемкий, как метод Гаусса-Зейделя.

# Используемая литература

1. С. В. Клименков / Конспекты лекций
2. В.В. Кириллов, А.А. Приблуда, С.В. Клименков, Д.Б. Афанасьев / Методические указания к лабораторным работам по курсу «Основы профессиональной деятельности»