

Министерство образования и науки Российской Федерации  
Новосибирский государственный технический университет  
Кафедра прикладной математики

Численные методы  
Практическая работа №1

Факультет: прикладной математики и информатики  
Группа: ПМ-63                      Вариант: 8  
Студент: Кожекин М.В.    Преподаватель: Задорожный

Новосибирск

2018

## 1. Цель работы

Разработать программу решения СЛАУ прямым методом с хранением матрицы в профильном или ленточном формате. Исследовать накопление погрешности и ее зависимость от числа обусловленности. Сравнить реализованный метод по точности получаемого решения и количеству действий с методом Гаусса.

Вариант 8:  $LL^T$  – разложение, матрица в профильном формате

## 2. Анализ

Разложение  $LL^T = A$  имеет следующий вид:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Матрица  $A$  должна быть симметричная, положительно определённая и иметь неотрицательную главную диагональ, что следует из формул разложения:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \quad i \in [1, n]$$
$$l_{ij} = \frac{1}{l_{jj}} \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right), \quad i \in [1, n], j \in [1, i-1]$$

**Остальные формулы:**

- Прямой обход

$$y_i = \frac{1}{l_{ii}} \left( f_i - \sum_{k=1}^n l_{ik} y_k \right), \quad i \in [1, n]$$

- Обратный обход

$$x_i = \frac{1}{l_{ii}} \left( y_i - \sum_{k=2}^n l_{ik} y_k \right), \quad i \in [n, 1]$$

Формат хранения профильный, значит, матрицы будут иметь следующий вид:

$$\begin{array}{cccccccc}
 a_{11} & & & & & & & \\
 & a_{22} & a_{23} & a_{24} & & & & \\
 & \underline{a_{32}} & a_{33} & 0 & a_{35} & a_{36} & & \\
 & \underline{a_{42}} & 0 & a_{44} & a_{45} & 0 & a_{47} & \\
 \rightarrow & a_{53} & a_{54} & a_{55} & a_{56} & 0 & a_{58} & a_{59} \\
 & a_{63} & 0 & a_{65} & a_{66} & 0 & a_{68} & 0 \\
 & & \underline{a_{74}} & 0 & 0 & a_{77} & 0 & a_{79} \\
 & & & \underline{a_{85}} & a_{86} & 0 & a_{88} & 0 \\
 & & & \underline{a_{95}} & 0 & a_{97} & 0 & a_{99}
 \end{array}$$

### 3. Текст программы

Для удобства программа была разбита на следующие модули:

head.h – заголовочный файл, в котором определяется точность вычислений

## profMatrix.h – Класс разреженной матрицы в профилном формате

### profMatrix.cpp – методы класса

vect.h – класс векторов  $X / F$ , для которых используется общая память.

## vect.cpp – методы класса

slae.h – класс СЛАУ

## slae.cpp – методы класса

test.cpp – файл тестов / исследований, использующий библиотеку catch.hpp

head.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <iomanip>

using std::vector;
using std::string;
using std::cout;
using std::endl;

// float || double
typedef double real;
typedef double real sum;
```

profMatrix.h

```
#include "head.h"
```

```
// Symmetric positive-definite sparse matrix
class matrix {

public:

    int getDimention() { return n; }
    int readAFromFile(std::ifstream& fin);
    void writeAToFile(std::ofstream& fout);
    int decomposeChol();

    void generateSparseMatrixA(int n_new, int max_width);
    void createHilbertMatrix(int size);
    void addConditionNumber(int k) { di[0] += pow(10, -k); }

protected:
    vector<real> di, al;
    vector<int> ia;
    int n;
};
```

matrix.cpp

```
#include "profMatrix.h"

// input sparse matrix A (n, di, ia, al)
int matrix::readAFromFile(std::ifstream& fin) {

    fin >> n;

    di.resize(n);
    for (int i = 0; i < di.size(); ++i) {
        fin >> di[i];
    }

    ia.resize(n + 1);
    for (int i = 0; i < ia.size(); ++i) {
        fin >> ia[i];
    }

    al.resize(ia.back());
    for (int i = 0; i < al.size(); ++i) {
        fin >> al[i];
    }

    return 0;
}

// Output sparse matrix A (n, di, ia, al)
void matrix::writeAToFile(std::ofstream& fout) {

    fout << n << endl;

    for (int i = 0; i < di.size(); ++i) {
        fout << di[i] << " ";
    }
    fout << endl;

    for (int i = 0; i < ia.size(); ++i) {
        fout << ia[i] << " ";
    }
}
```

```

fout << endl;

for (int i = 0; i < al.size(); ++i) {
    fout << al[i] << " ";
}

fout << endl;
}

// LL' decomposition of the A matrix
int matrix::decomposeChol() {

    real_sum tmp;

    // Идём построчно в верхнем треугольнике, что эквивалентно
    // Обходу нижнего треугольника по столбцам вниз, начиная с первого
    for (int i = 0; i < n; ++i) {

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        // Рассчёт элементов нижнего треугольника
        for (int k = i0; k < i1; ++k, ++j) {

            tmp = 0.0;
            int elem_i = ia[i]; // номер элемента i-й строки
            int elem_j = ia[j]; // номер элемента j-й строки
            int beg_i = i - (ia[i + 1] - ia[i]); // индекс первого элемента i-й
строки
            int beg_j = j - (ia[j + 1] - ia[j]); // индекс первого элемента j-й
строки

            int length_dif = beg_j - beg_i;

            if (length_dif >= 0)
                elem_i += length_dif;
            else
                elem_j += abs(length_dif);

            for (elem_i; elem_i < k; ++elem_i, ++elem_j)
                tmp += al[elem_i] * al[elem_j];

            al[k] = (al[k] - tmp) / di[j];
        }

        // Рассчёт диагонального элемента
        tmp = 0.0;
        for (int k = i0; k < i1; ++k)
            tmp += al[k] * al[k];
        di[i] = sqrt(di[i] - tmp);
    }

    return 0;
}

```

```

// Generate sparse matrix A
void matrix::generateSparseMatrixA(int n_new, int max_width) {

    n = n_new;
    di.resize(n, 0);
    ia.resize(n + 1, 0);

    int prev = 0;
    int tmp_width; // Длина профиля строки

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

        if (max_width >= i)
            tmp_width = i;
        else
            tmp_width = max_width;

        prev += tmp_width;

        ia[i + 1] = prev;

        int i0 = ia[i];
        int i1 = ia[i + 1];

        for (int k = i0; k < i1; ++k) {

            al.push_back(-(rand() % 5));

        }

    }

    vector<real_sum> tmp(n, 0);

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

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {

            tmp[i] += al[k];
            tmp[j] += al[k];

        }

    }

    for (int i = 0; i < n; ++i)
        di[i] = -tmp[i];
}

// Create D. Hilbert's matrix
void matrix::createHilbertMatrix(int size) {

    n = size;
    di.resize(n);
    ia.resize(n + 1);
    al.resize(n*(n - 1) / 2); // число элементов нижнего треугольника

```

```

    ia[0] = 0;
    ia[1] = 0;
    for (int i = 0; i < ia.size() - 1; ++i)
        ia[i + 1] = ia[i] + i;

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

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);
        for (int k = i0; k < i1; ++k, ++j) {

            al[k] = 1 / real(i + j + 1);
        }
        di[i] = 1 / real(i + j + 1);
    }
}

```

## vect.h

```

#include "head.h"

// Vectors F, x, y = L\F, x = L'y
class vect {

public:

    void getVectX(vector<real> &x) { x = F; };
    int readVectFromFile(std::ifstream& fin, int size);
    void generateVectX(int size);
    void writeVectToFile(std::ofstream& fout, char *str);
    void writexCompError(std::ofstream& fout, char *str);
    void writeTableToFile(std::ofstream& fout);
    bool isXcorrect();

protected:
    vector<real> F;
};

```

## vect.cpp

```

#include "head.h"
#include "vect.h"

// Input of vector
int vect::readVectFromFile(std::ifstream& fin, int size) {

    F.resize(size);
    for (int i = 0; i < size; ++i) {
        fin >> F[i];
    }

    return 0;
}

// Creating vector X = (1,2,...n)'

```

```

void vect::generateVectX(int size) {
    F.resize(size);
    for (int i = 0; i < size; ++i) {
        F[i] = i + 1;
    }
}

// Output of results
void vect::writeVectToFile(std::ofstream& fout, char *str) {

    fout << str << endl;
    fout << std::fixed << std::setprecision(std::numeric_limits<real>::digits10 + 1);
    for (int i = 0; i < F.size(); ++i) {
        fout << F[i] << endl;
    }
}

// generates 1/3 of table in research
void vect::writeTableToFile(std::ofstream& fout) {

    fout << std::fixed << std::setprecision(std::numeric_limits<real>::digits10 + 1);
    for (int i = 0; i < F.size(); ++i) {
        fout << F[i] << " ";
    }
    fout << " \t";

    fout << std::scientific;
    for (int i = 0; i < F.size(); ++i) {
        fout << F[i] - real(i + 1) << " ";
    }
    fout << " \t" << endl;
}

// Output of computational error
void vect::writexCompError(std::ofstream& fout, char *str) {

    fout << str << endl;
    fout << std::scientific;
    for (int i = 0; i < F.size(); ++i) {
        fout << F[i] - real(i + 1) << endl;
    }
}

// Check if X is almost equal to (1,2,...,n)'
bool vect::isXcorrect() {

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

        if (abs(F[i] - (real)(i + 1)) > std::numeric_limits<real>::digits10 + 2)
            return false;
    }

    return true;
}

```



## slae.h

```
#include "head.h"
#include "profMatrix.h"
#include "vect.h"

// System of linear algebraic equations
class SLAE : public matrix, public vect {

public:

    void writeMatrixToFile(std::ofstream& fout, char* str);

    void execDirectTraversal();
    void execReverseTraversal();

    void convAToDense();
    void convLToDense();
    void mult();
    void calcGauss();
    void calcGaussAdvanced();

private:
    vector<vector<double>>> A;
};
```

## slae.cpp

```
#include "head.h"
#include "slae.h"

// Output dense matrix A
void SLAE::writeMatrixToFile(std::ofstream& fout, char *str) {

    fout << std::fixed << std::setprecision(std::numeric_limits<real>::digits10 + 1) <<
    str << endl;
    for (int i = 0; i < A.size(); ++i) {
        for (int j = 0; j < A.size(); ++j) {
            fout << A[i][j] << "\t";
        }
        fout << ";" << endl;
    }
    fout << endl;
}

// Converting sparse matrix to dense format
void SLAE::convAToDense() {

    A.clear();
    A.resize(n);
    for (int i = 0; i < n; ++i) {
        A[i].resize(n, 0);
    }

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

        A[i][i] = di[i];
    }
}
```

```

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {

            A[i][j] = al[k];
            A[j][i] = al[k];

        }
    }
}

// Converting sparse matrix to dense format
void SLAE::convLToDense() {

    A.clear();
    A.resize(n);
    for (int i = 0; i < n; ++i) {
        A[i].resize(n, 0);
    }

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

        A[i][i] = di[i];
        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {

            A[i][j] = al[k];

        }
    }
}

// A*x = F
void SLAE::mult() {

    vector<real_sum> F_tmp(n, 0);

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

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {

            F_tmp[i] += al[k] * F[j]; // В F лежит x
            F_tmp[j] += al[k] * F[i];

        }

    }

    for (int i = 0; i < n; ++i) {
        F_tmp[i] += di[i] * F[i];
    }
}

```

```

        for (int i = 0; i < n; ++i)
            F[i] = real(F_tmp[i]);
    }

// Direct traversal (calculating y):      Ly = F          y = L\F
void SLAE::execDirectTraversal() {

    real_sum tmp;

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

        tmp = 0.0;
        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {
            tmp += al[k] * F[j];
        }

        F[i] = (F[i] - tmp) / di[i];
    }
}

// Reverse traversal (calculating x):      L'x = y          x = L'\y
void SLAE::execReverseTraversal() {

    vector<real> x;
    x.resize(n, 0);

    for (int i = n - 1; i >= 0; --i) {

        x[i] = F[i] / di[i];

        int i0 = ia[i];
        int i1 = ia[i + 1];
        int j = i - (i1 - i0);

        for (int k = i0; k < i1; ++k, ++j) {
            F[j] -= al[k] * x[i];
        }
    }

    F = x;
}

// Gaussian elemination
void SLAE::calcGauss() {

    // Приведение к верхне-треугольному виду
    for (int j = 0; j < A.size(); ++j) {
        for (int i = j + 1; i < A.size(); ++i) {

            real toMult = A[i][j] / A[j][j]; // Коэффициент, на который надо
умножить строку

```

```

        for (int k = 0; k < A.size(); ++k) // Отняли стоку
            A[i][k] -= toMult * A[j][k];

        F[i] -= toMult * F[j];
    }
}

// Обратный обход
vector<real> x;
x.resize(A.size(), 0);

for (int i = n - 1; i >= 0; --i) {

    real_sum tmp = 0.0;
    for (int j = i + 1; j < A.size(); ++j) {
        tmp += A[i][j] * x[j];
    }
    x[i] = (F[i] - tmp) / A[i][i];
}
F = x;
}

// Gaussian elemination with leading element selection
void SLAE::calcGaussAdvanced() {

    // Приведение к верхне-треугольному виду
    for (int j = 0; j < A.size(); ++j) {
        for (int i = j + 1; i < A.size(); ++i) {

            int max = -DBL_MAX, row = i;
            for (int k = i; k < A.size(); ++k) // Ищем строку с ведущим элементом
                if (A[k][j] > max) {
                    max = A[k][j];
                    row = k;
                }

            std::swap(A[row], A[i]); // Меняем строку с ведущим элементом на i-ю
            std::swap(F[row], F[i]);

            real toMult = A[i][j] / A[j][j]; // Коэффициент, на который надо
умножить строку

            for (int k = 0; k < A.size(); ++k) // Отняли стоку
                A[i][k] -= toMult * A[j][k];

            F[i] -= toMult * F[j];
        }
    }

    // Обратный обход
    vector<real> x;
    x.resize(A.size(), 0);

    for (int i = n - 1; i >= 0; --i) {

        real_sum tmp = 0.0;
        for (int j = i + 1; j < A.size(); ++j) {

```

```

        tmp += A[i][j] * x[j];
    }
    x[i] = (F[i] - tmp) / A[i][i];
}
F = x;
}

```

## test.cpp

```

#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include "head.h"
#include "slae.h"

TEST_CASE("Condition number research") {

    std::ifstream fin;
    std::ofstream fout;
    fout.open("x_condNum.txt");
    int max_width = 10, K_max = 15;

    /*matrix m;
    m.generateSparseMatrixA(size, 4);
    fout.open("A.txt");
    m.writeAToFile(fout);
    fout.close();*/
    cout << "Condition number research:" << endl;

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

        fin.open("A.txt");
        cout << "k = " << k << endl;
        SLAE slae;

        slae.readAFromFile(fin);
        slae.addConditionNumber(k);
        slae.generateVectX(slae.getDimention());

        slae.mult();

        slae.decomposeChol();
        slae.execDirectTraversal();
        slae.execReverseTraversal();

        slae.writeTableToFile(fout);
        fin.close();
        CHECK(slae.isXcorrect());
    }

    fout.close();
}

TEST_CASE("Hilbert matrix research") {

    std::ofstream fout;
    fout.open("x_Hilbert.txt");
    int K_max = 15;

```

```

cout << "Hilbert matrix research:" << endl;
for (int k = 0; k < K_max; ++k) {

    cout << "k = " << k << endl;
    SLAE slae;

    slae.createHilbertMatrix(k);
    slae.generateVectX(k);
    slae.mult();
    //slae.convAToDense();
    //slae.writeMatrixToFile(fout, "H:");

    slae.decomposeChol();
    slae.execDirectTraversal();
    slae.execReverseTraversal();

    slae.writeTableToFile(fout);
    //slae.writeVectToFile(fout, "Vector x:");
    CHECK(slae.isXcorrect());
}

fout.close();
}

```

```

TEST_CASE("Gauss' method research") {

    std::ifstream fin;
    std::ofstream fout;
    fout.open("x_Gauss.txt");
    int K_max = 15;

    cout << "Gauss' method research:" << endl;
    for (int k = 0; k < K_max; ++k) {

        fin.open("A.txt");
        cout << "k = " << k << endl;
        SLAE slae;

        slae.readAFromFile(fin);
        slae.addConditionNumber(k);
        slae.generateVectX(slae.getDimension());

        slae.mult();

        slae.convAToDense();
        slae.calcGauss();

        slae.writeTableToFile(fout);
        //slae.writeVectToFile(fout, "Vector x:");
        fin.close();
        CHECK(slae.isXcorrect());
    }

    fout.close();
}

```

```

TEST_CASE("Advanced Gauss' method research") {

    std::ifstream fin;
    std::ofstream fout;
    fout.open("x_AdvGauss.txt");
    int K_max = 15;

    cout << "Advanced Gauss' method research:" << endl;
    for (int k = 0; k < K_max; ++k) {

        fin.open("A.txt");
        cout << "k = " << k << endl;
        SLAE slae;

        slae.readAFromFile(fin);
        slae.addConditionNumber(k);
        slae.generateVectX(slae.getDimention());

        slae.mult();

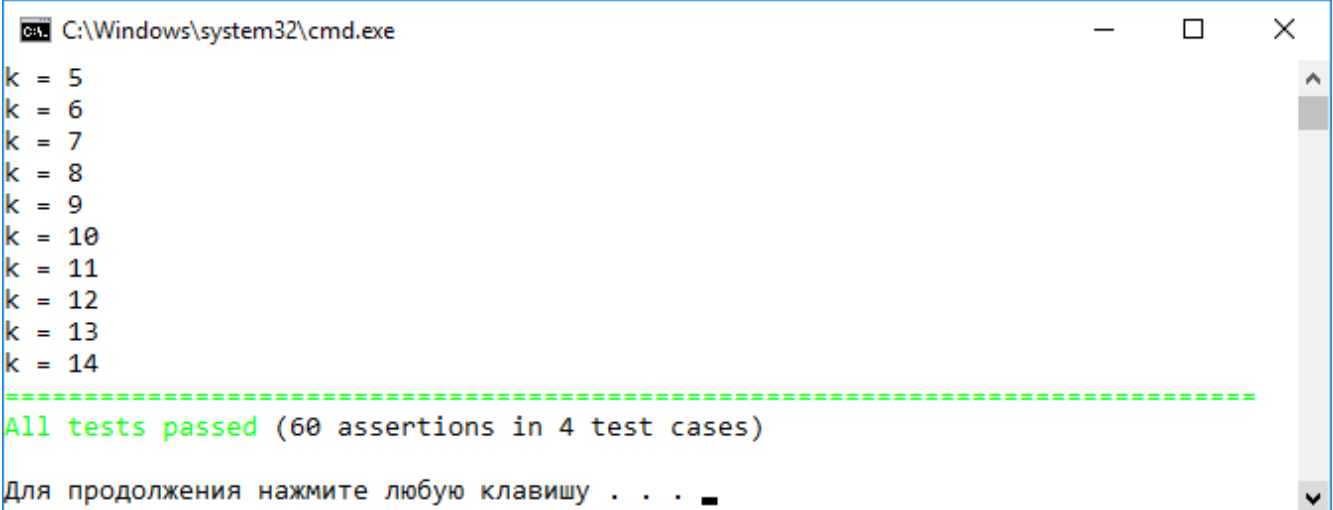
        slae.convAToDense();
        slae.calcGauss();

        slae.writeTableToFile(fout);
        //slae.writeVectToFile(fout, "Vector x:");
        fin.close();
        CHECK(slae.isXcorrect());
    }

    fout.close();
}

```

## 4. Тесты



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is as follows:

```

k = 5
k = 6
k = 7
k = 8
k = 9
k = 10
k = 11
k = 12
k = 13
k = 14
=====
All tests passed (60 assertions in 4 test cases)
Для продолжения нажмите любую клавишу . . .

```

## 5. Оценить влияние увеличения числа обусловленности на точность решения

$6 + 10^{-k}$	-1	-2	0	-3	0	0	0	0	0		1		$-17 + 10^{-k}$
-1	12	-4	-4	-3	0	0	0	0	0		2		-20
-2	-4	13	-4	-2	0	-1	0	0	0		3		-4
0	-4	-4	16	-4	-1	-1	-2	0	0		4		-5
-3	-3	-2	-4	17	-2	0	-1	-2	0	*	5	=	16
0	0	0	-1	-2	10	-2	-1	-3	-1		6		-13
0	0	-1	-1	0	-2	11	-4	-2	-1		7		-2
0	0	0	-2	-1	-1	-4	13	-2	-3		8		9
0	0	0	0	-2	-3	-2	-2	9	0		9		23
0	0	0	0	0	-1	-1	-3	0	5		10		13

k	$x^k$ (одинарная точность)	$x^* - x^k$ (одинарная точность)	$x^k$ (двойная точность)	$x^* - x^k$ (двойная точность)	$x^k$ (скалярное произведение)	$x^* - x^k$ (скалярное произведение)
0	0.9999877	-1.2338161e-05	1.0000000000000020	1.9984014443252818e-15	1.0000136	1.3589859e-05
	1.9999857	-1.4305115e-05	2.0000000000000022	2.2204460492503131e-15	2.0000160	1.5974045e-05
	2.9999859	-1.4066696e-05	3.0000000000000027	2.6645352591003757e-15	3.0000160	1.5974045e-05
	3.9999855	-1.4543533e-05	4.0000000000000027	2.6645352591003757e-15	4.0000167	1.6689301e-05
	4.9999852	-1.4781952e-05	5.0000000000000018	1.7763568394002505e-15	5.0000157	1.5735626e-05
	5.9999852	-1.4781952e-05	6.0000000000000027	2.6645352591003757e-15	6.0000172	1.7166138e-05
	6.9999857	-1.4305115e-05	7.0000000000000044	4.4408920985006262e-15	7.0000176	1.7642975e-05
	7.9999847	-1.5258789e-05	8.0000000000000036	3.5527136788005009e-15	8.0000181	1.8119812e-05
	8.9999847	-1.5258789e-05	9.0000000000000036	3.5527136788005009e-15	9.0000172	1.7166138e-05
	9.9999847	-1.5258789e-05	10.0000000000000036	3.5527136788005009e-15	10.0000172	1.7166138e-05
1	0.9998931	-1.0693073e-04	1.00000000000002358	2.3581137043038325e-13	1.0000261	2.6106834e-05
	1.9998912	-1.0883808e-04	2.00000000000002394	2.3936408410918375e-13	2.0000262	2.6226044e-05
	2.9998910	-1.0895729e-04	3.00000000000002389	2.3891999489933369e-13	3.0000262	2.6226044e-05
	3.9998913	-1.0871887e-04	4.00000000000002398	2.3980817331903381e-13	4.0000267	2.6702881e-05
	4.9998913	-1.0871887e-04	5.00000000000002398	2.3980817331903381e-13	5.0000267	2.6702881e-05
	5.9998913	-1.0871887e-04	6.00000000000002416	2.4158453015843406e-13	6.0000267	2.6702881e-05
	6.9998913	-1.0871887e-04	7.00000000000002416	2.4158453015843406e-13	7.0000272	2.7179718e-05
	7.9998903	-1.0967255e-04	8.00000000000002416	2.4158453015843406e-13	8.0000277	2.7656555e-05
	8.9998913	-1.0871887e-04	9.00000000000002434	2.4336088699783431e-13	9.0000267	2.6702881e-05
	9.9998903	-1.0967255e-04	10.00000000000002434	2.4336088699783431e-13	10.0000267	2.6702881e-05
2	1.0002869	2.8693676e-04	1.00000000000003568	3.5682568011452531e-13	0.9995165	-4.8351288e-04
	2.0002875	2.8753281e-04	2.00000000000003575	3.5749181392930041e-13	1.9995158	-4.8422813e-04
	3.0002878	2.8777122e-04	3.00000000000003570	3.5704772471945034e-13	2.9995158	-4.8422813e-04
	4.0002875	2.8753281e-04	4.00000000000003570	3.5704772471945034e-13	3.9995155	-4.8446655e-04
	5.0002875	2.8753281e-04	5.00000000000003579	3.5793590313915047e-13	4.9995155	-4.8446655e-04
	6.0002875	2.8753281e-04	6.00000000000003553	3.5527136788005009e-13	5.9995151	-4.8494339e-04
	7.0002875	2.8753281e-04	7.00000000000003535	3.5349501104064984e-13	6.9995146	-4.8542023e-04
	8.0002871	2.8705597e-04	8.00000000000003553	3.5527136788005009e-13	7.9995151	-4.8494339e-04
	9.0002871	2.8705597e-04	9.00000000000003553	3.5527136788005009e-13	8.9995155	-4.8446655e-04
	10.0002871	2.8705597e-04	10.00000000000003553	3.5527136788005009e-13	9.9995146	-4.8542023e-04
3	1.0047778	4.7777891e-03	0.9999999999804591	-1.9540924434124918e-11	1.0001043	1.0430813e-04
	2.0047791	4.7791004e-03	1.9999999999804554	-1.9544588170106181e-11	2.0001049	1.0490417e-04
	3.0047791	4.7791004e-03	2.9999999999804561	-1.9543922036291406e-11	3.0001049	1.0490417e-04
	4.0047793	4.7793388e-03	3.9999999999804547	-1.9545254303920956e-11	4.0001049	1.0490417e-04
	5.0047789	4.7788620e-03	4.9999999999804547	-1.9545254303920956e-11	5.0001044	1.0442734e-04
	6.0047789	4.7788620e-03	5.9999999999804530	-1.9547030660760356e-11	6.0001040	1.0395050e-04
	7.0047789	4.7788620e-03	6.9999999999804521	-1.9547918839180056e-11	7.0001040	1.0395050e-04



	8.0047789 9.0047789 10.0047789	4.7788620e-03 4.7788620e-03 4.7788620e-03	7.999999999804521 8.999999999804530 9.999999999804530	-1.9547918839180056e-11 -1.9547030660760356e-11 -1.9547030660760356e-11	8.0001040 9.0001040 10.0001040	1.0395050e-04 1.0395050e-04 1.0395050e-04
4	1.2376183 2.2376220 3.2376220 4.2376227 5.2376227 6.2376242 7.2376237 8.2376242 9.2376242 10.2376242	2.3761833e-01 2.3762202e-01 2.3762202e-01 2.3762274e-01 2.3762274e-01 2.3762417e-01 2.3762369e-01 2.3762417e-01 2.3762417e-01 2.3762417e-01	1.0000000000000004 2.0000000000000004 3.0000000000000009 4.0000000000000009 5.0000000000000000 5.9999999999999991 7.0000000000000009 8.0000000000000018 9.0000000000000000 10.0000000000000000	4.4408920985006262e-16 4.4408920985006262e-16 8.8817841970012523e-16 8.8817841970012523e-16 0.000000000000000e+00 -8.8817841970012523e-16 8.8817841970012523e-16 1.7763568394002505e-15 0.000000000000000e+00 0.000000000000000e+00	0.8591874 1.8591852 2.8591852 3.8591847 4.8591852 5.8591852 6.8591838 7.8591838 8.8591843 9.8591833	-1.4081258e-01 -1.4081478e-01 -1.4081478e-01 -1.4081526e-01 -1.4081478e-01 -1.4081478e-01 -1.4081621e-01 -1.4081621e-01 -1.4081573e-01 -1.4081669e-01
5	0.6666675 1.6666670 2.6666670 3.6666670 4.6666670 5.6666679 6.6666679 7.6666675 8.6666679 9.6666670	-3.3333254e-01 -3.3333302e-01 -3.3333302e-01 -3.3333302e-01 -3.3333302e-01 -3.3333206e-01 -3.3333206e-01 -3.3333254e-01 -3.3333206e-01 -3.3333302e-01	1.0000000003552727 2.0000000003552740 3.0000000003552740 4.0000000003552740 5.0000000003552731 6.0000000003552749 7.0000000003552740 8.0000000003552731 9.0000000003552731 10.0000000003552749	3.5527270014767964e-10 3.5527403241530919e-10 3.5527403241530919e-10 3.5527403241530919e-10 3.5527314423688949e-10 3.5527492059372889e-10 3.5527403241530919e-10 3.5527314423688949e-10 3.5527314423688949e-10 3.5527492059372889e-10	0.6333838 1.6333835 2.6333835 3.6333835 4.6333833 5.6333833 6.6333833 7.6333833 8.6333838 9.6333828	-3.6661619e-01 -3.6661649e-01 -3.6661649e-01 -3.6661649e-01 -3.6661673e-01 -3.6661673e-01 -3.6661673e-01 -3.6661673e-01 -3.6661625e-01 -3.6661720e-01
6			1.0000000017763626 2.0000000017763631 3.0000000017763635 4.0000000017763631 5.0000000017763631 6.0000000017763622 7.0000000017763613 8.0000000017763604 9.0000000017763622 10.0000000017763604	1.7763626125599785e-09 1.7763630566491884e-09 1.7763635007383982e-09 1.7763630566491884e-09 1.7763630566491884e-09 1.7763621684707687e-09 1.7763612802923490e-09 1.7763603921139293e-09 1.7763621684707687e-09 1.7763603921139293e-09		
7			1.0000000355271372 2.0000000355271386 3.0000000355271386 4.0000000355271386 5.0000000355271377 6.0000000355271368 7.0000000355271359 8.0000000355271368 9.0000000355271386 10.0000000355271386	3.5527137232094219e-08 3.5527138564361849e-08 3.5527138564361849e-08 3.5527138564361849e-08 3.5527137676183429e-08 3.5527136788005009e-08 3.5527135899826590e-08 3.5527136788005009e-08 3.5527138564361849e-08 3.5527138564361849e-08		
8			0.9999969801949719 1.9999969801949664 2.9999969801949669 3.9999969801949660 4.9999969801949664 5.9999969801949637 6.9999969801949637 7.9999969801949637 8.9999969801949646 9.9999969801949646	-3.0198050281482480e-06 -3.0198050335883408e-06 -3.0198050331442516e-06 -3.0198050340324301e-06 -3.0198050335883408e-06 -3.0198050362528761e-06 -3.0198050362528761e-06 -3.0198050362528761e-06 -3.0198050353646977e-06 -3.0198050353646977e-06		
9			0.9999715783939044 1.9999715783939001 2.9999715783938998 3.9999715783938985 4.9999715783938994 5.9999715783938967 6.9999715783938949 7.9999715783938958	-2.8421606095618834e-05 -2.8421606099948704e-05 -2.8421606100170749e-05 -2.8421606101503016e-05 -2.8421606100614838e-05 -2.8421606103279373e-05 -2.8421606105055730e-05 -2.8421606104167552e-05		

			8.9999715783938967 9.9999715783938949	-2.8421606103279373e-05 -2.8421606105055730e-05		
10			0.9998401335772825 1.9998401335772795 2.9998401335772793 3.9998401335772789 4.9998401335772797 5.9998401335772797 6.9998401335772789 7.9998401335772780 8.9998401335772780 9.9998401335772780	-1.5986642271748064e-04 -1.5986642272047824e-04 -1.5986642272070029e-04 -1.5986642272114437e-04 -1.5986642272025620e-04 -1.5986642272025620e-04 -1.5986642272114437e-04 -1.5986642272203255e-04 -1.5986642272203255e-04 -1.5986642272203255e-04		
11			1.0001776514478631 2.0001776514478631 3.0001776514478635 4.0001776514478626 5.0001776514478635 6.0001776514478617 7.0001776514478635 8.0001776514478635 9.0001776514478635 10.0001776514478617	1.7765144786308085e-04 1.7765144786308085e-04 1.7765144786352494e-04 1.7765144786263676e-04 1.7765144786352494e-04 1.7765144786174858e-04 1.7765144786352494e-04 1.7765144786352494e-04 1.7765144786352494e-04 1.7765144786174858e-04		
12			1.0000000000000013 2.0000000000000009 3.0000000000000009 4.0000000000000009 5.0000000000000009 5.9999999999999982 6.9999999999999982 7.9999999999999991 9.0000000000000000 10.0000000000000000	1.3322676295501878e-15 8.8817841970012523e-16 8.8817841970012523e-16 8.8817841970012523e-16 8.8817841970012523e-16 -1.7763568394002505e-15 -1.7763568394002505e-15 -8.8817841970012523e-16 0.0000000000000000e+00 0.0000000000000000e+00		
13			1.2429906542056020 2.2429906542056064 3.2429906542056068 4.2429906542056068 5.2429906542056059 6.2429906542056051 7.2429906542056068 8.2429906542056059 9.2429906542056059 10.2429906542056059	2.4299065420560195e-01 2.4299065420560639e-01 2.4299065420560684e-01 2.4299065420560684e-01 2.4299065420560595e-01 2.4299065420560506e-01 2.4299065420560684e-01 2.4299065420560595e-01 2.4299065420560595e-01 2.4299065420560595e-01		

## 6. Исследования на матрицах Гильберта

k	$x^k$ (одинарная точность)	$x^* - x^k$ (одинарная точность)	$x^k$ (двойная точность)	$x^* - x^k$ (двойная точность)	$x^k$ (скалярное произведение)	$x^* - x^k$ (скалярное произведение)
1	1.0000000	0.0000000e+00	1.0000000000000000	0.0000000000000000e+00	1.0000000	0.0000000e+00
2	0.9999996 2.0000007	-3.5762787e-07 7.1525574e-07	1.0000000000000009 1.9999999999999984	8.8817841970012523e-16 -1.5543122344752192e-15	0.9999996 2.0000007	-3.5762787e-07 7.1525574e-07
3	1.0000012 1.9999903 3.0000105	1.1920929e-06 -9.6559525e-06 1.0490417e-05	1.0000000000000080 1.9999999999999574 3.0000000000000404	7.9936057773011271e-15 -4.2632564145606011e-14 4.0412118096355698e-14	1.0000005 1.9999952 3.0000055	4.7683716e-07 -4.7683716e-06 5.4836273e-06
4	0.9999710 2.0003226 2.9992268 4.0005012	-2.8967857e-05 3.2258034e-04 -7.7319145e-04 5.0115585e-04	1.0000000000000777 1.9999999999991689 3.0000000000019460 3.9999999999987561	7.7715611723760958e-14 -8.3111295623439219e-13 1.9459989175629744e-12 -1.2438938767900254e-12	1.0000229 1.9997396 3.0006270 3.9995930	2.2888184e-05 -2.6035309e-04 6.2704086e-04 -4.0698051e-04

5	0.9999734	-2.6583672e-05	1.0000000000008673	8.6730622683717229e-13	0.9999077	-9.2267990e-05
	2.0005662	5.6624413e-04	1.999999999845264	-1.5473622383410657e-11	2.0015869	1.5869141e-03
	2.9974306	-2.5694370e-03	3.0000000000649170	6.4916960695882153e-11	2.9935102	-6.4897537e-03
	4.0039706	3.9706230e-03	3.999999999037099	-9.6290087014949677e-11	4.0094452	9.4451904e-03
	4.9980354	-1.9645691e-03	5.0000000000465121	4.6512127482856158e-11	4.9955039	-4.4960976e-03
6	1.0018680	1.8680096e-03	1.0000000000002986	2.9864999362416711e-13	0.9996994	-3.0064583e-04
	1.9502965	-4.9703479e-02	2.0000000000002411	2.4114044094858400e-13	2.0076928	7.6928139e-03
	3.3209269	3.2092690e-01	2.999999999599867	-4.0013325985910342e-11	2.9512355	-4.8764467e-02
	3.1931067	-8.0689335e-01	4.0000000001744693	1.7446932787379410e-10	4.1214051	1.2140512e-01
	5.8677635	8.6776352e-01	4.999999997495621	-2.5043789264600491e-10	4.8701615	-1.2983847e-01
7	5.6650724	-3.3492756e-01	6.0000000001163318	1.1633183305548300e-10	6.0499530	4.9952984e-02
	0.9966615	-3.3384562e-03	1.0000000000033595	3.3595348725157237e-12	1.0051255	5.1255226e-03
	2.1497023	1.4970231e-01	1.999999998978635	-1.0213652146262575e-10	1.7797129	-2.2028708e-01
	1.4473505	-1.5526495e+00	3.0000000007678587	7.6785866554018867e-10	5.2181320	2.2181320e+00
	10.3579378	6.3579378e+00	3.999999976380343	-2.3619657341100719e-09	-4.8856635	-8.8856640e+00
8	-7.1180482	-1.2118048e+01	5.0000000034476964	3.4476963506335778e-09	21.6501884	1.6650188e+01
	16.7937546	1.0793755e+01	5.999999976281426	-2.3718573771702722e-09	-8.6306229	-1.4630623e+01
	3.3684211	-3.6315789e+00	7.0000000006161223	6.1612226431861927e-10	11.8681908	4.8681908e+00
			0.9999999995919329	-4.0806713563767971e-10	1.0023696	2.3696423e-03
			2.0000000217492935	2.1749293477313358e-08	1.7628487	-2.3715127e-01
9			2.9999997172193300	-2.8278066999121165e-07	7.0400681	4.0400681e+00
			4.0000015252289680	1.5252289680134368e-06	-21.5438633	-2.5543863e+01
			4.9999959045643640	-4.0954356359534927e-06	81.3339157	7.6333916e+01
			6.0000057824924999	5.7824924999394511e-06	-110.5036011	-1.1650360e+02
			6.9999958921310892	-4.1078689108076105e-06	94.8450394	8.7845039e+01
10			8.0000011573055581	1.1573055580527125e-06	-17.9466000	-2.5946600e+01
			0.999999997624809	-2.3751911548686166e-10		
			2.0000000133720408	1.3372040807979602e-08		
			2.9999998100063099	-1.8999369011396539e-07		
			4.0000011620391147	1.1620391147104669e-06		
11			4.999962916955267	-3.7083044732710846e-06		
			6.0000066669470353	6.6669470353275528e-06		
			6.9999931934366524	-6.8065633476166454e-06		
			8.0000036833801609	3.6833801608793237e-06		
			8.9999991793771059	-8.2062289408213474e-07		
12			0.9999999827822252	-1.7217774761491000e-08		
			2.0000014944041151	1.4944041151210286e-06		
			2.9999680586844506	-3.1941315549399718e-05		
			4.0002911732274855	2.9117322748550833e-04		
			4.9986081321770026	-1.3918678229973835e-03		
			6.0038329856153796	3.8329856153795916e-03		
			6.9937021370022761	-6.2978629977239464e-03		
			8.0060934826165777	6.0934826165777167e-03		
			8.9967977316105490	-3.2022683894510351e-03		
			10.0007048342844840	7.0483428448397945e-04		
			0.9999999130257904	-8.6974209612122877e-08		
			2.0000090692165484	9.0692165484185239e-06		
			2.9997653936670177	-2.3460633298233446e-04		
			4.0026174603797786	2.6174603797786133e-03		
			4.9844317559150273	-1.5568244084972704e-02		
			6.0546666530123154	5.4666653012315436e-02		
			6.8810932046289937	-1.1890679537100635e-01		
			8.1619690009017276	1.6196900090172761e-01		
			8.8655538770166675	-1.3444612298333247e-01		
			10.0621680232049293	6.2168023204929312e-02		
			10.9877255960191107	-1.2274403980889304e		
			0.9999998067128988	-1.9328710121335746e-07		
			2.0000249345012961	2.4934501296147715e-05		
			2.9992053510168017	-7.9464898319825394e-04		
			4.0109390878838997	1.0939087883899745e-02		
			4.9191491580356912	-8.0850841964308806e-02		

			6.3575984978839815 5.9981989320167459 9.8216447466712129 6.8561466154002595 11.5752596672057315 10.3432058246810783 12.1186275245889306	3.5759849788398146e-01 -1.0018010679832541e+00 1.8216447466712129e+00 -2.1438533845997405e+00 1.5752596672057315e+00 -6.5679417531892170e		
--	--	--	---	--	--	--

## 7. Исследование другого алгоритма (LDU)

Разложение матрицы LDU:

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \cdot \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$D_{ii} = A_{ii} - \sum_{k=1}^{i-1} D_{ik} D_{jk} D_{kk}, \quad i \in [1, n]$$

$$L_{ij} = \frac{1}{D_{jj}} \left( A_{ij} - \sum_{k=1}^{j-1} L_{ik} D_{kk} U_{kj} \right), \quad j \in [1, n-1], i \in [j+1, n]$$

$$U_{ij} = \frac{1}{D_{ii}} \left( A_{ij} - \sum_{k=1}^{j-1} L_{ik} D_{kk} U_{kj} \right), \quad i \in [1, n-1], j \in [i+1, n]$$

Сравнение по алгоритмической сложности:

Метод	L	D	U	Прямой обход	Обратный обход	Всего
$LL^T$	$\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$	-	-	$n^2$	$n^2$	$\frac{n^3}{3} + 3n^2 + \frac{8n}{3}$
LDU	$\frac{n^3 - n^2}{2}$	$\frac{3n^2 - n}{2}$	$\frac{n^3 - n^2}{2}$	$n^2 + n$	$n^2$	$n^3 + \frac{5n^2}{2} + \frac{n}{2}$

Вывод:

Метод решения СЛАУ при помощи  $LL^T$  разложения быстрее в 3 раза, чем метод, использующий LDU разложение, однако, при его использовании появляются дополнительные ограничения: матрица A должна быть положительно определённой и симметричной.

## 8. Исследование метода Гаусса

### Сравнение по алгоритмической сложности:

Метод	L	Прямой обход	Обратный обход	Всего
$LL^T$	$\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$	$n^2$	$n^2$	$\frac{n^3}{3} + 3n^2 + \frac{8n}{3}$
Метод Гаусса	-	$\frac{2n^3}{3} + \frac{n^2}{2} + \frac{5n}{6}$	$n^2$	$\frac{2n^3}{3} + \frac{3n^2}{2} + \frac{5n}{6}$
Метод Гаусса с ведущим главным элементом	-	$\frac{2n^3}{3} + \frac{5n^2}{2} + \frac{5n}{6}$	$n^2$	$\frac{2n^3}{3} + \frac{7n^2}{2} + \frac{5n}{6}$

Вывод:

1. Метод решения СЛАУ при помощи  $LL^T$  разложения в 2 раза быстрее, чем метод Гаусса, однако, для  $LL^T$  разложения требуется положительная определённость и симметричность матрицы A.
2. Разница по быстродействию между обычным методом Гаусса и методом Гаусса с выбором ведущего элемента незначительна. Однако выбор ведущего элемента позволяет нам иметь нулевые элементы на главной диагонали.

### Сравнение по точности решения (двойная точность):

К	Гаусс		$LL^T$	
0	1.0000000000000000 2.0000000000000000 3.0000000000000004 4.0000000000000000 5.0000000000000000 6.0000000000000000 7.0000000000000000 8.0000000000000000 9.0000000000000000 10.0000000000000000	0.0000000000000000e+00 0.0000000000000000e+00 4.4408920985006262e-16 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00	1.0000000000000020 2.0000000000000022 3.0000000000000027 4.0000000000000027 5.0000000000000018 6.0000000000000027 7.0000000000000044 8.0000000000000036 9.0000000000000036 10.0000000000000036	1.9984014443252818e-15 2.2204460492503131e-15 2.6645352591003757e-15 2.6645352591003757e-15 1.7763568394002505e-15 2.6645352591003757e-15 4.4408920985006262e-15 3.5527136788005009e-15 3.5527136788005009e-15 3.5527136788005009e-15
1	1.00000000000000788 2.00000000000000799 3.00000000000000804 4.00000000000000799 5.00000000000000790 6.00000000000000782 7.00000000000000790 8.00000000000000799 9.00000000000000782 10.00000000000000799	7.8825834748386114e-14 7.9936057773011271e-14 8.0380146982861334e-14 7.9936057773011271e-14 7.9047879353311146e-14 7.8159700933611020e-14 7.9047879353311146e-14 7.9936057773011271e-14 7.8159700933611020e-14 7.9936057773011271e-14	1.00000000000002358 2.00000000000002394 3.00000000000002389 4.00000000000002398 5.00000000000002398 6.00000000000002416 7.00000000000002416 8.00000000000002416 9.00000000000002434 10.00000000000002434	2.3581137043038325e-13 2.3936408410918375e-13 2.3891999489933369e-13 2.3980817331903381e-13 2.3980817331903381e-13 2.4158453015843406e-13 2.4158453015843406e-13 2.4158453015843406e-13 2.4336088699783431e-13 2.4336088699783431e-13
2	0.9999999999987548 1.9999999999987528 2.9999999999987530 3.9999999999987530	-1.2452261444195756e-12 -1.2472245458639009e-12 -1.2470025012589758e-12 -1.2470025012589758e-12	1.00000000000003568 2.00000000000003575 3.00000000000003570 4.00000000000003570	3.5682568011452531e-13 3.5749181392930041e-13 3.5704772471945034e-13 3.5704772471945034e-13

	4.999999999987530 5.999999999987512 6.999999999987530 7.999999999987521 8.999999999987512 9.999999999987512	-1.2470025012589758e-12 -1.2487788580983761e-12 -1.2470025012589758e-12 -1.2478906796786760e-12 -1.2487788580983761e-12 -1.2487788580983761e-12	5.0000000000003579 6.0000000000003553 7.0000000000003535 8.0000000000003553 9.0000000000003553 10.0000000000003553	3.5793590313915047e-13 3.5527136788005009e-13 3.5349501104064984e-13 3.5527136788005009e-13 3.5527136788005009e-13 3.5527136788005009e-13
3	1.0000000000177691 2.0000000000177720 3.0000000000177716 4.0000000000177725 5.0000000000177725 6.0000000000177725 7.0000000000177725 8.0000000000177725 9.0000000000177742 10.0000000000177725	1.7769119509125630e-11 1.7772006088989656e-11 1.7771561999779806e-11 1.7772450178199506e-11 1.7772450178199506e-11 1.7772450178199506e-11 1.7772450178199506e-11 1.7772450178199506e-11 1.7774226535038906e-11 1.7772450178199506e-11	0.999999999804591 1.999999999804554 2.999999999804561 3.999999999804547 4.999999999804547 5.999999999804530 6.999999999804521 7.999999999804521 8.999999999804530 9.999999999804530	-1.9540924434124918e-11 -1.9544588170106181e-11 -1.9543922036291406e-11 -1.9545254303920956e-11 -1.9545254303920956e-11 -1.9547030660760356e-11 -1.9547918839180056e-11 -1.9547918839180056e-11 -1.9547030660760356e-11 -1.9547030660760356e-11
4	1.0000000000621738 2.0000000000621756 3.0000000000621756 4.0000000000621760 5.0000000000621752 6.0000000000621752 7.0000000000621734 8.0000000000621760 9.0000000000621725 10.0000000000621760	6.2173821646638316e-11 6.2175598003477171e-11 6.2175598003477171e-11 6.2176042092687567e-11 6.2175153914267867e-11 6.2175153914267867e-11 6.2173377557428466e-11 6.2176042092687567e-11 6.2172489379008766e-11 6.2176042092687567e-11	1.0000000000000004 2.0000000000000004 3.0000000000000009 4.0000000000000009 5.0000000000000000 5.9999999999999991 7.0000000000000009 8.0000000000000018 9.0000000000000000 10.0000000000000000	4.4408920985006262e-16 4.4408920985006262e-16 8.8817841970012523e-16 8.8817841970012523e-16 0.0000000000000000e+00 -8.8817841970012523e-16 8.8817841970012523e-16 1.7763568394002505e-15 0.0000000000000000e+00 0.0000000000000000e+00
5	0.9999999978239588 1.9999999978239560 2.9999999978239558 3.9999999978239544 4.9999999978239549 5.9999999978239522 6.9999999978239531 7.9999999978239513 8.9999999978239522 9.9999999978239522	-2.1760412360904979e-09 -2.1760440116480595e-09 -2.1760442336926644e-09 -2.1760455659602940e-09 -2.1760451218710841e-09 -2.1760477864063432e-09 -2.1760468982279235e-09 -2.1760486745847629e-09 -2.1760477864063432e-09 -2.1760477864063432e-09	1.0000000003552727 2.0000000003552740 3.0000000003552740 4.0000000003552740 5.0000000003552731 6.0000000003552749 7.0000000003552740 8.0000000003552731 9.0000000003552731 10.0000000003552749	3.5527270014767964e-10 3.5527403241530919e-10 3.5527403241530919e-10 3.5527403241530919e-10 3.5527314423688949e-10 3.5527492059372889e-10 3.5527403241530919e-10 3.5527314423688949e-10 3.5527314423688949e-10 3.5527492059372889e-10
6	1.0000000168753920 2.0000000168753940 3.0000000168753944 4.0000000168753944 5.0000000168753944 6.0000000168753962 7.0000000168753944 8.0000000168753971 9.0000000168753971 10.0000000168753989	1.6875391972703824e-08 1.6875393971105268e-08 1.6875394415194478e-08 1.6875394415194478e-08 1.6875394415194478e-08 1.6875396191551317e-08 1.6875394415194478e-08 1.6875397079729737e-08 1.6875397079729737e-08 1.687539856086576e-08	1.0000000017763626 2.0000000017763631 3.0000000017763635 4.0000000017763631 5.0000000017763631 6.0000000017763622 7.0000000017763613 8.0000000017763604 9.0000000017763622 10.0000000017763604	1.7763626125599785e-09 1.7763630566491884e-09 1.7763635007383982e-09 1.7763630566491884e-09 1.7763630566491884e-09 1.7763621684707687e-09 1.7763612802923490e-09 1.7763603921139293e-09 1.7763621684707687e-09 1.7763603921139293e-09
7	1.0000000310862491 2.0000000310862491 3.0000000310862500 4.0000000310862491 5.0000000310862500 6.0000000310862482 7.0000000310862482 8.0000000310862482 9.0000000310862482 10.0000000310862465	3.1086249130396482e-08 3.1086249130396482e-08 3.1086250018574901e-08 3.1086249130396482e-08 3.1086250018574901e-08 3.1086248242218062e-08 3.1086248242218062e-08 3.1086248242218062e-08 3.1086248242218062e-08 3.1086246465861223e-08	1.0000000355271372 2.0000000355271386 3.0000000355271386 4.0000000355271386 5.0000000355271377 6.0000000355271368 7.0000000355271359 8.0000000355271368 9.0000000355271386 10.0000000355271386	3.5527137232094219e-08 3.5527138564361849e-08 3.5527138564361849e-08 3.5527138564361849e-08 3.5527137676183429e-08 3.5527136788005009e-08 3.5527135899826590e-08 3.5527136788005009e-08 3.5527138564361849e-08 3.5527138564361849e-08
8	0.9999970246036053 1.9999970246036005 2.9999970246036005 3.9999970246036014 4.9999970246036005	-2.9753963947110051e-06 -2.9753963994849641e-06 -2.9753963994849641e-06 -2.9753963985967857e-06 -2.9753963994849641e-06	0.9999969801949719 1.9999969801949664 2.9999969801949669 3.9999969801949660 4.9999969801949664	-3.0198050281482480e-06 -3.0198050335883408e-06 -3.0198050331442516e-06 -3.0198050340324301e-06 -3.0198050335883408e-06

	5.9999970246035987 6.9999970246035970 7.9999970246035979 8.9999970246035979 9.9999970246035979	-2.9753964012613210e-06 -2.9753964030376778e-06 -2.9753964021494994e-06 -2.9753964021494994e-06 -2.9753964021494994e-06	5.9999969801949637 6.9999969801949637 7.9999969801949637 8.9999969801949646 9.9999969801949646	-3.0198050362528761e-06 -3.0198050362528761e-06 -3.0198050362528761e-06 -3.0198050353646977e-06 -3.0198050353646977e-06
9	1.0000173195238968 2.0000173195238995 3.0000173195238995 4.0000173195239013 5.0000173195238995 6.0000173195239022 7.0000173195239013 8.0000173195239004 9.0000173195239022 10.0000173195239004	1.7319523896830447e-05 1.7319523899494982e-05 1.7319523899494982e-05 1.7319523901271339e-05 1.7319523899494982e-05 1.7319523902159517e-05 1.7319523901271339e-05 1.7319523900383160e-05 1.7319523902159517e-05 1.7319523900383160e-05	0.9999715783939044 1.9999715783939001 2.9999715783938998 3.9999715783938985 4.9999715783938994 5.9999715783938967 6.9999715783938949 7.9999715783938958 8.9999715783938967 9.9999715783938949	-2.8421606095618834e-05 -2.8421606099948704e-05 -2.8421606100170749e-05 -2.8421606101503016e-05 -2.8421606100614838e-05 -2.8421606103279373e-05 -2.8421606105055730e-05 -2.8421606104167552e-05 -2.8421606103279373e-05 -2.8421606105055730e-05
10	0.9995648158016371 1.9995648158016299 2.9995648158016306 3.9995648158016284 4.9995648158016293 5.9995648158016248 6.9995648158016248 7.9995648158016257 8.9995648158016266 9.9995648158016266	-4.3518419836285904e-04 -4.3518419837007549e-04 -4.3518419836940936e-04 -4.3518419837162980e-04 -4.3518419837074163e-04 -4.3518419837518252e-04 -4.3518419837518252e-04 -4.3518419837429434e-04 -4.3518419837340616e-04 -4.3518419837340616e-04	0.9998401335772825 1.9998401335772795 2.9998401335772793 3.9998401335772789 4.9998401335772797 5.9998401335772797 6.9998401335772789 7.9998401335772780 8.9998401335772780 9.9998401335772780	-1.5986642271748064e-04 -1.5986642272047824e-04 -1.5986642272070029e-04 -1.5986642272114437e-04 -1.5986642272025620e-04 -1.5986642272025620e-04 -1.5986642272114437e-04 -1.5986642272203255e-04 -1.5986642272203255e-04 -1.5986642272203255e-04
11	1.0007550521874293 2.0007550521874298 3.0007550521874302 4.0007550521874302 5.0007550521874302 6.0007550521874311 7.0007550521874311 8.0007550521874329 9.0007550521874311 10.0007550521874311	7.5505218742932811e-04 7.5505218742977220e-04 7.5505218743021629e-04 7.5505218743021629e-04 7.5505218743021629e-04 7.5505218743110447e-04 7.5505218743110447e-04 7.5505218743288083e-04 7.5505218743110447e-04 7.5505218743110447e-04	1.0001776514478631 2.0001776514478631 3.0001776514478635 4.0001776514478626 5.0001776514478635 6.0001776514478617 7.0001776514478635 8.0001776514478635 9.0001776514478635 10.0001776514478617	1.7765144786308085e-04 1.7765144786308085e-04 1.7765144786352494e-04 1.7765144786263676e-04 1.7765144786352494e-04 1.7765144786174858e-04 1.7765144786352494e-04 1.7765144786352494e-04 1.7765144786352494e-04 1.7765144786174858e-04
12	1.0066637050199885 2.0066637050199891 3.0066637050199887 4.0066637050199887 5.0066637050199896 6.0066637050199905 7.0066637050199896 8.0066637050199905 9.0066637050199905 10.0066637050199905	6.6637050199884751e-03 6.6637050199891412e-03 6.6637050199886971e-03 6.6637050199886971e-03 6.6637050199895853e-03 6.6637050199904735e-03 6.6637050199895853e-03 6.6637050199904735e-03 6.6637050199904735e-03 6.6637050199904735e-03	1.0000000000000013 2.0000000000000009 3.0000000000000009 4.0000000000000009 5.0000000000000009 5.9999999999999982 6.9999999999999982 7.9999999999999991 9.0000000000000000 10.0000000000000000	1.3322676295501878e-15 8.8817841970012523e-16 8.8817841970012523e-16 8.8817841970012523e-16 8.8817841970012523e-16 -1.7763568394002505e-15 -1.7763568394002505e-15 -8.8817841970012523e-16 0.0000000000000000e+00 0.0000000000000000e+00
13	0.9162995594713699 1.9162995594713685 2.9162995594713692 3.9162995594713688 4.9162995594713683 5.9162995594713674 6.9162995594713692 7.9162995594713683 8.9162995594713674 9.9162995594713657	-8.3700440528630127e-02 -8.3700440528631459e-02 -8.3700440528630793e-02 -8.3700440528631237e-02 -8.3700440528631681e-02 -8.3700440528632569e-02 -8.3700440528630793e-02 -8.3700440528631681e-02 -8.3700440528632569e-02 -8.3700440528634346e-02	1.2429906542056020 2.2429906542056064 3.2429906542056068 4.2429906542056068 5.2429906542056059 6.2429906542056051 7.2429906542056068 8.2429906542056059 9.2429906542056059 10.2429906542056059	2.4299065420560195e-01 2.4299065420560639e-01 2.4299065420560684e-01 2.4299065420560684e-01 2.4299065420560595e-01 2.4299065420560506e-01 2.4299065420560684e-01 2.4299065420560595e-01 2.4299065420560595e-01 2.4299065420560595e-01

Вывод:

Метод Гаусса не отличается по точности решения от метода решения СЛАУ через  $LL^T$  разложение.