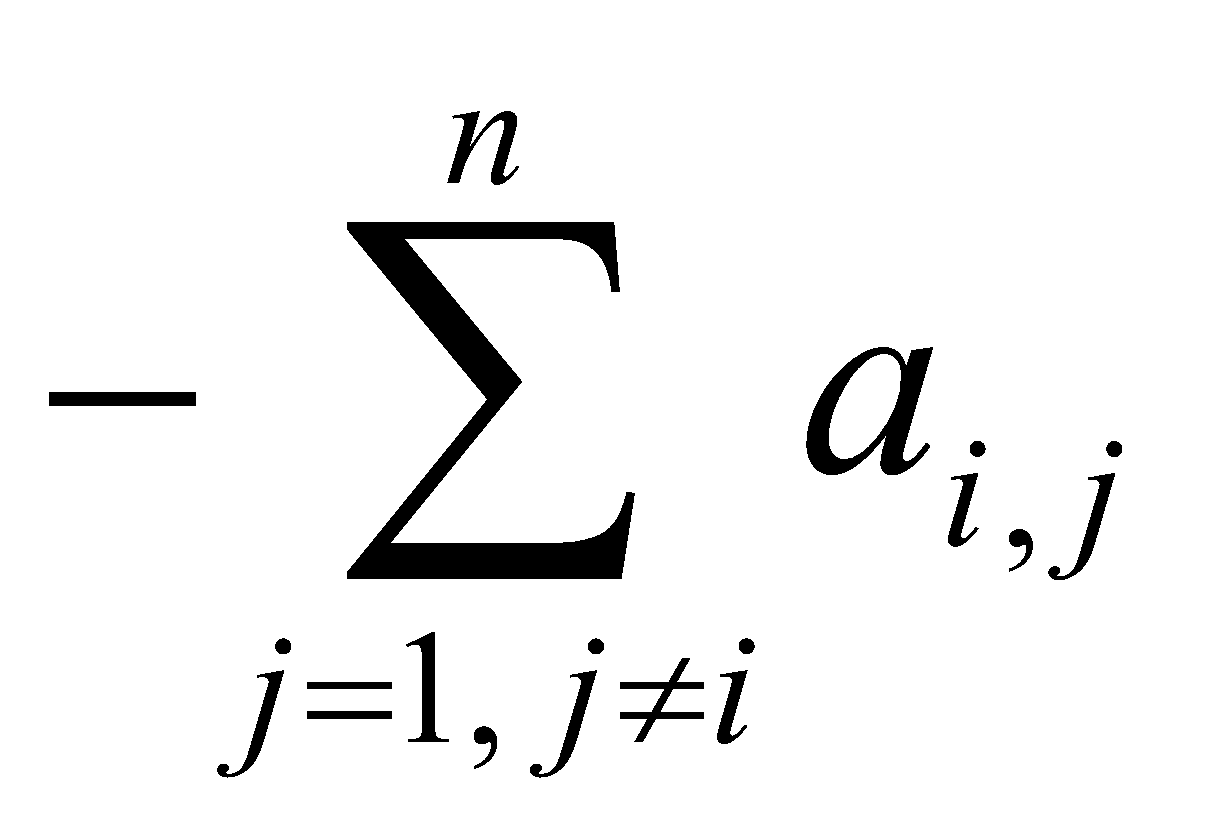
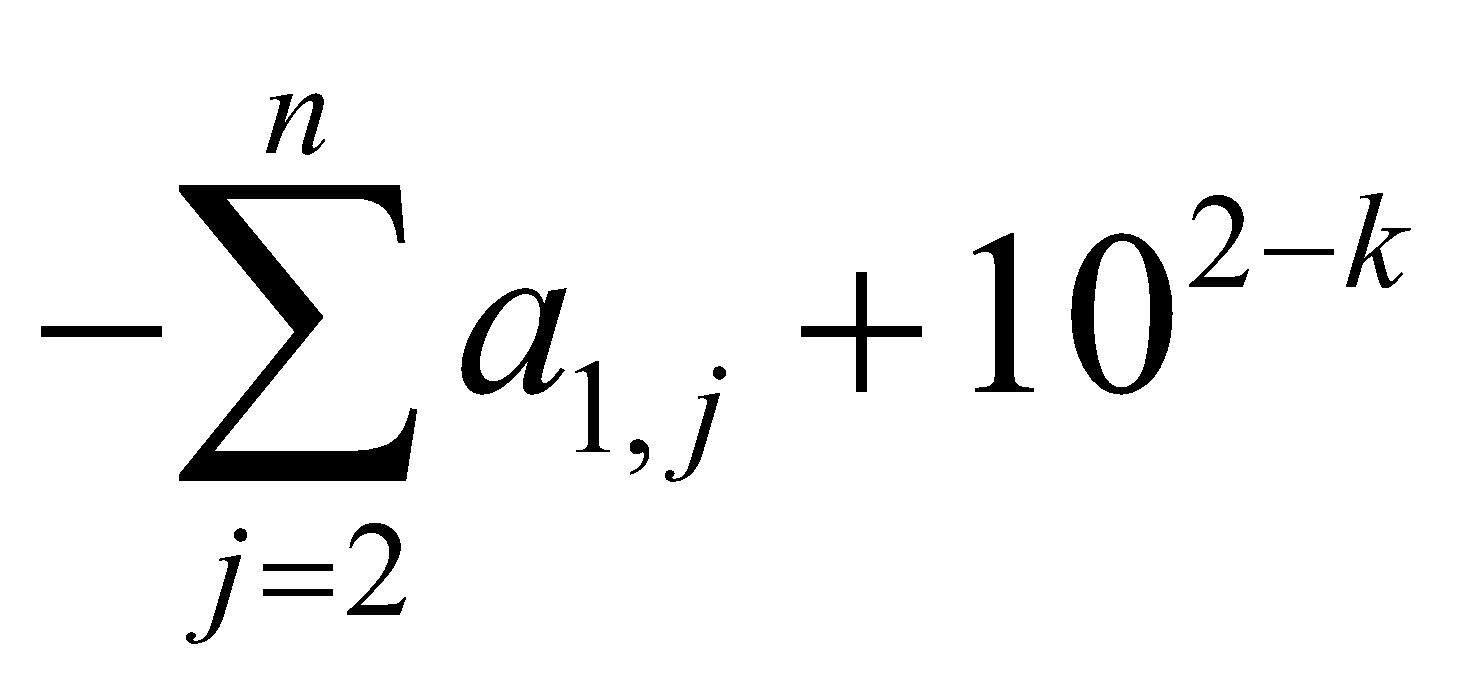
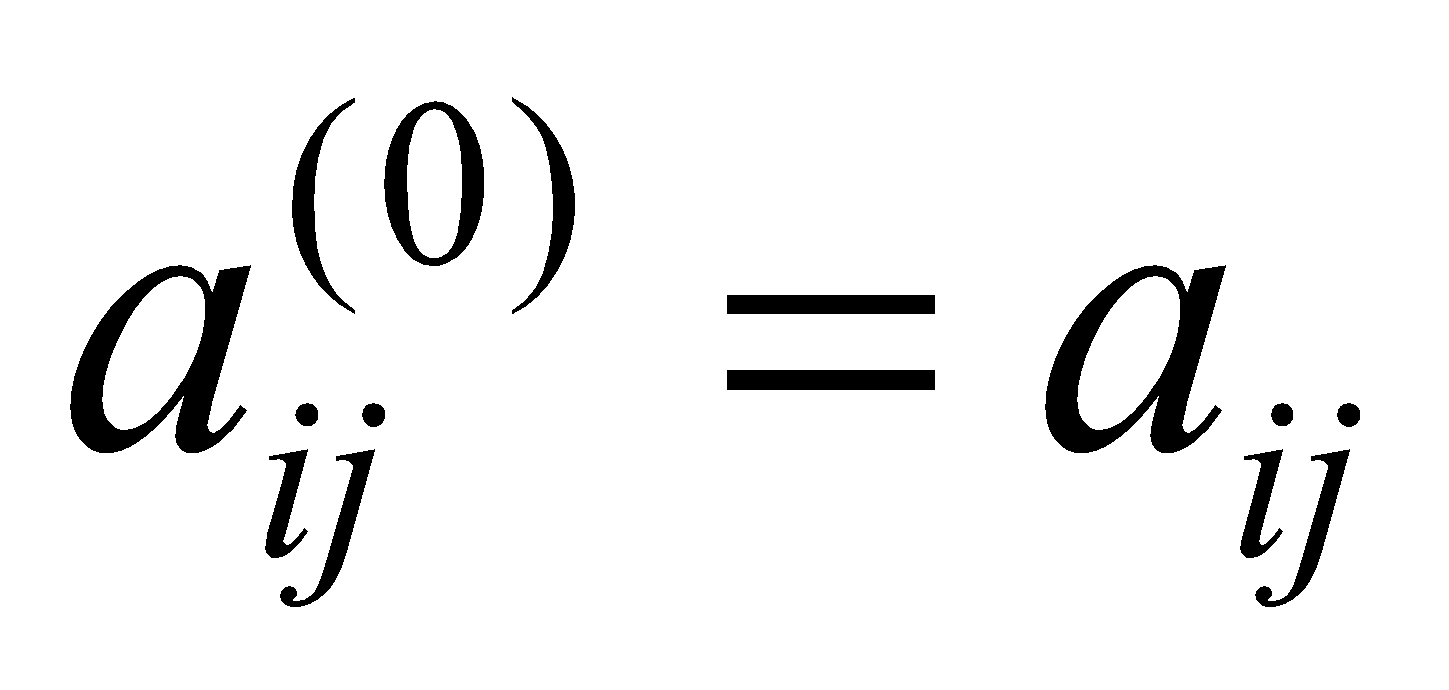
**Лабораторная работа №2**

Разработать программу численного решения СЛАУ на основе LDLT-разложения. Сравнить время реализации рассматриваемого примера с временем реализации аналогичного примера в работе 1 «Метод Гаусса». Матрицу (порядка *n*) системы сформировать следующим образом:

* недиагональные элементы *ai,j*, *i<j*, выбираются случайным образом из диапазона от 0 до *–*1000; если *i>j*, то полагается *ai,j*=*aj,i*.
* *ai,i=**, 2≤i*≤*n*; *a*11*=**.*

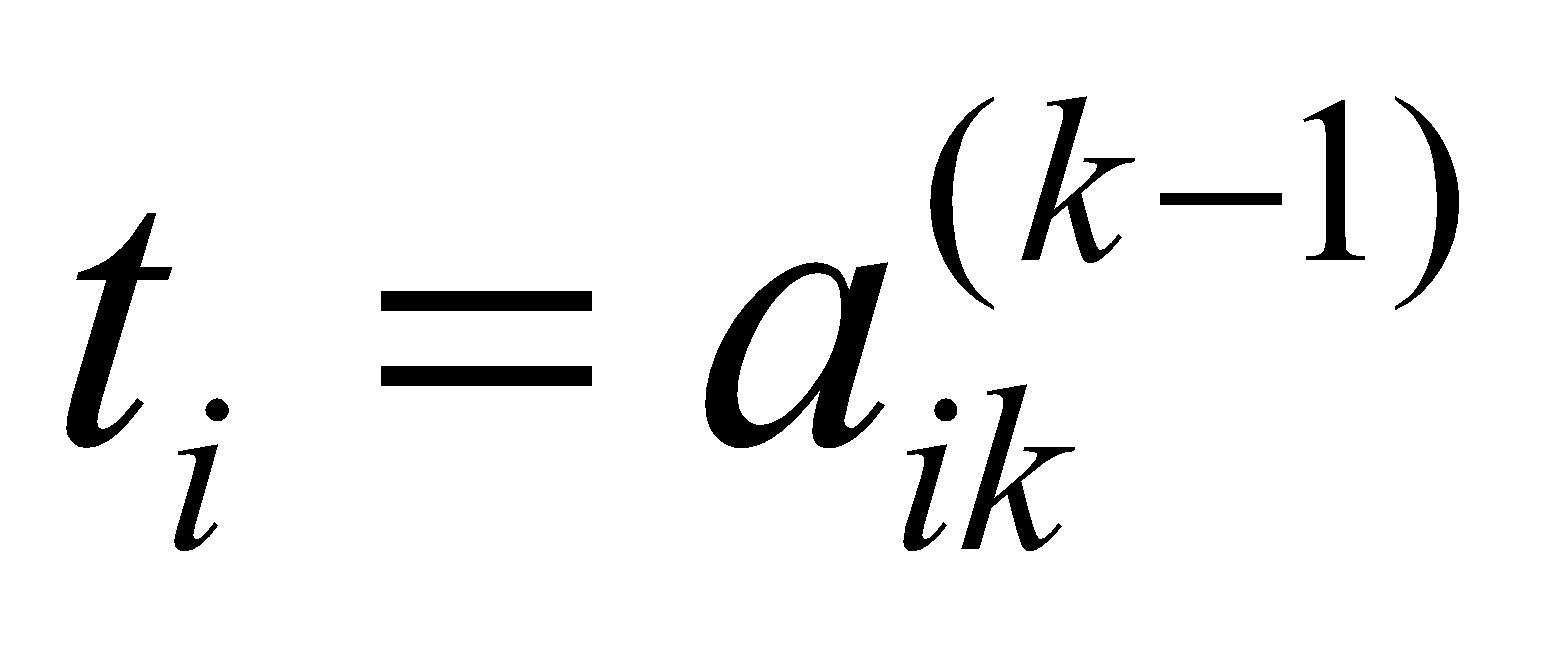
Относительная погрешность , где – точное решение (*m*, *m*+1, ..., *n*+*m*–1).

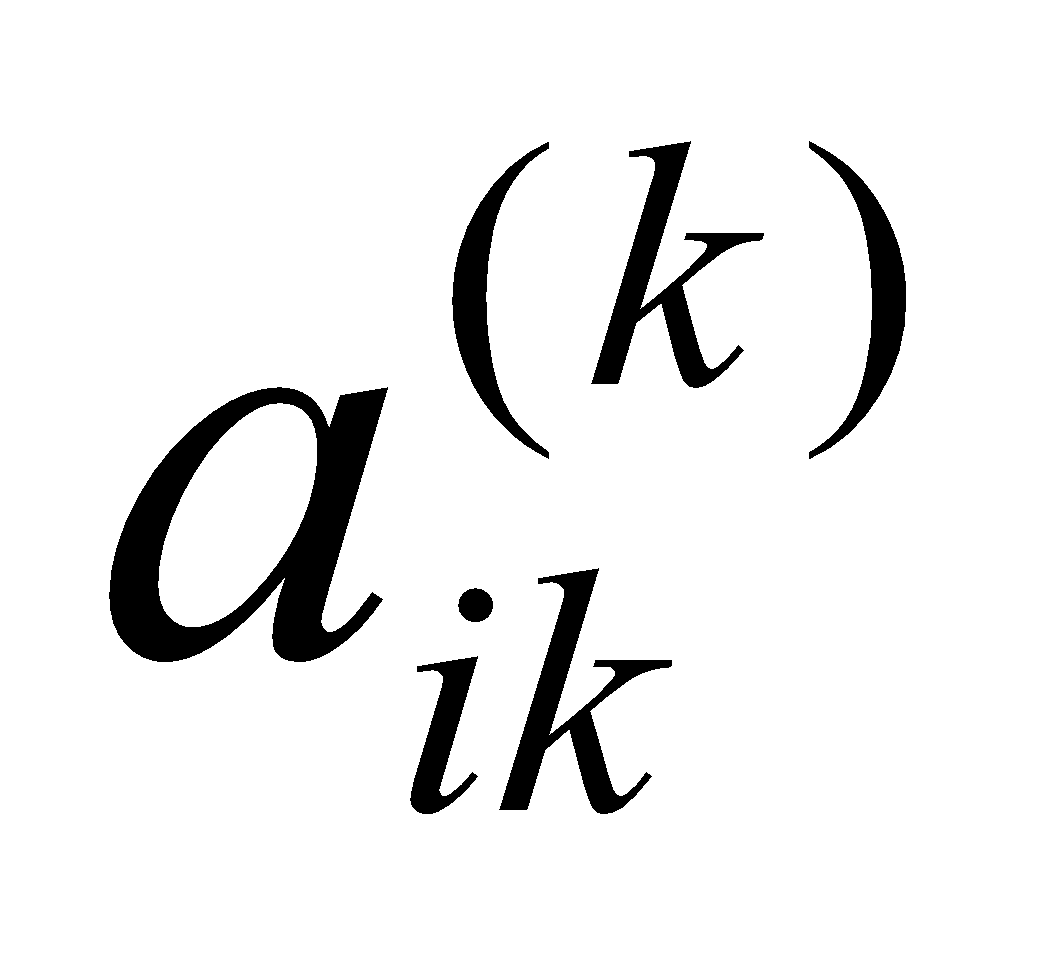
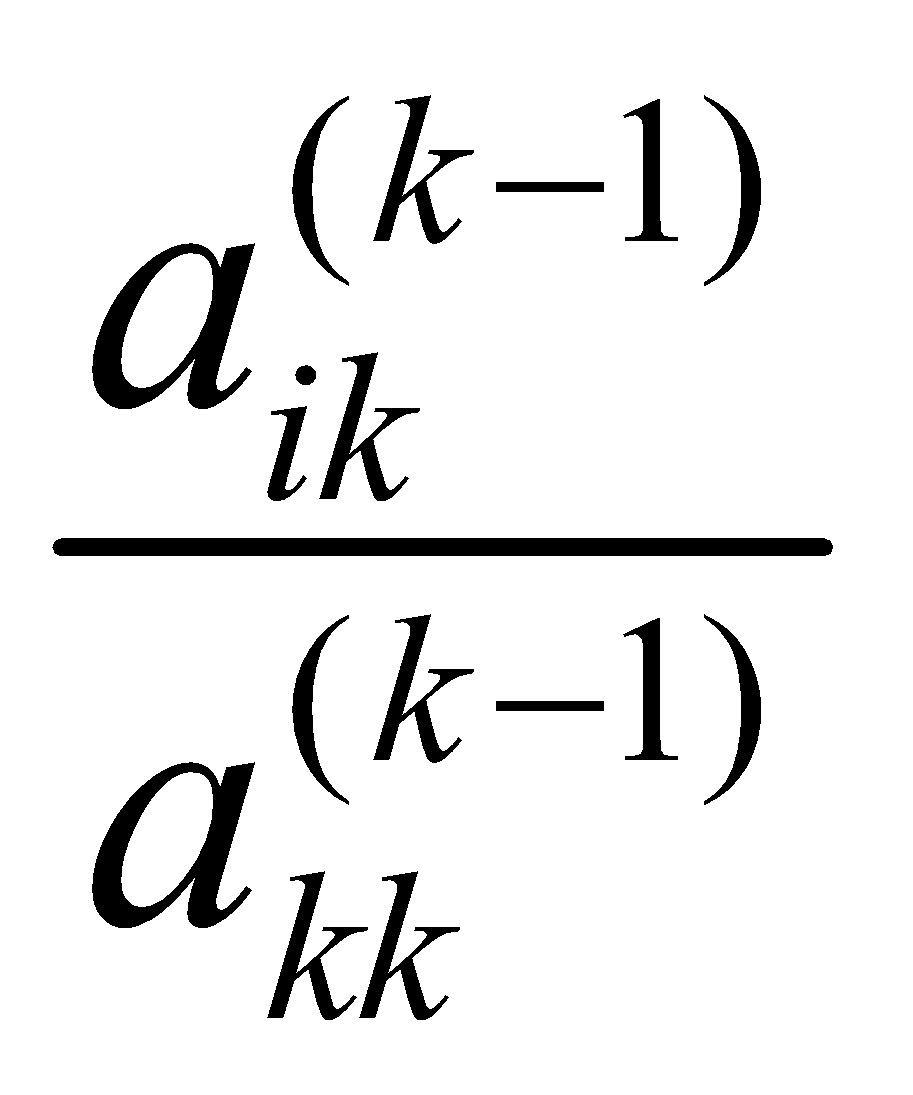
Разложение происходит по алгоритму:

*, i =*1*,*2*,… ,n, j =*1*,*2*,…,i*;

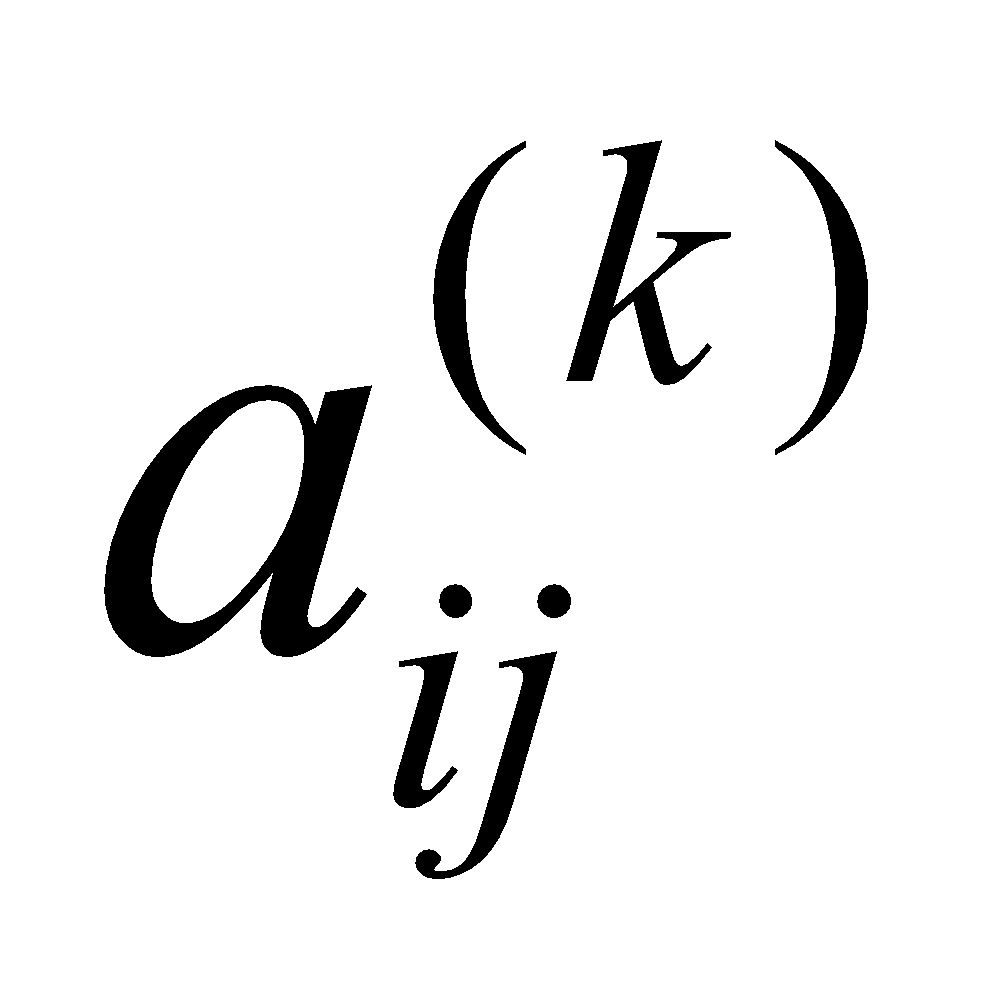
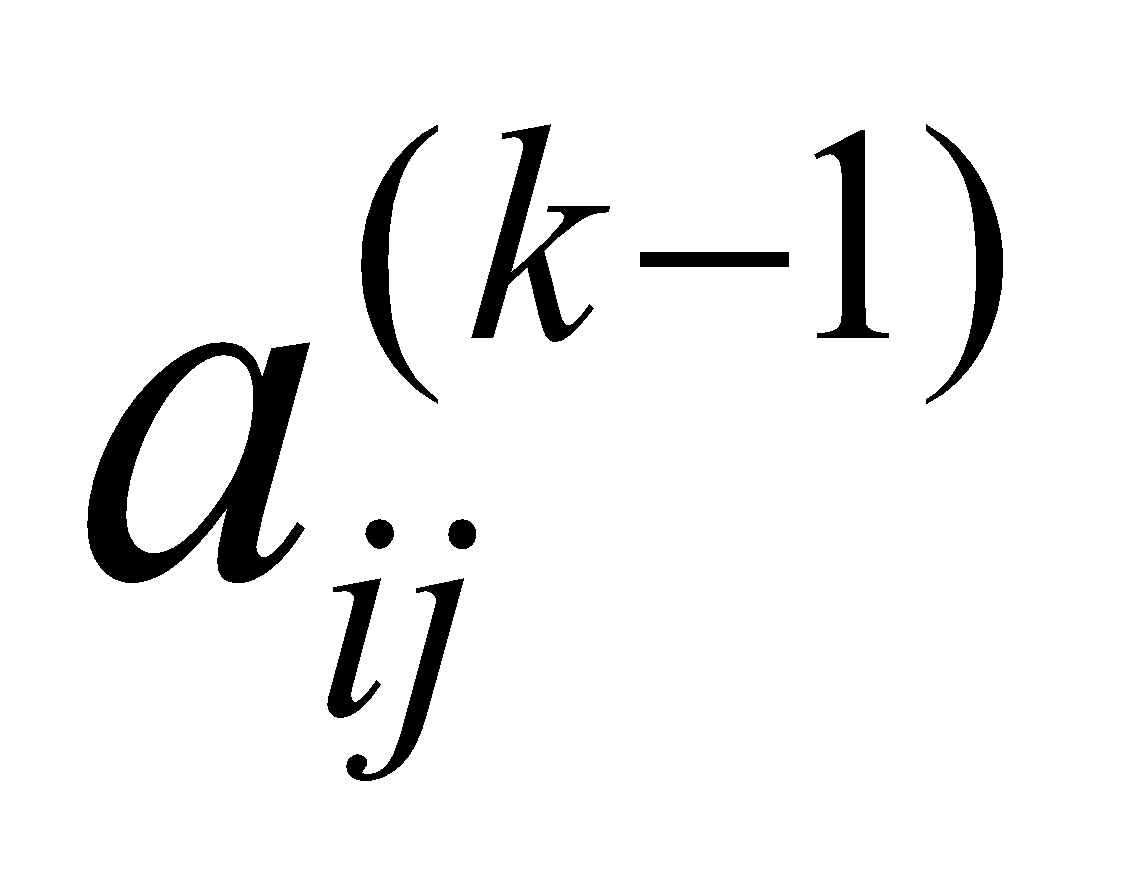
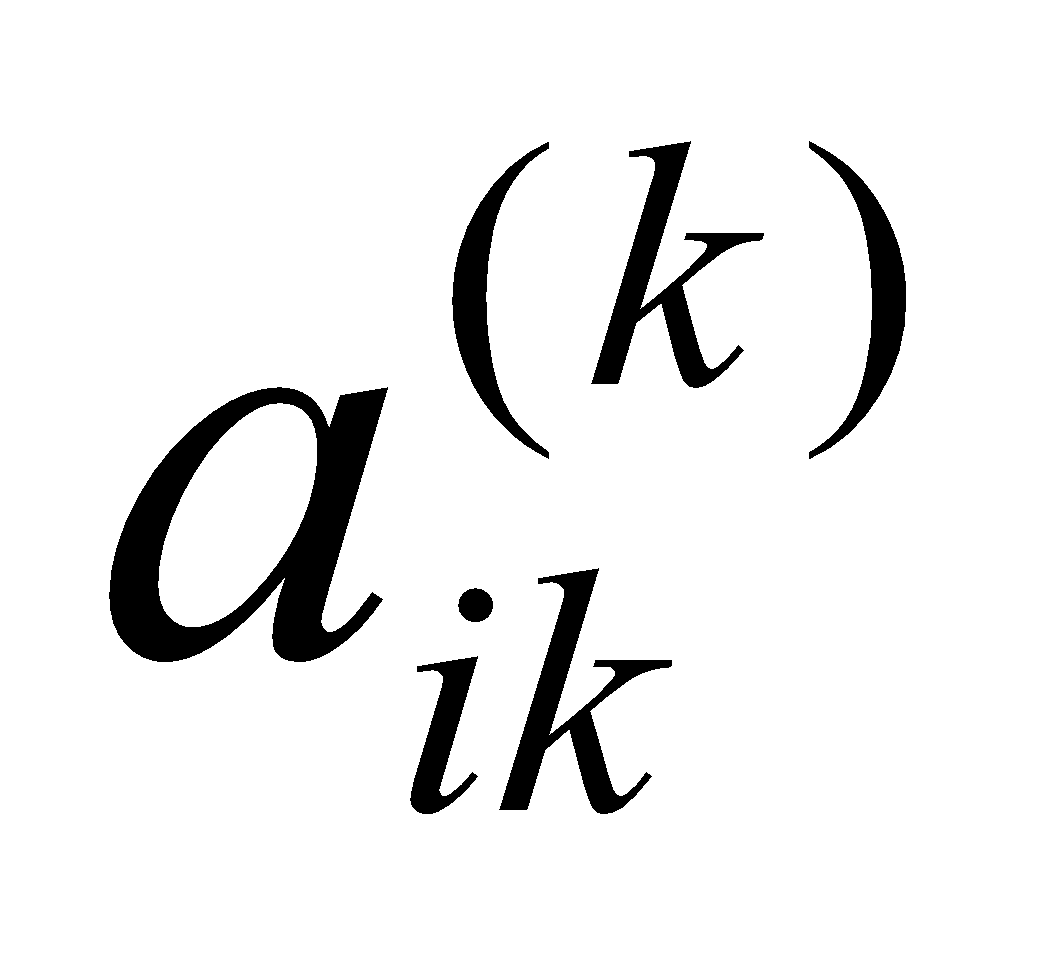
*k =* 1*,* 2*, … , n–*1:

*i = k+*1*, k+*2*, … , n*:

,

=,

*j = k+*1*, … , i*:

=–t*j.*

Решение уравнения состоит из последовательного решения системы уравнений:

Ly = b,

Dz = y,

LT x = z.

**Входные данные.**

n = 1700, m = 14, k = 4

**Листинг программы.**

**#include <iostream>**

#include <vector>

#include <stdlib.h>

#include <ctime>

#include <cmath>

#include <chrono>

#include <string>

int size = 1700;

int m = 14;

int k = 2;

std::vector<double> generateVectorB(size\_t size) {

    std::vector<double>  ans(size, 0);

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

        ans[i] = m + i;

    }

    return ans;

}

std::vector<double> multiply(std::vector<std::vector<double>> A, std::vector<double> x) {

    std::vector<double> ans(x.size(), 0);

    {

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

            double sum = 0;

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

                sum += A[i][j] \* x[j];

            }

            ans[i] = sum;

        }

    }

    return ans;

}

std::vector<std::vector<double>> generateMatrix(int size) {

    std::vector<std::vector<double>> matrix(size, std::vector<double>(size, 0));

    for (int i = 0; i < size; i++)

    {

        double diagElement = 0;

        for (int j = 0; j < size; j++)

        {

            if (i != j && matrix[i][j] == 0){

                matrix[i][j] = (double)rand() / RAND\_MAX \* -1000.0f;

                matrix[j][i] = matrix[i][j];

                diagElement -= matrix[i][j];

            }

        }

        if (i == 0)

        {

            matrix[i][i] = diagElement + std::pow(10, 2 - k);

        }

        else{

            matrix[i][i] = diagElement;

        }

    }

    return matrix;

}

std::vector<double> SolveLyEqB(const std::vector<std::vector<double>> &lMatrix, const std::vector<double> &bVector)

{

    std::vector<double> y(size);

    for (int i = 0; i < size; i++)

    {

        double sum = 0.0;

        for (int j = 0; j < i; j++)

        {

            sum += lMatrix[i][j] \* y[j];

        }

        y[i] = bVector[i] - sum;

    }

    return y;

}

std::vector<double> SolveDzEqY(const std::vector<std::vector<double>> &DMatrix, const std::vector<double> &yVector)

{

    std::vector<double> z(size);

    for (int i = 0; i < size; i++)

    {

        z[i] = yVector[i] / DMatrix[i][i];

    }

    return z;

}

std::vector<double> SolveLTxEqZ(const std::vector<std::vector<double>> &ltMatrix, const std::vector<double> &zVector)

{

    std::vector<double> x(size);

    for (int i = size - 1; i >= 0; i--)

    {

        double sum = 0.0;

        for (int j = i + 1; j < size; j++)

        {

            sum += ltMatrix[j][i] \* x[j];

        }

        x[i] = zVector[i] - sum;

    }

    return x;

}

std::vector<double> SolveSystem(std::vector<std::vector<double>> &matrix, const std::vector<double> &vector)

{

    std::vector<double> t(size);

    for (int k = 0; k < size - 1; ++k)

    {

        for (int i = k + 1; i < size; ++i)

        {

            t[i] = matrix[i][k];

            matrix[i][k] /= matrix[k][k];

            for (int j = k + 1; j <= i; ++j)

            {

                matrix[i][j] -= matrix[i][k] \* t[j];

            }

        }

    }

    const std::vector<double> y = SolveLyEqB(matrix, vector);

    const std::vector<double> z = SolveDzEqY(matrix, y);

    const std::vector<double> x = SolveLTxEqZ(matrix, z);

    return x;

}

std::vector<double> gaussWithoutMainElem(std::vector<std::vector<double>> matrix, std::vector<double> b) {

    std::vector<std::vector<double>> matrix\_copy(b.size(), std::vector<double>(b.size(), 0));

    std::vector<double> b\_copy(b.size(), 0);

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

        for(int j = 0; j < b.size(); ++j) {

            matrix\_copy[i][j] = matrix[i][j];

        }

        b\_copy[i] = b[i];

    }

    for(int k = 0; k < b.size() - 1; ++k) {

        for(int i = k + 1; i < b.size(); ++i) {

            double l = matrix\_copy[i][k] / matrix\_copy[k][k];

            matrix\_copy[i][k] = 0.0f;

            for(int j = k + 1; j < b.size(); ++j) {

                matrix\_copy[i][j] -= l \* matrix\_copy[k][j];

            }

            b\_copy[i] -= l \* b\_copy[k];

        }

    }

    std::vector<double> x(b.size(), 0);

    x[b.size() - 1] = b\_copy[b.size() - 1] / matrix\_copy[b.size() - 1][b.size() - 1];

    for(int i = b.size() - 2; i >= 0; --i) {

        double sum = 0.0f;

        for(int j = i + 1; j < b.size(); ++j) {

            sum += matrix\_copy[i][j] \* x[j];

        }

        x[i] = (b\_copy[i] - sum) / matrix\_copy[i][i];

    }

    return x;

}

void print5(std::vector<double> vector) {

    for (int i = 0; i < 5; i++)

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

    std::cout << "\n";

}

double relativeError(std::vector<double> x\_, std::vector<double> x) {

    double max\_1 = -1;

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

        max\_1 = std::max(std::abs(x\_[i] - x[i]), max\_1);

    }

    double max\_2 = -1;

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

        max\_2 = std::max(std::abs(x[i]), max\_2);

    }

    return max\_1 / max\_2;

}

int main()

{

    std::vector<std::vector<double>> matrix = generateMatrix(size);

    std::vector<double> x = generateVectorB(size);

    std::vector<double> b = multiply(matrix, x);

    auto start = std::chrono::steady\_clock::now();

    std::vector<double> x\_ = gaussWithoutMainElem(matrix, b);

    auto end = std::chrono::steady\_clock::now();

    const int time1 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

    start = std::chrono::steady\_clock::now();

    const std::vector<double> y = SolveSystem(matrix, b);

    end = std::chrono::steady\_clock::now();

    const int time2 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

    std::cout << "Without choosing element" << "\n";

    std::cout << "5 cordinats of solution ";

    print5(x\_);

    std::cout << "otnositelnaya pogreshnost " << relativeError(x\_, x) << "\n";

    std::cout << "Runtime " << time1 << " ms\n";

    std::cout << "LDL\n";

    std::cout << "5 cordinats of solution ";

    print5(y);

    std::cout << "Otnositelnaya pogreshnost " << relativeError(y, x) << "\n";

    std::cout << "Runtime " << time2 << " ms\n";

    return 0;

}

**Выходные данные.**

Gauss:

5 cordinats of solution 14 15 16 17 18

otnositelnaya pogreshnost 1.75252e-010

Runtime 11816 ms

LDL

5 cordinats of solution 14 15 16 17 18

Otnositelnaya pogreshnost 1.60678e-010

Runtime 6948 ms

**Выводы.**

Применение LDLT-разложения для работы с симметричными матрицами экономит время почти в два раза.