1. **Постановка Задачи.**

Разработать программу численного решения СЛАУ на основе LDLT-разложения.

Матрицу системы сформировать следующим образом:

* недиагональные элементы *ai,j*, *i<j*, выбираются из чисел 0, –1, –2, –3, *–*4 произвольным образом; если *i>j*, то полагается *ai,j*=*aj,i*.
* *ai,i=*, 2≤*i*≤*n*;
* *a*11*=*, *k*≥0.

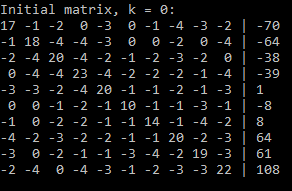
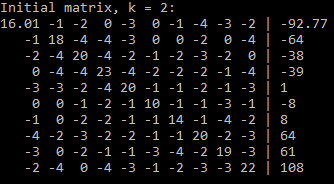
Правую часть *b* задать умножением матрицы *A* на вектор *x=*(*m*, *m*+1, ... , *n*+*m*–1): *b=Ax*.

Для вычислений выбрать параметры:

* *m* – номер в списке студенческой группы;
* *n* – одно из чисел в пределах от 10 до 12;
* *k* – рассмотреть два случая: *k=*0, *k=*(номер студенческой группы); элементы *ai,j* при фиксированных *i* и *j* в обоих случаях одни и те же (матрицы отличаются только элементом *a*11).

Программно реализовать (в качестве языка программирования выбрать C или C++) вычисления для рассматриваемого примера. Использовать алгоритм (5) файла «LDLt\_RtR разложения», требующий хранения только нижнего треугольника матрицы. В процессе факторизации матрицы *A* (*A=LDLT*) нижняя треугольная матрица *L* (за исключением единиц на главной диагонали) хранится на месте нижнего треугольника матрицы *A,* диагональная матрица *D* хранится на месте главной диагонали матрицы *A*.

**2. Входные данные**

**3.Листинг программы**

Файл SoLE.h:

#pragma once

#include <vector>

#include <iostream>

class SoLE

{

public:

SoLE();

SoLE(const SoLE&);

SoLE(int);

~SoLE();

void solve();

friend std::ostream& operator<<(std::ostream&, const SoLE&);

private:

std::vector<std::vector<float>> A;

std::vector<float> b;

std::vector<float> X;

int k;

static const int m = 23; // Номер в группе

static const int n = 10; // Одно из чисел в пределах от 12 до 15

float error(std::vector<float>);

void printLD();

};

Файл SoLE.cpp:

#include "SoLE.h"

#include <algorithm>

#include <ctime>

#include <iomanip>

#define A(i,j) (i < j ? A[j][i] : A[i][j])

SoLE::SoLE()

{

}

SoLE::SoLE(const SoLE & S) //copy and change A[0][0]

{

A = S.A;

b = S.b;

X = S.X;

k = 2;

A[0][0] -= 0.99;

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

{

b[i] = 0;

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

b[i] += A(i, j) \* X[j];

}

}

SoLE::SoLE(int \_k) : k(\_k) // k = 0, k = номер студенческой группы

{

//srand(time(NULL));

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

{

A.emplace\_back(i + 1);

for (int j = 0; j < i + 1; ++j)

{

if (i != j)

A[i][j] = -rand() % 5;

}

}

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

{

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

if (i != j)

A[i][i] -= A(i, j);

}

if (k == 0)

A[0][0] += 1; // 10^-k, k = 0

else if (k == 2)

A[0][0] += 0.01; // 10^-k, k = 2

else

throw std::invalid\_argument("k != 0 && k != 2");

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

X.push\_back(m + i);

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

{

b.push\_back(0);

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

b[i] += A(i, j) \* X[j];

}

}

SoLE::~SoLE()

{

}

void SoLE::solve()

{

std::vector<float> x(n);

std::vector<float> t(n);

for (int k = 0; k < n - 1; ++k) // LDL-разложение +

{

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

{

t[i] = A[i][k];

A[i][k] /= A[k][k];

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

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

}

}

printLD();

std::cout << std::endl;

for (int i = 1; i < n; ++i) // solving Ly = b +

{

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

b[i] -= b[j] \* A[i][j];

}

// vector b contains y

for (int i = 0; i < n; ++i)// multiplying D and Lt

{

for (int j = i; j < n; ++j)

if (i != j)

A[j][i] \*= A[i][i];

}

// A contains (DLt)t

for (int i = n - 1; i >= 0; --i) // обратный ход +

{

x[i] += b[i];

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

{

x[i] -= A[j][i] \* x[j];

}

x[i] /= A[i][i];

}

std::cout << std::setprecision(6) << "x\* = (";

std::for\_each(x.begin(), x.end() - 1, [](float val) { std::cout << val << ", "; });

std::cout << x[n - 1] << ")\n"

<< "Relative error: " << std::defaultfloat << error(x) \* 100 << "%\n";

}

float SoLE::error(std::vector<float> x)

{

std::vector<float> tmp(X);

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

tmp[i] -= x[i];

float max1 = abs(tmp[0]), max2 = abs(X[0]);

for (int i = 1; i < n; ++i)

{

if (abs(tmp[i]) > max1)

max1 = abs(tmp[i]);

if (abs(X[i]) > max2)

max2 = abs(X[i]);

}

return max1 / max2;

}

void SoLE::printLD()

{

std::cout << "Bottom triangle of the transformed matrix A, k = " << k << ":\n" << std::setprecision(2) << std::fixed;

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

{

for (int j = 0; j < i + 1; ++j)

std::cout << std::setw(5) << A[i][j] << ' ';

std::cout << std::endl;

}

}

std::ostream & operator<<(std::ostream & os, const SoLE & S)

{

for (int i = 0; i < S.n; ++i)

{

for (int j = 0; j < S.n; ++j)

os << std::setw((S.k && !j ? 5 : 2)) << (i < j ? S.A[j][i] : S.A[i][j]) << ' ';

os << "| " << S.b[i] << std::endl;

}

return os;

}

Файл lab2.cpp:

#include <iostream>

#include "SoLE.h"

#include <algorithm>

#include <ctime>

using namespace std;

int main()

{

SoLE s0(0), s1(s0);

cout << "Initial matrix, k = 0:\n"<< s0 << "\n";

s0.solve();

cout << "\n\nInitial matrix, k = 2:\n" << std::defaultfloat << s1 << "\n";

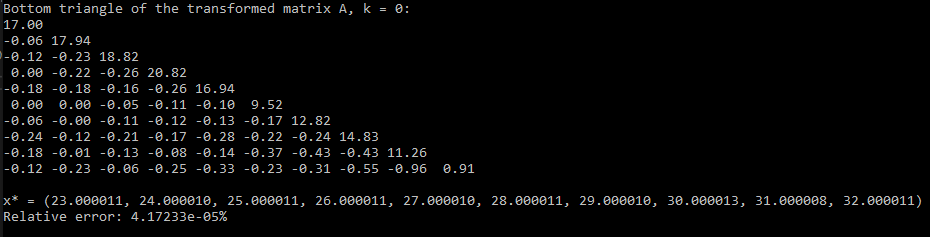
s1.solve();

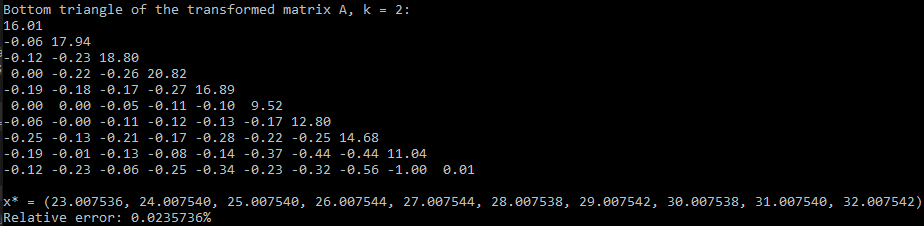
system("pause");

return 0;

}

**4. Выходные данные**





**5. Выводы**

Метод LDLT-разложения так же, как и метод Гаусса без выбора ведущего элемента, демонстрирует высокую точность результатов, если у исходной матрицы имеется диагональное преобладание. Но для данного метода необходимо в два раза меньше арифметических операций. Следовательно, он предпочтителен методу Гаусса для симметричных матриц.

Судя по выходным данным, чем сильнее диагональное преоблодание у матрицы, тем выше точность ответа.