|  |  |  |
| --- | --- | --- |
| ч | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое работа № 1 | | |
| по дисциплине «Численные методы» | | |
| **Прямые методы решения СЛАУ** | | |
|  | | |
|  |  |  |
| Группа ПМ-01 | будник светлана |
|  | самсонов семён |
| Вариант 8 |  |
|  |  |
| Преподаватель | Задорожный александр геннадьевич |
|  |  |
|  |  |
| Новосибирск, 2022 | | |

# Цели работы

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

# Анализ

**Входные данные:** симметричная, положительно определенная матрица , в формате: размер матрицы **size**, диагональные элементы **di**, внедиагональные элементы (нижний треугольник) **al**, индексы **ia**. И вектор правых частей **.**

**Выходные данные:** вектор неизвестных **.**

Пусть дана система линейных алгебраических уравнений:

-разложение является разложением матрицы A в произведение нижнетреугольной матрицы L и транспонированной матрицы .

Предположим, что нам удалось разложить матрицу:

Подставляем в систему и получаем:

тогда подставляя в, получим:

Таким образом, решение изначальной системы сводится к основным этапам:

1. Разбить матрицу А на элементы матриц ;
2. Решить систему с нижней треугольной матрицей (прямой ход);
3. Решить систему с верхней треугольной матрицей (обратный ход).

Общие формулы будут выглядеть следующим образом:

# Текст программы

## Программа для решения СЛАУ LL^T:

* Программа разбита на несколько файлов:
* Main.cpp:

// Выбор точности работы программы:

// \_\_ALL\_FLOAT\_\_ - все дробные числа и счётчики в float,

// \_\_SEMI\_FLOAT\_\_ - все дробные числа в float, счётчики в double,

//\_\_ALL\_DOUBLE\_\_ - все дробные числа и счётчики в double

#define \_\_ALL\_DOUBLE\_\_

#include "predefined\_types.h"

#include "matrixIO.h"

#include "profile\_matrix\_operations.h"

#include <fstream>

#include <iostream>

#include <vector>

using namespace std;

void DotsToCommas(const string& fileName) {

auto ifile = ifstream(fileName);

vector<string> content;

while (!ifile.eof()) {

content.push\_back("");

ifile >> content.back();

}

ifile.close();

auto ofile = ofstream(fileName);

for (auto elem : content) {

auto index = elem.find('.');

if (index != static\_cast<size\_t>(-1)) {

elem[index] = ',';

}

ofile << elem << endl;

}

ofile.close();

}

int main()

{

setlocale(LC\_ALL, "ru-RU.utf8");

int returnCode = 0;

#pragma region DataInput

size\_t matrixSize = GetSizetFromFile("./matrix\_size.txt");

real\_t\* matrixDiag = GetArrayFromFile<real\_t>("./matrix\_diag.txt", matrixSize);

size\_t\* matrixIA = GetArrayFromFile<size\_t>("./matrix\_IA.txt", matrixSize + 1);

// На случай, если введённая матрица индексов начинается с 1, а не с 0

if (matrixIA[0] == 1)

{

for (int i = 0; i < matrixSize + 1; i++) matrixIA[i]--;

}

size\_t alSize = matrixIA[matrixSize];

real\_t\* matrixAL = GetArrayFromFile<real\_t>("./matrix\_AL.txt", alSize);

real\_t\* vectorB = GetArrayFromFile<real\_t>("./vector\_b.txt", matrixSize);

#pragma endregion

try {

GetMatrixL(matrixSize, matrixDiag, matrixAL, matrixIA);

//PrintArray(matrixDiag, matrixSize, g\_coutPrecision);

//cout << endl;

//PrintArray(matrixAL, alSize, g\_coutPrecision);

//cout << endl;

GetVectorY(matrixSize, matrixDiag, matrixAL, matrixIA, vectorB);

//cout << "Полученный вектор y: ";

//PrintArray(vectorB, matrixSize, g\_coutPrecision);

//cout << endl;

GetVectorX(matrixSize, matrixDiag, matrixAL, matrixIA, vectorB);

cout << "Полученный вектор x: ";

PrintArray(vectorB, matrixSize, g\_coutPrecision);

cout << endl;

auto outputFilePath = g\_outputFileName;

auto outputFile = ofstream(outputFilePath);

PrintArray(vectorB, matrixSize, g\_coutPrecision, outputFile);

outputFile.close();

DotsToCommas(outputFilePath);

}

catch (const exception& e) {

cerr << e.what() << endl;

returnCode = -1;

}

delete[] matrixDiag;

delete[] matrixAL;

delete[] matrixIA;

delete[] vectorB;

return returnCode;

}

matrixIO.h:

#pragma once

#include <fstream>

#include <iostream>

/// <summary>

/// Функция получения числа типа size\_t из файла

/// </summary>

/// <param name="filePath"> - путь до файла </param>

/// <returns> число из файла </returns>

inline size\_t GetSizetFromFile(const char\* filePath)

{

auto fin = std::ifstream(filePath);

size\_t matrix\_size = 0;

fin >> matrix\_size;

fin.close();

return matrix\_size;

}

/// <summary>

/// Функция считывания массива данных с файла

/// </summary>

/// <typeparam name="T"> - тип данных, который необходимо считать </typeparam>

/// <param name="filePath"> - путь до файла</param>

/// <param name="count"> - число элементов, необходимое считать </param>

/// <returns> указатель на начало считанного массива </returns>

template<typename T>

T\* GetArrayFromFile(const char\* filePath, const size\_t count)

{

T\* arr = new T[count];

auto fin = std::ifstream(filePath);

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

{

fin >> arr[i];

}

fin.close();

return arr;

}

/// <summary>

/// Вывод массива в произвольный поток

/// </summary>

/// <typeparam name="T"> - тип данных для вывода</typeparam>

/// <param name="\_arr"> - указатель на начало массива </param>

/// <param name="\_size"> - длина массива </param>

/// <param name="\_out"> - поток вывода (cout по умолчанию)</param>

template <typename T>

void PrintArray(const T\* \_arr, const size\_t \_size, const int precision = 7, std::ostream& \_out = std::cout)

{

auto prevPrec = \_out.precision();

auto prevFlags = \_out.flags();

\_out.precision(precision);

\_out.setf(std::ios\_base::fixed);

//\_out << "[ ";

//for (int i = 0; i < \_size - 1; i++)

//{

// \_out << \_arr[i] << ", ";

//}

//\_out << \_arr[\_size - 1] << " ]";

for (int i = 0; i < \_size; i++) {

\_out << \_arr[i] << std::endl;

}

\_out.precision(prevPrec);

\_out.flags(prevFlags);

}

* predefined\_types.h:

#pragma once

#include <cmath>

#include <string>

#ifdef \_\_ALL\_FLOAT\_\_

using real\_t = float;

using accum\_t = float;

constexpr real\_t g\_eps = FLT\_EPSILON \* 10.0;

constexpr int g\_coutPrecision = 7;

const std::string g\_outputFileName = "allFloatOutput.txt";

#else

#ifdef \_\_SEMI\_FLOAT\_\_

using real\_t = float;

using accum\_t = double;

constexpr real\_t g\_eps = FLT\_EPSILON \* 10.0;

constexpr int g\_coutPrecision = 7;

const std::string g\_outputFileName = "scalarDoubleOutput.txt";

#else // \_\_ALL\_DOUBLE\_\_

using real\_t = double;

using accum\_t = double;

constexpr real\_t g\_eps = DBL\_EPSILON \* 10.0;

constexpr int g\_coutPrecision = 15;

const std::string g\_outputFileName = "allDoubleOutput.txt";

#endif

#endif

inline bool IsAlmostEq(real\_t \_first, real\_t \_second, real\_t maxRelDif = g\_eps) {

real\_t delta = abs(\_first - \_second);

\_first = abs(\_first); \_second = abs(\_second);

real\_t maxNum = \_first > \_second ? \_first : \_second;

return delta <= maxNum \* maxRelDif;

}

profile\_matrix\_operations.h:

#pragma once

#include "predefined\_types.h"

#include <iostream>

// Функция преобразует вектор y в вектор x (перезаписывает элементы!)

void GetVectorX(size\_t \_size, real\_t\* \_matrixDiag, real\_t\* \_matrixL, size\_t\* \_matrixIA, real\_t\* \_vecB) {

// Сначала отдельно считаем последний элемент

\_vecB[\_size - 1] /= \_matrixDiag[\_size - 1];

for (size\_t j = \_size - 1; j > 0; j--) {

// Вычитаем все элементы, содержащие x\_j

size\_t jColFirstInd = j - \_matrixIA[j + 1] + \_matrixIA[j];

real\_t\* jColPtr = \_matrixL + (\_matrixIA[j] - jColFirstInd);

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

\_vecB[i] -= \_vecB[j] \* jColPtr[i];

}

// Высчитываем следующий x\_j (x\_(j-1))

\_vecB[j - 1] /= \_matrixDiag[j - 1];

}

}

// Функция преобразует вектор b в вектор y (перезаписывает элементы!)

void GetVectorY(size\_t \_size, real\_t\* \_matrixDiag, real\_t\* \_matrixL, size\_t\* \_matrixIA, real\_t\* \_vecB) {

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

accum\_t accum = 0.0;

size\_t iStrFirstInd = i - \_matrixIA[i + 1] + \_matrixIA[i];

real\_t\* iStrPtr = \_matrixL + (\_matrixIA[i] - iStrFirstInd);

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

accum += iStrPtr[j] \* \_vecB[j];

}

\_vecB[i] = (\_vecB[i] - accum) / \_matrixDiag[i];

}

}

// Функция преобразует переданную матрицу в матрицу L (перезаписывает элементы!)

void GetMatrixL(size\_t \_size, real\_t\* \_matrixDiag, real\_t\* \_matrixAL, size\_t\* \_matrixIA) {

\_matrixDiag[0] = std::sqrt(\_matrixDiag[0]); // Находим отдельно первый элемент

accum\_t accum = 0.0;

for (size\_t i = 1; i < \_size; i++) {

// Находим все элементы i-ой строки до диагонального элемента

size\_t iStrLen = \_matrixIA[i + 1] - \_matrixIA[i];

size\_t iStrFirstInd = i - iStrLen;

real\_t\* iStrPtr = \_matrixAL + (\_matrixIA[i] - iStrFirstInd); // Указатель на строку со смещением до "нулевой" позиции

// Идём от первого элемента в i-ой строке

for (size\_t j = iStrFirstInd, jElemInd = \_matrixIA[i]; j < i; j++, jElemInd++) {

size\_t jStrFirstInd = j - \_matrixIA[j + 1] + \_matrixIA[j];

real\_t\* jStrPtr = \_matrixAL + (\_matrixIA[j] - jStrFirstInd);

accum = 0.0;

// k выбираем как максимальный начальный индекс для двух строк

for (size\_t k = std::max(iStrFirstInd, jStrFirstInd); k < j; k++) {

accum += iStrPtr[k] \* jStrPtr[k];

}

\_matrixAL[jElemInd] = (\_matrixAL[jElemInd] - accum) / \_matrixDiag[j];

}

// Находим диагональный элемент на этой строке

accum = 0.0;

for (size\_t k = \_matrixIA[i]; k < \_matrixIA[i + 1]; k++) {

accum += \_matrixAL[k] \* \_matrixAL[k];

}

// Если пошли мнимые корни

if (accum > \_matrixDiag[i]) {

std::cout << "Во время вычисления матрицы возникли мнимые корни. Проверьте матрицу и повторите попытку." << std::endl;

throw std::exception("Illegal matrix.");

}

if (IsAlmostEq(\_matrixDiag[i], accum)) {

std::cout << "Определитель матрицы равен нулю. Проверьте матрицу и повторите попытку" << std::endl;

throw std::exception("Illegal matrix.");

}

else {

\_matrixDiag[i] = std::sqrt(\_matrixDiag[i] - accum); // Находим i-ый диагональный элемент

}

}

}

## Программа для решения СЛАУ методом Гаусса:

main.cpp:

// Выбор точности работы программы:

// \_\_ALL\_FLOAT\_\_ - все дробные числа и счётчики в float,

// \_\_SEMI\_FLOAT\_\_ - все дробные числа в float, счётчики в double,

//\_\_ALL\_DOUBLE\_\_ - все дробные числа и счётчики в double

#define \_\_ALL\_DOUBLE\_\_

#include "predefined\_types.h"

#include <iostream>

#include <fstream>

#include <vector>

#include <format>

using namespace std;

using vec\_t = vector<real\_t>;

using matrix\_t = vector<vec\_t>;

const string diagFilePath = "./matrix\_diag.txt";

const string iaFilePath = "./matrix\_IA.txt";

const string auFilePath = "./matrix\_AU.txt";

const string alFilePath = "./matrix\_AL.txt";

const string sizeFilePath = "./matrix\_size.txt";

const string vectorbFilePath = "./vector\_b.txt";

template <typename T>

void GetVectorFromFile(vector<T>& vec, const string& filePath) {

auto file = ifstream(filePath);

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

file >> vec[i];

}

file.close();

}

void GetMatrixFromProfile(matrix\_t& matrix) {

auto sizeFile = ifstream(sizeFilePath);

size\_t size;

sizeFile >> size;

sizeFile.close();

vector<size\_t> matrixIA(size + 1);

GetVectorFromFile(matrixIA, iaFilePath);

if (matrixIA[0] == 1) {

for (auto& elem : matrixIA) {

elem--;

}

}

vec\_t matrixDiag(size);

vec\_t matrixAL(matrixIA[size]);

vec\_t matrixAU(matrixIA[size]);

GetVectorFromFile(matrixDiag, diagFilePath);

GetVectorFromFile(matrixAL, alFilePath);

GetVectorFromFile(matrixAU, auFilePath);

// Заполняем саму матрицу + диагональ

matrix.resize(size);

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

matrix[i].resize(size);

matrix[i][i] = matrixDiag[i];

}

//Заполняем верхний и нижний треугольники

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

size\_t iStrFirstInd = i - matrixIA[i + 1] + matrixIA[i];

for (size\_t j = iStrFirstInd, k = matrixIA[i]; k < matrixIA[i + 1]; k++, j++) {

matrix[i][j] = matrixAL[k];

matrix[j][i] = matrixAU[k];

}

}

}

void PrintMatrixVector(matrix\_t& matrix, vec\_t& vecB) {

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

for (auto& elem : matrix[i]) {

cout << format("{0:18.{1}f}", elem, g\_coutPrecision);

}

cout << format(" | {0:18.{1}f}\n", vecB[i], g\_coutPrecision);

}

}

void SlauSolve(matrix\_t& matrix, vec\_t& b, vec\_t& x) {

void SlauSolve(matrix\_t& matrix, vec\_t& b, vec\_t& x) {

// Прямой ход (приведение к треугольному виду)

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

// выбираем максимальный элемент в i-ом столбце под i-ой строкой

size\_t maxElemInd = i;

real\_t maxElem = abs(matrix[i][i]);

for (size\_t k = i + 1; k < matrix.size(); k++) {

if (abs(matrix[k][i]) > maxElem) {

maxElem = abs(matrix[k][i]);

maxElemInd = k;

}

}

// Если элемент находится в другой строке, то меняем строки местами, а также меняем элементы в b местами

if (maxElemInd != i) {

swap(matrix[i], matrix[maxElemInd]);

swap(b[i], b[maxElemInd]);

}

// Приводим все нижележащие строки к виду, когда i-ый столбец равен нулю

for (size\_t k = i + 1; k < matrix.size(); k++) {

real\_t multCoef = matrix[k][i] / matrix[i][i];

if (!IsAlmostEq(multCoef, 0)) {

for (size\_t j = i; j < matrix.size(); j++) {

real\_t num = matrix[i][j] \* multCoef;

if (IsAlmostEq(matrix[k][j], num)) {

matrix[k][j] = 0;

}

else {

matrix[k][j] -= num;

}

}

real\_t num = b[i] \* multCoef;

if (IsAlmostEq(b[k], num)) {

b[k] = 0;

}

else {

b[k] -= num;

}

}

}

}

// Обратный ход (получение векторов x)

for (size\_t i = matrix.size(); i > 0;) {

i--; // Во избежания проблем с условием цикла (i всегда >= 0, поэтому выходил вечный цикл), вынес декремент сюда

if (IsAlmostEq(matrix[i][i], 0.0)) {

cout << "В данной СЛАУ присутствуют свободные коэффициенты. Проверьте её на корректность" << endl;

throw exception("Illegal matrix: free coefs");

}

accum\_t accum = 0.0;

for (size\_t j = i + 1; j < matrix.size(); j++) {

accum += matrix[i][j] \* x[j];

}

x[i] = (b[i] - accum) / matrix[i][i];

}

}

int main() {

setlocale(LC\_ALL, "ru-RU");

matrix\_t matrix;

GetMatrixFromProfile(matrix);

vec\_t vecB(matrix.size());

vec\_t vecX(matrix.size());

GetVectorFromFile(vecB, vectorbFilePath);

//PrintMatrixVector(matrix, vecB);

try {

SlauSolve(matrix, vecB, vecX);

}

catch (exception& e) {

cerr << e.what() << endl;

return -1;

}

cout << "Полученный вектор X: " << endl;

auto vectorxFile = ofstream(g\_outputFileName);

for (auto& elem : vecX) {

auto ss = format("{0:.{1}f}\n", elem, g\_coutPrecision);

auto index = ss.find('.');

if (index != static\_cast<size\_t>(-1)) {

ss[index] = ',';

}

cout << ss;

vectorxFile << ss;

}

vectorxFile.close();

return 0;

}

## Программа для генерации СЛАУ с матрицей Гильберта:

main.cpp:

#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

using namespace std;

using matrix = vector<vector<double>>;

const string diagFilePath = "./matrix\_diag.txt";

const string iaFilePath = "./matrix\_IA.txt";

const string auFilePath = "./matrix\_AU.txt";

const string alFilePath = "./matrix\_AL.txt";

const string sizeFilePath = "./matrix\_size.txt";

const string vectorbFilePath = "./vector\_b.txt";

void getHilbertMatrix(matrix& mas) {

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

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

mas[i][j] = 1.0 / (i + j + 1);

}

}

}

void fprintProfileMatrix(matrix& mas) {

auto sizeFile = ofstream(sizeFilePath);

sizeFile << mas.size() << endl;

sizeFile.close();

auto diagFile = ofstream(diagFilePath);

diagFile.precision(15);

diagFile.setf(std::ios::fixed);

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

diagFile << mas[i][i] << " ";

}

diagFile << endl;

diagFile.close();

auto iaFile = ofstream(iaFilePath);

auto auFile = ofstream(auFilePath);

auto alFile = ofstream(alFilePath);

auFile.precision(15);

auFile.setf(std::ios::fixed);

alFile.precision(15);

alFile.setf(std::ios::fixed);

iaFile << "1 ";

int lastIAindex = 1;

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

int strLen = 0;

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

if (mas[j][i] == 0) continue;

auFile << mas[j][i] << " ";

alFile << mas[i][j] << " ";

strLen++;

}

lastIAindex += strLen;

iaFile << lastIAindex << " ";

}

iaFile << endl;

auFile << endl;

alFile << endl;

iaFile.close();

auFile.close();

alFile.close();

}

void getVectorX(vector<double>& x) {

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

x[i] = i + 1;

}

}

void getVectorB(matrix& mas, vector<double>& x, vector<double>& b) {

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

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

b[i] += mas[i][j] \* x[j];

}

}

}

void fprintVector(vector<double>& b) {

auto vecFile = ofstream(vectorbFilePath);

vecFile.precision(15);

vecFile.setf(std::ios::fixed);

for (auto elem : b) {

vecFile << elem << " ";

}

vecFile << endl;

vecFile.close();

}

int main(int argc, char \*\*argv) {

setlocale(LC\_ALL, "ru-RU");

int matrixSize = 0;

if (argc < 2) {

cout << "Введите размер матрицы: ";

cin >> matrixSize;

}

else {

stringstream ss;

ss << argv[1];

ss >> matrixSize;

}

matrix mas;

mas.resize(matrixSize);

for (auto& elem : mas) {

elem.resize(matrixSize);

}

vector<double> x(matrixSize);

vector<double> b(matrixSize);

getHilbertMatrix(mas);

fprintProfileMatrix(mas);

getVectorX(x);

getVectorB(mas, x, b);

fprintVector(b);

return 0;

}

# Тестирование программы

## Часть 1. Набор тестов для проверки правильности работы программы:

*Ввод программы:*

n = 7

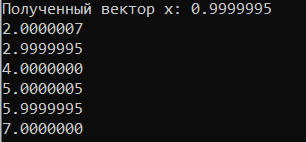
di = 1 2 3 4 5 6 7

ia = 1 1 1 2 3 7 8 12

al = 1 1 1 1 1 1 1 1 1 1 1

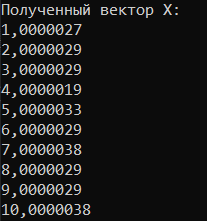
b = 6 12 27 31 48 48 67

*Вывод программы (тестирование в типе float):*

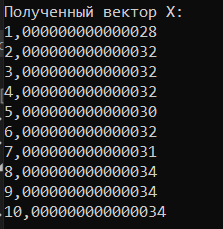


## ***Решение СЛАУ методом Гаусса:***

*Вывод программы (тестирование в типе float):*



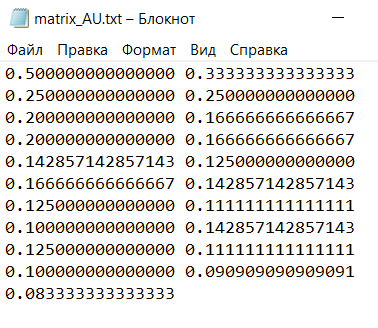
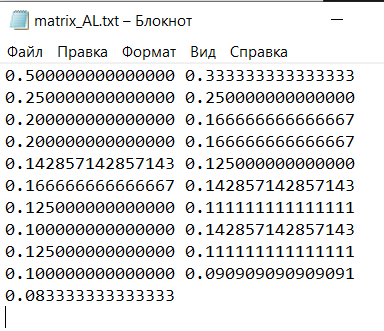
*Вывод программы (тестирование в типе double):*

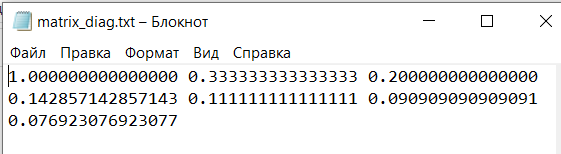


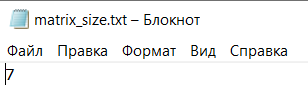
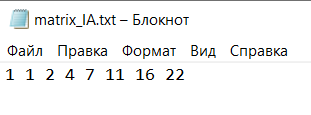
## ***Генерация СЛАУ по матрице Гильберта размера*** ***k:***

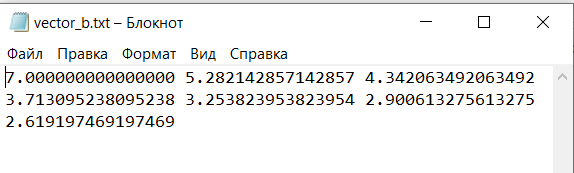
Вывод программы: набор файлов, содержащих матрицу А в профильном формате, и соответствующий вектор b:

k = 7



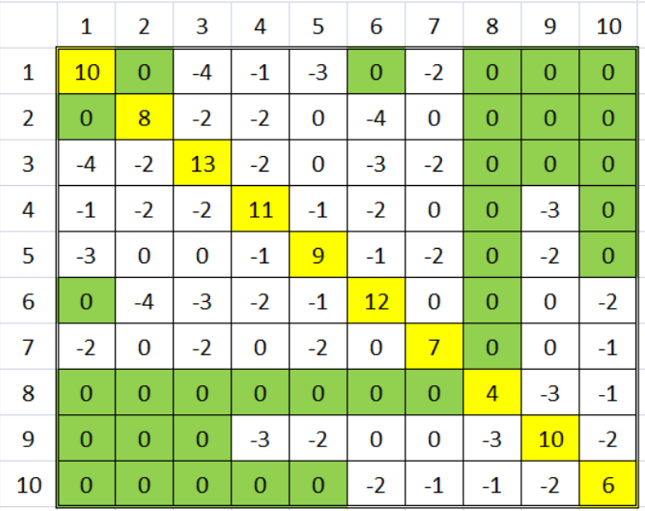






# Исследование на матрице 10х10:

**Изначальная матрица:**



**size** = 10

**di** = 10 8 13 11 9 12 7 4 10 6

**ia** = 1 1 1 3 6 10 14 20 20 25 29

**al** = -4 -2 -1 -2 -2 -3 0 0 -1 -4 -3 -2 -1 -2 0 -2 0 -2 0 -3 -2 0 0 -3 -2 -1 -1 -2

**x** = 1 2 3 4 5 6 7 8 9 10

**b** = -35 -22 -9 -11 0 22 21 -5 24 15

При исследовании к первому элементу прибавляется значение , k меняется

1. **Оценка влияния увеличения числа обусловленности на точность решения:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k | x^\* | x^k (одинарная точность) | x^\*-x^k (одинарная точность) | x^k (двойная точность) | x^\*- x^k (двойная точность) | x^k (скаляр. Произв.) | x^\* - x^k (скаляр. Произв.) |
| 0 | 1 | 0,9999982 | 1,8E-06 | 0,999999999999989 | 1,1E-14 | 0,9999935 | 6,5E-06 |
| 2 | 1,999998 | 2,1E-06 | 1,99999999999998 | 2,0E-14 | 1,999993 | 7,3E-06 |
| 3 | 2,999998 | 1,7E-06 | 2,99999999999998 | 2,0E-14 | 2,999993 | 7,2E-06 |
| 4 | 3,999998 | 2,4E-06 | 3,99999999999998 | 2,0E-14 | 3,999993 | 7,2E-06 |
| 5 | 4,999998 | 2,4E-06 | 4,99999999999998 | 2,0E-14 | 4,999993 | 7,2E-06 |
| 6 | 5,999998 | 1,9E-06 | 5,99999999999998 | 2,0E-14 | 5,999993 | 7,2E-06 |
| 7 | 6,999998 | 2,4E-06 | 6,99999999999998 | 2,0E-14 | 6,999993 | 6,7E-06 |
| 8 | 7,999997 | 2,9E-06 | 7,99999999999998 | 2,0E-14 | 7,999993 | 6,7E-06 |
| 9 | 8,999997 | 2,9E-06 | 8,99999999999998 | 2,0E-14 | 8,999993 | 6,7E-06 |
| 10 | 9,999996 | 3,8E-06 | 9,99999999999998 | 2,0E-14 | 9,999992 | 7,6E-06 |
| 1 | 1 | 0,9999884 | 1,2E-05 | 1,00000000000005 | -5,0E-14 | 0,9999777 | 2,2E-05 |
| 2 | 1,9999883 | 1,2E-05 | 2,00000000000005 | -5,0E-14 | 1,9999771 | 2,3E-05 |
| 3 | 2,9999886 | 1,1E-05 | 3,00000000000005 | -5,0E-14 | 2,9999776 | 2,2E-05 |
| 4 | 3,999989 | 1,1E-05 | 4,00000000000005 | -5,0E-14 | 3,9999774 | 2,3E-05 |
| 5 | 4,9999886 | 1,1E-05 | 5,00000000000005 | -5,0E-14 | 4,9999781 | 2,2E-05 |
| 6 | 5,9999886 | 1,1E-05 | 6,00000000000005 | -5,0E-14 | 5,9999771 | 2,3E-05 |
| 7 | 6,9999886 | 1,1E-05 | 7,00000000000005 | -5,0E-14 | 6,9999781 | 2,2E-05 |
| 8 | 7,9999886 | 1,1E-05 | 8,00000000000005 | -5,0E-14 | 7,9999771 | 2,3E-05 |
| 9 | 8,9999886 | 1,1E-05 | 9,00000000000005 | -5,0E-14 | 8,9999771 | 2,3E-05 |
| 10 | 9,9999895 | 1,1E-05 | 10,0000000000000 | 0,0E+00 | 9,9999771 | 2,3E-05 |
| 2 | 1 | 0,9992355 | 7,6E-04 | 1,00000000000178 | -1,8E-12 | 1,0005131 | -5,1E-04 |
| 2 | 1,9992344 | 7,7E-04 | 2,00000000000178 | -1,8E-12 | 2,0005131 | -5,1E-04 |
| 3 | 2,9992349 | 7,7E-04 | 3,00000000000178 | -1,8E-12 | 3,0005133 | -5,1E-04 |
| 4 | 3,9992347 | 7,7E-04 | 4,00000000000178 | -1,8E-12 | 4,0005136 | -5,1E-04 |
| 5 | 4,9992342 | 7,7E-04 | 5,00000000000178 | -1,8E-12 | 5,0005140 | -5,1E-04 |
| 6 | 5,9992347 | 7,7E-04 | 6,00000000000178 | -1,8E-12 | 6,0005131 | -5,1E-04 |
| 7 | 6,9992347 | 7,7E-04 | 7,00000000000178 | -1,8E-12 | 7,0005136 | -5,1E-04 |
| 8 | 7,9992342 | 7,7E-04 | 8,00000000000178 | -1,8E-12 | 8,0005140 | -5,1E-04 |
| 9 | 8,9992342 | 7,7E-04 | 9,00000000000178 | -1,8E-12 | 9,0005140 | -5,1E-04 |
| 10 | 9,9992342 | 7,7E-04 | 10,0000000000017 | -1,7E-12 | ######### | -5,1E-04 |
| 3 | 1 | 1,0047717 | -4,8E-03 | 1,00000000000533 | -5,3E-12 | 0,9996427 | 3,6E-04 |
| 2 | 2,0047724 | -4,8E-03 | 2,00000000000533 | -5,3E-12 | 1,9996424 | 3,6E-04 |
| 3 | 3,0047724 | -4,8E-03 | 3,00000000000533 | -5,3E-12 | 2,9996426 | 3,6E-04 |
| 4 | 4,0047722 | -4,8E-03 | 4,00000000000533 | -5,3E-12 | 3,9996428 | 3,6E-04 |
| 5 | 5,0047717 | -4,8E-03 | 5,00000000000533 | -5,3E-12 | 4,9996424 | 3,6E-04 |
| 6 | 6,0047727 | -4,8E-03 | 6,00000000000533 | -5,3E-12 | 5,9996424 | 3,6E-04 |
| 7 | 7,0047722 | -4,8E-03 | 7,00000000000533 | -5,3E-12 | 6,9996424 | 3,6E-04 |
| 8 | 8,0047722 | -4,8E-03 | 8,00000000000533 | -5,3E-12 | 7,9996424 | 3,6E-04 |
| 9 | 9,0047722 | -4,8E-03 | 9,00000000000533 | -5,3E-12 | 8,9996424 | 3,6E-04 |
| 10 | 10,004773 | -4,8E-03 | 10,0000000000053 | -5,3E-12 | 9,9996424 | 3,6E-04 |
|  |  |  |  |  |  |  |  |
| 4 | 1 | 0,9339619 | 6,6E-02 | 0,999999999822362 | 1,8E-10 | 1,0287756 | -2,9E-02 |
| 2 | 1,9339604 | 6,6E-02 | 1,99999999982236 | 1,8E-10 | 2,0287759 | -2,9E-02 |
| 3 | 2,9339612 | 6,6E-02 | 2,99999999982236 | 1,8E-10 | 3,0287762 | -2,9E-02 |
| 4 | 3,9339612 | 6,6E-02 | 3,99999999982236 | 1,8E-10 | 4,0287762 | -2,9E-02 |
| 5 | 4,9339614 | 6,6E-02 | 4,99999999982236 | 1,8E-10 | 5,0287757 | -2,9E-02 |
| 6 | 5,9339604 | 6,6E-02 | 5,99999999982236 | 1,8E-10 | 6,0287766 | -2,9E-02 |
| 7 | 6,9339619 | 6,6E-02 | 6,99999999982236 | 1,8E-10 | 7,0287762 | -2,9E-02 |
| 8 | 7,9339619 | 6,6E-02 | 7,99999999982235 | 1,8E-10 | 8,0287762 | -2,9E-02 |
| 9 | 8,9339619 | 6,6E-02 | 8,99999999982235 | 1,8E-10 | 9,0287762 | -2,9E-02 |
| 10 | 9,9339619 | 6,6E-02 | 9,99999999982235 | 1,8E-10 | 10,028777 | -2,9E-02 |
| 5 | 1 | -0,1111096 | 1,1E+00 | 0,99999999769073 | 2,3E-09 | 0,1368439 | 8,6E-01 |
| 2 | 0,8888882 | 1,1E+00 | 1,99999999769073 | 2,3E-09 | 1,1368423 | 8,6E-01 |
| 3 | 1,888889 | 1,1E+00 | 2,99999999769073 | 2,3E-09 | 2,1368427 | 8,6E-01 |
| 4 | 2,8888888 | 1,1E+00 | 3,99999999769073 | 2,3E-09 | 3,1368425 | 8,6E-01 |
| 5 | 3,8888896 | 1,1E+00 | 4,99999999769073 | 2,3E-09 | 4,1368432 | 8,6E-01 |
| 6 | 4,8888879 | 1,1E+00 | 5,99999999769073 | 2,3E-09 | 5,1368423 | 8,6E-01 |
| 7 | 5,8888893 | 1,1E+00 | 6,99999999769073 | 2,3E-09 | 6,1368427 | 8,6E-01 |
| 8 | 6,8888893 | 1,1E+00 | 7,99999999769072 | 2,3E-09 | 7,1368418 | 8,6E-01 |
| 9 | 7,8888893 | 1,1E+00 | 8,99999999769072 | 2,3E-09 | 8,1368418 | 8,6E-01 |
| 10 | 8,8888884 | 1,1E+00 | 9,99999999769072 | 2,3E-09 | 9,1368418 | 8,6E-01 |
| 6 | 1 | - | - | 0,999999994670924 | 5,3E-09 | - | - |
| 2 | 1,99999999467092 | 5,3E-09 |
| 3 | 2,99999999467092 | 5,3E-09 |
| 4 | 3,99999999467092 | 5,3E-09 |
| 5 | 4,99999999467092 | 5,3E-09 |
| 6 | 5,99999999467092 | 5,3E-09 |
| 7 | 6,99999999467092 | 5,3E-09 |
| 8 | 7,99999999467092 | 5,3E-09 |
| 9 | 8,99999999467092 | 5,3E-09 |
| 10 | 9,99999999467092 | 5,3E-09 |
| 7 | 1 | - | - | 1,00000014210855 | -1,4E-07 | - | - |
| 2 | 2,00000014210855 | -1,4E-07 |
| 3 | 3,00000014210855 | -1,4E-07 |
| 4 | 4,00000014210855 | -1,4E-07 |
| 5 | 5,00000014210855 | -1,4E-07 |
| 6 | 6,00000014210855 | -1,4E-07 |
| 7 | 7,00000014210855 | -1,4E-07 |
| 8 | 8,00000014210855 | -1,4E-07 |
| 9 | 9,00000014210855 | -1,4E-07 |
| 10 | 10,0000001421085 | -1,4E-07 |
|  |  |  |  |  |  |  |  |
| 8 | 1 | - | - | 0,999999999999998 | 2,0E-15 | - | - |
| 2 | 1,99999999999999 | 1,0E-14 |
| 3 | 2,99999999999999 | 1,0E-14 |
| 4 | 3,99999999999999 | 1,0E-14 |
| 5 | 4,99999999999999 | 9,8E-15 |
| 6 | 5,99999999999999 | 9,8E-15 |
| 7 | 6,99999999999999 | 9,8E-15 |
| 8 | 8,00000000000000 | 0,0E+00 |
| 9 | 9,00000000000000 | 0,0E+00 |
| 10 | 10,00000000000000 | 0,0E+00 |
| 9 | 1 | - | - | 1,00001421087878 | -1,4E-05 | - | - |
| 2 | 2,00001421087878 | -1,4E-05 |
| 3 | 3,00001421087878 | -1,4E-05 |
| 4 | 4,00001421087878 | -1,4E-05 |
| 5 | 5,00001421087878 | -1,4E-05 |
| 6 | 6,00001421087878 | -1,4E-05 |
| 7 | 7,00001421087878 | -1,4E-05 |
| 8 | 8,00001421087878 | -1,4E-05 |
| 9 | 9,00001421087878 | -1,4E-05 |
| 10 | 10,0000142108787 | -1,4E-05 |
| 10 | 1 | - | - | 0,99985789398892 | 1,4E-04 | - | - |
| 2 | 1,99985789398891 | 1,4E-04 |
| 3 | 2,99985789398891 | 1,4E-04 |
| 4 | 3,99985789398891 | 1,4E-04 |
| 5 | 4,99985789398891 | 1,4E-04 |
| 6 | 5,99985789398891 | 1,4E-04 |
| 7 | 6,99985789398891 | 1,4E-04 |
| 8 | 7,99985789398891 | 1,4E-04 |
| 9 | 8,99985789398891 | 1,4E-04 |
| 10 | 9,99985789398891 | 1,4E-04 |
| 11 | 1 | - | - | 0,99822411649796 | 1,8E-03 | - | - |
| 2 | 1,99822411649795 | 1,8E-03 |
| 3 | 2,99822411649795 | 1,8E-03 |
| 4 | 3,99822411649795 | 1,8E-03 |
| 5 | 4,99822411649795 | 1,8E-03 |
| 6 | 5,99822411649795 | 1,8E-03 |
| 7 | 6,99822411649795 | 1,8E-03 |
| 8 | 7,99822411649795 | 1,8E-03 |
| 9 | 8,99822411649795 | 1,8E-03 |
| 10 | 9,99822411649795 | 1,8E-03 |
|  |  |  |  |  |  |  |  |
| 12 | 1 | - | - | 0,99115826702034 | 8,8E-03 | - | - |
| 2 | 1,99115826702033 | 8,8E-03 |
| 3 | 2,99115826702033 | 8,8E-03 |
| 4 | 3,99115826702033 | 8,8E-03 |
| 5 | 4,99115826702033 | 8,8E-03 |
| 6 | 5,99115826702033 | 8,8E-03 |
| 7 | 6,99115826702033 | 8,8E-03 |
| 8 | 7,99115826702033 | 8,8E-03 |
| 9 | 8,99115826702033 | 8,8E-03 |
| 10 | 9,99115826702033 | 8,8E-03 |
| 13 | 1 | - | - | 0,87826086956522 | 1,2E-01 | - | - |
| 2 | 1,87826086956521 | 1,2E-01 |
| 3 | 2,87826086956521 | 1,2E-01 |
| 4 | 3,87826086956521 | 1,2E-01 |
| 5 | 4,87826086956521 | 1,2E-01 |
| 6 | 5,87826086956521 | 1,2E-01 |
| 7 | 6,87826086956521 | 1,2E-01 |
| 8 | 7,87826086956521 | 1,2E-01 |
| 9 | 8,87826086956521 | 1,2E-01 |
| 10 | 9,87826086956521 | 1,2E-01 |

Погрешность результата на смешанной точности (скалярные произведения двойной точности) немного меньше одинарной точности, при этом погрешность на двойной точности значительно меньше.

С увеличением k падает точность вычисленного вектора x. Связано это с тем, что при увеличении k увеличивается обусловленность матрицы A и слегка меняется вектор b. При чтении программой вектора b появляется небольшая погрешность представления значения. За счёт того, что обусловленность матрицы A возросла, вся СЛАУ стала более зависимой от погрешностей вычисления и начальных данных, в результате значительно преумножив погрешность решения x.

Программа перестала работать при k=6 для float и при k=14 для double потому, что при данных значениях k мантисса числа A11 (8 и 16 знаков соответственно) не помещается полностью в мантиссу переменных компьютера (мантисса 7.5 и 15 знаков соответственно), а потому число обрезается и матрица принимает вид первоначальной матрицы, которая является вырожденной, а такие матрицы разложением LLt однозначно решить нельзя.

1. **Оценка влияния числа обусловленности на точность решения на матрицах Гильберта:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| k | x^\* | 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 (скаляр. Произв.) |
| 1 | 1 | 1,0000000 | 0,0E+00 | 1,00000000000000 | 0,0E+00 | 1,0000000 | 0,0E+00 |
| 2 | 1 | 1,0000005 | -5,0E-07 | 0,99999999999999 | 6,0E-15 | 1,0000005 | -5,0E-07 |
| 2 | 1,9999992 | 8,0E-07 | 2,00000000000001 | -1,0E-14 | 1,9999992 | 8,0E-07 |
| 3 | 1 | 1,0000019 | -1,9E-06 | 0,99999999999997 | 2,6E-14 | 1,0000002 | -2,0E-07 |
| 2 | 1,9999890 | 1,1E-05 | 2,00000000000015 | -1,5E-13 | 1,9999994 | 6,0E-07 |
| 3 | 3,0000105 | -1,1E-05 | 2,99999999999985 | 1,5E-13 | 3,0000105 | -1,1E-05 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 4 | 1 | 0,9999877 | 1,2E-05 | 0,99999999999996 | 4,4E-14 | 0,9999819 | 1,8E-05 |
| 2 | 2,0001223 | -1,2E-04 | 2,00000000000026 | -2,6E-13 | 2,0001841 | -1,8E-04 |
| 3 | 2,9997272 | 2,7E-04 | 2,99999999999973 | 2,7E-13 | 2,9995818 | 4,2E-04 |
| 4 | 4,0001674 | -1,7E-04 | 4,00000000000000 | 0,0E+00 | 4,0002608 | -2,6E-04 |
| 5 | 1 | 0,9999734 | 2,7E-05 | 0,99999999999889 | 1,1E-12 | 1,0000250 | -2,5E-05 |
| 2 | 2,0005662 | -5,7E-04 | 2,00000000002046 | -2,0E-11 | 1,9995662 | 4,3E-04 |
| 3 | 2,9974306 | 2,6E-03 | 2,99999999991327 | 8,7E-11 | 3,0018432 | -1,8E-03 |
| 4 | 4,0039706 | -4,0E-03 | 4,00000000012908 | -1,3E-10 | 3,9972157 | 2,8E-03 |
| 5 | 4,9980354 | 2,0E-03 | 4,99999999993757 | 6,2E-11 | 5,0013690 | -1,4E-03 |
| 6 | 1 | 1,0005283 | -5,3E-04 | 0,99999999999846 | 1,5E-12 | 1,0005548 | -5,5E-04 |
| 2 | 1,9892746 | 1,1E-02 | 2,00000000003960 | -4,0E-11 | 1,9874848 | 1,3E-02 |
| 3 | 3,0543039 | -5,4E-02 | 2,99999999975452 | 2,5E-10 | 3,0704846 | -7,0E-02 |
| 4 | 3,8914797 | 1,1E-01 | 4,00000000059481 | -5,9E-10 | 3,8424528 | 1,6E-01 |
| 5 | 5,0932388 | -9,3E-02 | 4,99999999938040 | 6,2E-10 | 5,1527419 | -1,5E-01 |
| 6 | 5,9712925 | 2,9E-02 | 6,00000000023266 | -2,3E-10 | 5,9462752 | 5,4E-02 |
| 7 | 1 | 0,9966615 | 3,3E-03 | 0,99999999994816 | 5,2E-11 | 1,0060165 | -6,0E-03 |
| 2 | 2,1497023 | -1,5E-01 | 2,00000000213563 | -2,1E-09 | 1,7555341 | 2,4E-01 |
| 3 | 1,4473505 | 1,6E+00 | 2,99999997898349 | 2,1E-08 | 5,3824620 | -2,4E+00 |
| 4 | 10,3579378 | -6,4E+00 | 4,00000008296186 | -8,3E-08 | -5,3350449 | 9,3E+00 |
| 5 | -7,1180482 | 1,2E+01 | 4,99999984614889 | 1,5E-07 | 22,2102814 | -1,7E+01 |
| 6 | 16,7937546 | -1,1E+01 | 6,00000013413927 | -1,3E-07 | -8,9341421 | 1,5E+01 |
| 7 | 3,3684211 | 3,6E+00 | 6,99999995563913 | 4,4E-08 | 11,9195375 | -4,9E+00 |
| 8 | 1 | - | - | 1,000000000467220 | -4,7E-10 | - | - |
| 2 | 1,99999997475567 | 2,5E-08 |
| 3 | 3,00000033213047 | -3,3E-07 |
| 4 | 3,99999819035825 | 1,8E-06 |
| 5 | 5,00000490138199 | -4,9E-06 |
| 6 | 5,99999302788620 | 7,0E-06 |
| 7 | 7,00000498494722 | -5,0E-06 |
| 8 | 7,99999858769972 | 1,4E-06 |
| 9 | 1 | - | - | 0,99999999172222 | 8,3E-09 | - | - |
| 2 | 2,00000056772607 | -5,7E-07 |
| 3 | 2,99999042889671 | 9,6E-06 |
| 4 | 4,00006813616557 | -6,8E-05 |
| 5 | 4,99975059464219 | 2,5E-04 |
| 6 | 6,00050843506325 | -5,1E-04 |
| 7 | 6,99941682156542 | 5,8E-04 |
| 8 | 8,00035188797683 | -3,5E-04 |
| 9 | 8,99991313058972 | 8,7E-05 |
|  |  |  |  |  |  |  |  |
| 10 | 1 | - | - | 1,00000000960873 | -9,6E-09 | - | - |
| 2 | 1,99999924128823 | 7,6E-07 |
| 3 | 3,00001499792620 | -1,5E-05 |
| 4 | 3,99987203993835 | 1,3E-04 |
| 5 | 5,00057769188825 | -5,8E-04 |
| 6 | 5,99848678505499 | 1,5E-03 |
| 7 | 7,00237855967474 | -2,4E-03 |
| 8 | 7,99778795088552 | 2,2E-03 |
| 9 | 9,00112178711574 | -1,1E-03 |
| 10 | 9,99976093454208 | 2,4E-04 |
| 11 | 1 | - | - | 1,00000014998147 | -1,5E-07 | - | - |
| 2 | 1,99998388881467 | 1,6E-05 |
| 3 | 3,00042605264422 | -4,3E-04 |
| 4 | 3,99516523562639 | 4,8E-03 |
| 5 | 5,02914649999994 | -2,9E-02 |
| 6 | 5,89652773939997 | 1,0E-01 |
| 7 | 7,22711482605527 | -2,3E-01 |
| 8 | 7,68826682548186 | 3,1E-01 |
| 9 | 9,26044931986466 | -2,6E-01 |
| 10 | 9,87889083596274 | 1,2E-01 |
| 11 | 11,0240287373990 | -2,4E-02 |
| 12 | 1 | - | - | 0,99999994634208 | 5,4E-08 | - | - |
| 2 | 2,00000051645272 | -5,2E-07 |
| 3 | 3,00014493537155 | -1,4E-04 |
| 4 | 3,99620127606946 | 3,8E-03 |
| 5 | 5,03912525888273 | -3,9E-02 |
| 6 | 5,78575544921813 | 2,1E-01 |
| 7 | 7,69886024156141 | -7,0E-01 |
| 8 | 6,57402450273745 | 1,4E+00 |
| 9 | 10,8377931276082 | -1,8E+00 |
| 10 | 8,54625578074441 | 1,5E+00 |
| 11 | 11,6444143633113 | -6,4E-01 |
| 12 | 11,8774243122790 | 1,2E-01 |

Погрешность вычислений со смешанной точностью также незначительно меньше погрешности с одинарной точностью, погрешность вычислений с двойной точностью гораздо меньше погрешности вычислений с одинарной точностью.

Погрешность вычислений, так же, как и в предыдущем исследовании, увеличивается за счёт увеличения числа обусловленности матрицы при увеличении k, а также за счёт погрешности вычисления вектора b.

При этом при увеличении k уменьшается определитель матрицы, поэтому при k = 8 для float и при k = 13 для double определитель матрицы в памяти ЭВМ равен нулю, матрица становится обусловленной, а при таком условии нельзя однозначно решить СЛАУ методом LLt.

1. **Метод Гаусса:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| k | x^\* | x^k (LLt, двойная точность) | x^\*- x^k (LLt, двойная точность) | x^k (Гаусс, двойная точность) | x^\*- x^k (Гаусс, двойная точность) |
| 0 | 1 | 0,999999999999989 | 1,1E-14 | 1,00000000000002 | -2,0E-14 |
| 2 | 1,99999999999998 | 2,0E-14 | 2,00000000000003 | -3,0E-14 |
| 3 | 2,99999999999998 | 2,0E-14 | 3,00000000000003 | -3,0E-14 |
| 4 | 3,99999999999998 | 2,0E-14 | 4,00000000000003 | -3,0E-14 |
| 5 | 4,99999999999998 | 2,0E-14 | 5,00000000000003 | -3,0E-14 |
| 6 | 5,99999999999998 | 2,0E-14 | 6,00000000000003 | -3,0E-14 |
| 7 | 6,99999999999998 | 2,0E-14 | 7,00000000000003 | -3,0E-14 |
| 8 | 7,99999999999998 | 2,0E-14 | 8,00000000000003 | -3,0E-14 |
| 9 | 8,99999999999998 | 2,0E-14 | 9,00000000000003 | -3,0E-14 |
| 10 | 9,99999999999998 | 2,0E-14 | 10,0000000000000 | 0,0E+00 |
| 1 | 1 | 1,00000000000005 | -5,0E-14 | 0,999999999999807 | 1,9E-13 |
| 2 | 2,00000000000005 | -5,0E-14 | 1,99999999999980 | 2,0E-13 |
| 3 | 3,00000000000005 | -5,0E-14 | 2,99999999999980 | 2,0E-13 |
| 4 | 4,00000000000005 | -5,0E-14 | 3,99999999999980 | 2,0E-13 |
| 5 | 5,00000000000005 | -5,0E-14 | 4,99999999999980 | 2,0E-13 |
| 6 | 6,00000000000005 | -5,0E-14 | 5,99999999999980 | 2,0E-13 |
| 7 | 7,00000000000005 | -5,0E-14 | 6,99999999999980 | 2,0E-13 |
| 8 | 8,00000000000005 | -5,0E-14 | 7,99999999999980 | 2,0E-13 |
| 9 | 9,00000000000005 | -5,0E-14 | 8,99999999999980 | 2,0E-13 |
| 10 | 10,0000000000000 | 0,0E+00 | 9,99999999999980 | 2,0E-13 |
| 2 | 1 | 1,00000000000178 | -1,8E-12 | 0,999999999999911 | 8,9E-14 |
| 2 | 2,00000000000178 | -1,8E-12 | 1,99999999999991 | 9,0E-14 |
| 3 | 3,00000000000178 | -1,8E-12 | 2,99999999999991 | 9,0E-14 |
| 4 | 4,00000000000178 | -1,8E-12 | 3,99999999999991 | 9,0E-14 |
| 5 | 5,00000000000178 | -1,8E-12 | 4,99999999999991 | 9,0E-14 |
| 6 | 6,00000000000178 | -1,8E-12 | 5,99999999999991 | 9,0E-14 |
| 7 | 7,00000000000178 | -1,8E-12 | 6,99999999999991 | 9,0E-14 |
| 8 | 8,00000000000178 | -1,8E-12 | 7,99999999999991 | 9,0E-14 |
| 9 | 9,00000000000178 | -1,8E-12 | 8,99999999999991 | 9,1E-14 |
| 10 | 10,0000000000017 | -1,7E-12 | 9,99999999999991 | 9,1E-14 |
| 3 | 1 | 1,00000000000533 | -5,3E-12 | 0,999999999988450 | 1,2E-11 |
| 2 | 2,00000000000533 | -5,3E-12 | 1,99999999998844 | 1,2E-11 |
| 3 | 3,00000000000533 | -5,3E-12 | 2,99999999998844 | 1,2E-11 |
| 4 | 4,00000000000533 | -5,3E-12 | 3,99999999998845 | 1,2E-11 |
| 5 | 5,00000000000533 | -5,3E-12 | 4,99999999998844 | 1,2E-11 |
| 6 | 6,00000000000533 | -5,3E-12 | 5,99999999998844 | 1,2E-11 |
| 7 | 7,00000000000533 | -5,3E-12 | 6,99999999998844 | 1,2E-11 |
| 8 | 8,00000000000533 | -5,3E-12 | 7,99999999998845 | 1,2E-11 |
| 9 | 9,00000000000533 | -5,3E-12 | 8,99999999998844 | 1,2E-11 |
| 10 | 10,0000000000053 | -5,3E-12 | 9,99999999998845 | 1,2E-11 |
|  |  |  |  |  |  |
| 4 | 1 | 0,999999999822362 | 1,8E-10 | 0,999999999964473 | 3,6E-11 |
| 2 | 1,99999999982236 | 1,8E-10 | 1,99999999996447 | 3,6E-11 |
| 3 | 2,99999999982236 | 1,8E-10 | 2,99999999996447 | 3,6E-11 |
| 4 | 3,99999999982236 | 1,8E-10 | 3,99999999996447 | 3,6E-11 |
| 5 | 4,99999999982236 | 1,8E-10 | 4,99999999996447 | 3,6E-11 |
| 6 | 5,99999999982236 | 1,8E-10 | 5,99999999996447 | 3,6E-11 |
| 7 | 6,99999999982236 | 1,8E-10 | 6,99999999996447 | 3,6E-11 |
| 8 | 7,99999999982235 | 1,8E-10 | 7,99999999996447 | 3,6E-11 |
| 9 | 8,99999999982235 | 1,8E-10 | 8,99999999996447 | 3,6E-11 |
| 10 | 9,99999999982235 | 1,8E-10 | 9,99999999996447 | 3,6E-11 |
| 5 | 1 | 0,99999999769073 | 2,3E-09 | 1,00000000008882 | -8,9E-11 |
| 2 | 1,99999999769073 | 2,3E-09 | 2,00000000008882 | -8,9E-11 |
| 3 | 2,99999999769073 | 2,3E-09 | 3,00000000008882 | -8,9E-11 |
| 4 | 3,99999999769073 | 2,3E-09 | 4,00000000008881 | -8,9E-11 |
| 5 | 4,99999999769073 | 2,3E-09 | 5,00000000008882 | -8,9E-11 |
| 6 | 5,99999999769073 | 2,3E-09 | 6,00000000008882 | -8,9E-11 |
| 7 | 6,99999999769073 | 2,3E-09 | 7,00000000008881 | -8,9E-11 |
| 8 | 7,99999999769072 | 2,3E-09 | 8,00000000008882 | -8,9E-11 |
| 9 | 8,99999999769072 | 2,3E-09 | 9,00000000008882 | -8,9E-11 |
| 10 | 9,99999999769072 | 2,3E-09 | 10,0000000000888 | -8,9E-11 |
| 6 | 1 | 0,999999994670924 | 5,3E-09 | 1,00000001154631 | -1,2E-08 |
| 2 | 1,99999999467092 | 5,3E-09 | 2,00000001154632 | -1,2E-08 |
| 3 | 2,99999999467092 | 5,3E-09 | 3,00000001154632 | -1,2E-08 |
| 4 | 3,99999999467092 | 5,3E-09 | 4,00000001154632 | -1,2E-08 |
| 5 | 4,99999999467092 | 5,3E-09 | 5,00000001154632 | -1,2E-08 |
| 6 | 5,99999999467092 | 5,3E-09 | 6,00000001154632 | -1,2E-08 |
| 7 | 6,99999999467092 | 5,3E-09 | 7,00000001154632 | -1,2E-08 |
| 8 | 7,99999999467092 | 5,3E-09 | 8,00000001154632 | -1,2E-08 |
| 9 | 8,99999999467092 | 5,3E-09 | 9,00000001154632 | -1,2E-08 |
| 10 | 9,99999999467092 | 5,3E-09 | 10,0000000115463 | -1,2E-08 |
| 7 | 1 | 1,00000014210855 | -1,4E-07 | 1,00000000000000 | 0,0E+00 |
| 2 | 2,00000014210855 | -1,4E-07 | 1,99999999999999 | 1,0E-14 |
| 3 | 3,00000014210855 | -1,4E-07 | 3,00000000000000 | 0,0E+00 |
| 4 | 4,00000014210855 | -1,4E-07 | 3,99999999999999 | 1,0E-14 |
| 5 | 5,00000014210855 | -1,4E-07 | 4,99999999999999 | 9,8E-15 |
| 6 | 6,00000014210855 | -1,4E-07 | 5,99999999999999 | 9,8E-15 |
| 7 | 7,00000014210855 | -1,4E-07 | 6,99999999999999 | 9,8E-15 |
| 8 | 8,00000014210855 | -1,4E-07 | 8,00000000000000 | 0,0E+00 |
| 9 | 9,00000014210855 | -1,4E-07 | 9,00000000000000 | 0,0E+00 |
| 10 | 10,0000001421085 | -1,4E-07 | 10,0000000000000 | 0,0E+00 |
|  |  |  |  |  |  |
| 8 | 1 | 0,999999999999998 | 2,0E-15 | 0,999999911182172 | 8,9E-08 |
| 2 | 1,99999999999999 | 1,0E-14 | 1,99999991118217 | 8,9E-08 |
| 3 | 2,99999999999999 | 1,0E-14 | 2,99999991118217 | 8,9E-08 |
| 4 | 3,99999999999999 | 1,0E-14 | 3,99999991118217 | 8,9E-08 |
| 5 | 4,99999999999999 | 9,8E-15 | 4,99999991118217 | 8,9E-08 |
| 6 | 5,99999999999999 | 9,8E-15 | 5,99999991118217 | 8,9E-08 |
| 7 | 6,99999999999999 | 9,8E-15 | 6,99999991118217 | 8,9E-08 |
| 8 | 8,00000000000000 | 0,0E+00 | 7,99999991118217 | 8,9E-08 |
| 9 | 9,00000000000000 | 0,0E+00 | 8,99999991118217 | 8,9E-08 |
| 10 | 10,00000000000000 | 0,0E+00 | 9,99999991118217 | 8,9E-08 |
| 9 | 1 | 1,00001421087878 | -1,4E-05 | 0,999989341859848 | 1,1E-05 |
| 2 | 2,00001421087878 | -1,4E-05 | 1,99998934185984 | 1,1E-05 |
| 3 | 3,00001421087878 | -1,4E-05 | 2,99998934185984 | 1,1E-05 |
| 4 | 4,00001421087878 | -1,4E-05 | 3,99998934185984 | 1,1E-05 |
| 5 | 5,00001421087878 | -1,4E-05 | 4,99998934185984 | 1,1E-05 |
| 6 | 6,00001421087878 | -1,4E-05 | 5,99998934185984 | 1,1E-05 |
| 7 | 7,00001421087878 | -1,4E-05 | 6,99998934185984 | 1,1E-05 |
| 8 | 8,00001421087878 | -1,4E-05 | 7,99998934185984 | 1,1E-05 |
| 9 | 9,00001421087878 | -1,4E-05 | 8,99998934185984 | 1,1E-05 |
| 10 | 10,0000142108787 | -1,4E-05 | 9,99998934185984 | 1,1E-05 |
| 10 | 1 | 0,99985789398892 | 1,4E-04 | 0,999946709772541 | 5,3E-05 |
| 2 | 1,99985789398891 | 1,4E-04 | 1,99994670977254 | 5,3E-05 |
| 3 | 2,99985789398891 | 1,4E-04 | 2,99994670977254 | 5,3E-05 |
| 4 | 3,99985789398891 | 1,4E-04 | 3,99994670977254 | 5,3E-05 |
| 5 | 4,99985789398891 | 1,4E-04 | 4,99994670977254 | 5,3E-05 |
| 6 | 5,99985789398891 | 1,4E-04 | 5,99994670977254 | 5,3E-05 |
| 7 | 6,99985789398891 | 1,4E-04 | 6,99994670977254 | 5,3E-05 |
| 8 | 7,99985789398891 | 1,4E-04 | 7,99994670977254 | 5,3E-05 |
| 9 | 8,99985789398891 | 1,4E-04 | 8,99994670977254 | 5,3E-05 |
| 10 | 9,99985789398891 | 1,4E-04 | 9,99994670977253 | 5,3E-05 |
| 11 | 1 | 0,99822411649796 | 1,8E-03 | 1,00044412861964 | -4,4E-04 |
| 2 | 1,99822411649795 | 1,8E-03 | 2,00044412861964 | -4,4E-04 |
| 3 | 2,99822411649795 | 1,8E-03 | 3,00044412861964 | -4,4E-04 |
| 4 | 3,99822411649795 | 1,8E-03 | 4,00044412861965 | -4,4E-04 |
| 5 | 4,99822411649795 | 1,8E-03 | 5,00044412861964 | -4,4E-04 |
| 6 | 5,99822411649795 | 1,8E-03 | 6,00044412861964 | -4,4E-04 |
| 7 | 6,99822411649795 | 1,8E-03 | 7,00044412861964 | -4,4E-04 |
| 8 | 7,99822411649795 | 1,8E-03 | 8,00044412861965 | -4,4E-04 |
| 9 | 8,99822411649795 | 1,8E-03 | 9,00044412861965 | -4,4E-04 |
| 10 | 9,99822411649795 | 1,8E-03 | 10,0004441286196 | -4,4E-04 |
|  |  |  |  |  |  |
| 12 | 1 | 0,99115826702034 | 8,8E-03 | 1,00622222222222 | -6,2E-03 |
| 2 | 1,99115826702033 | 8,8E-03 | 2,00622222222222 | -6,2E-03 |
| 3 | 2,99115826702033 | 8,8E-03 | 3,00622222222222 | -6,2E-03 |
| 4 | 3,99115826702033 | 8,8E-03 | 4,00622222222222 | -6,2E-03 |
| 5 | 4,99115826702033 | 8,8E-03 | 5,00622222222222 | -6,2E-03 |
| 6 | 5,99115826702033 | 8,8E-03 | 6,00622222222222 | -6,2E-03 |
| 7 | 6,99115826702033 | 8,8E-03 | 7,00622222222222 | -6,2E-03 |
| 8 | 7,99115826702033 | 8,8E-03 | 8,00622222222222 | -6,2E-03 |
| 9 | 8,99115826702033 | 8,8E-03 | 9,00622222222222 | -6,2E-03 |
| 10 | 9,99115826702033 | 8,8E-03 | 10,0062222222222 | -6,2E-03 |
| 13 | 1 | 0,87826086956522 | 1,2E-01 | 1,01801801801801 | -1,8E-02 |
| 2 | 1,87826086956521 | 1,2E-01 | 2,01801801801801 | -1,8E-02 |
| 3 | 2,87826086956521 | 1,2E-01 | 3,01801801801801 | -1,8E-02 |
| 4 | 3,87826086956521 | 1,2E-01 | 4,01801801801801 | -1,8E-02 |
| 5 | 4,87826086956521 | 1,2E-01 | 5,01801801801801 | -1,8E-02 |
| 6 | 5,87826086956521 | 1,2E-01 | 6,01801801801802 | -1,8E-02 |
| 7 | 6,87826086956521 | 1,2E-01 | 7,01801801801801 | -1,8E-02 |
| 8 | 7,87826086956521 | 1,2E-01 | 8,01801801801801 | -1,8E-02 |
| 9 | 8,87826086956521 | 1,2E-01 | 9,01801801801801 | -1,8E-02 |
| 10 | 9,87826086956521 | 1,2E-01 | 10,0180180180180 | -1,8E-02 |
| 14 | 1 | - | - | 0,615384615384615 | 3,8E-01 |
| 2 | - | - | 1,61538461538461 | 3,8E-01 |
| 3 | - | - | 2,61538461538461 | 3,8E-01 |
| 4 | - | - | 3,61538461538461 | 3,8E-01 |
| 5 | - | - | 4,61538461538461 | 3,8E-01 |
| 6 | - | - | 5,61538461538461 | 3,8E-01 |
| 7 | - | - | 6,61538461538461 | 3,8E-01 |
| 8 | - | - | 7,61538461538461 | 3,8E-01 |
| 9 | - | - | 8,61538461538461 | 3,8E-01 |
| 10 | - | - | 9,61538461538461 | 3,8E-01 |

**Выводы:** погрешность метода LLt и метода Гаусса практически совпадают. При этом метод Гаусса смог посчитать матрицу при k = 14.

# Сравнение числа шагов на плотных матрицах в различных методах

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Метод | L | D | U | Прямой ход | Обратный ход | Всего |
| LLt |  |  | – |  |  |  |
| Метод Гаусса |  | | | – |  |  |
| LU (sq) |  |  |  |  |  |  |

**Выводы:** метод LLt выигрывает в 2 раза по количеству действий на больших плотных матрицах у методов Гаусса и LU(sq). На небольших матрицах эффект будет слабо заметен. При этом метод LLt, в отличие от метода Гаусса, можно использовать только для симметричных невырожденных положительно определённых матриц, когда Гаусс будет работать на любых невырожденных матрицах.