Белорусский Государственный Университет

Факультет Прикладной Математики и Информатики

Отчет

Лабораторная работа №2

«Решение систем на основе разложения симметричных матриц»

Профессор кафедры вычислительной математики ФПМИ

Лиходед Николай Александрович

Студент 2 группы 2 курса

Сачек Илья Валерьевич

2019 год

**Постановка задачи**:

Пусть дана система линейных алгебраических уравнений *Ax=b*. Матрица системы симметрическая: *ai,j*=*aj,i***.** 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*.

При формировании матрицы *A* можно (для простоты формирования) использовать все *n*2 элементов матрицы, при программной реализации LDLT-разложения и решения систем с треугольными матрицами – только нижний треугольник матрицы *A*.

**Входные данные:**

Начальная матрица при k = 2:

23.01 -1 -1 -1 -4 -3 -2 -4 -4 0 -3

-1 1.01 -2 -1 -2 0 0 -4 -1 0 -3

-1 -2 1.01 0 -3 -1 -3 -3 -2 -2 -1

-1 -1 0 1.01 -3 0 -2 -1 -4 -1 -1

-4 -2 -3 -3 4.01 -1 0 -3 -4 -1 -2

-3 0 -1 0 -1 3.01 -1 -4 -2 -4 -1

-2 0 -3 -2 0 -1 2.01 -3 -3 -2 -3

-4 -4 -3 -1 -3 -4 -3 4.01 -2 -3 -4

-4 -1 -2 -4 -4 -2 -3 -2 4.01 -4 -3

0 0 -2 -1 -1 -4 -2 -3 -4 0.01 0

-3 -3 -1 -1 -2 -1 -3 -4 -3 0 3.01

Начальная матрица при k = 0:

24 -1 -1 -1 -4 -3 -2 -4 -4 0 -3

-1 2 -2 -1 -2 0 0 -4 -1 0 -3

-1 -2 2 0 -3 -1 -3 -3 -2 -2 -1

-1 -1 0 2 -3 0 -2 -1 -4 -1 -1

-4 -2 -3 -3 5 -1 0 -3 -4 -1 -2

-3 0 -1 0 -1 4 -1 -4 -2 -4 -1

-2 0 -3 -2 0 -1 3 -3 -3 -2 -3

-4 -4 -3 -1 -3 -4 -3 5 -2 -3 -4

-4 -1 -2 -4 -4 -2 -3 -2 5 -4 -3

0 0 -2 -1 -1 -4 -2 -3 -4 1 0

-3 -3 -1 -1 -2 -1 -3 -4 -3 0 4

**Листинг программы:**

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

#include <fstream>

float R(float x) {

int k = 1000;

return round(x \* k) / (float)k;

}

//float R(float x) { return x; }

float CountInfelicity(std::vector<float> x\_de\_ure, std::vector<float> x\_de\_facto) {

float sum\_x2 = 0, sum\_x = 0;

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

sum\_x2 += x\_de\_facto[i] \* x\_de\_facto[i];

sum\_x += (x\_de\_facto[i] - x\_de\_ure[i]) \* (x\_de\_facto[i] - x\_de\_ure[i]);

}

return sum\_x / sum\_x2;

}

void LogMatrix(std::vector<std::vector<float>> matrix) {

std::ofstream fout("logs.txt", std::ios::app);

for (const auto& vec : matrix) {

for (auto& el : vec) {

fout << el << '\t';

}

fout << std::endl;

}

fout << std::endl << std::endl;

fout.close();

}

void LogVector(std::vector<float> vector) {

std::ofstream fout("logs.txt", std::ios::app);

for (const auto& el : vector) {

fout << el << '\t';

}

fout << std::endl << std::endl;

fout.close();

}

std::vector<std::vector<float>> MultMatrix(std::vector<std::vector<float>> a, std::vector<std::vector<float>> b\_) {

std::vector<std::vector<float>> c(a.size());

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

c[i].resize(b\_[i].size());

for (int j = 0; j < b\_[i].size(); j++) {

c[i][j] = 0;

for (int k = 0; k < a[i].size(); k++)

c[i][j] += a[i][k] \* b\_[k][j];

}

}

return c;

}

void PrintMatrix(const std::vector<std::vector<float>>& matrix) {

for (const std::vector<float>& vec : matrix) {

for (const float& x\_ : vec) {

std::cout << std::setw(7) << std::setprecision(2) << std::left << x\_ << " ";

}

std::cout << std::endl;

}

}

class Matrix {

public:

~Matrix() {

matrix\_.clear();

}

// создание матрицы А

Matrix() {

srand(time(NULL));

// случайное количество элементов матрицы

int n = rand() % 3 + 10;

// запоминаем числа для не диагональных элементов от -4 до 0

std::vector<int> no\_diag;

for (int i = 0; i < (n \* n - n) / 2; i++) {

no\_diag.push\_back(rand() % 5 - 4);

}

matrix\_.resize(n);

for (auto& line : matrix\_) {

line.resize(n, -100);

}

// заполняем матрицу А

int p = 0, sum\_1\_line = 0;

float sum\_1\_col = 0.0f;

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

for (int j = 0; j < matrix\_.size(); j++) {

if (i != j) {

if (matrix\_[i][j] == -100) {

matrix\_[i][j] = no\_diag[p];

matrix\_[j][i] = no\_diag[p];

p++;

}

}

if (i == 0 && j != 0) {

sum\_1\_line += matrix\_[0][j];

}

if (j == 0 && i != 0) {

sum\_1\_col += matrix\_[i][0];

}

}

}

// переписываем матрицу А с к = 0

matrix\_k\_0\_ = matrix\_;

float sum\_1\_col\_k\_0 = (float)sum\_1\_line;

matrix\_k\_0\_[0][0] = -(float)sum\_1\_line + 1;

sum\_1\_col\_k\_0 += matrix\_k\_0\_[0][0];

matrix\_[0][0] = -(float)sum\_1\_line + pow(10, -2);

sum\_1\_col += matrix\_[0][0];

for (int i = 1; i < matrix\_.size(); i++) {

matrix\_k\_0\_[i][i] = sum\_1\_col\_k\_0 - matrix\_k\_0\_[i][0];

matrix\_[i][i] = sum\_1\_col - matrix\_[i][0];

}

int m = 17;

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

x\_.push\_back(m + i);

}

// с помощью умножения матриц находим вектора b

b\_ = MultMat(x\_, matrix\_);

b\_k\_0\_ = MultMat(x\_, matrix\_k\_0\_);

// пишем исходные матрицы в файл

LogMatrix(matrix\_);

LogMatrix(matrix\_k\_0\_);

}

// функция умножения матриц

std::vector<float> MultMat(std::vector<float> a, std::vector<std::vector<float>> matrix) {

std::vector<float> c(matrix.size());

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

c[i] = 0;

for (int k = 0; k < matrix.size(); k++) {

c[i] += matrix[i][k] \* a[k];

}

}

return c;

}

// функция решения

std::pair<std::vector<float>, std::vector<float>> Solve() {

int line\_length = matrix\_.size();

// заполянем матрицу гаусса

std::vector<std::vector<float>> matrix\_transform(line\_length);

std::vector<std::vector<float>> matrix\_transform\_k\_0(line\_length);

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

matrix\_transform[i].resize(line\_length + 1);

matrix\_transform\_k\_0[i].resize(line\_length + 1);

for (int j = 0; j < line\_length + 1; j++) {

if (j != line\_length) {

matrix\_transform[i][j] = matrix\_[i][j];

matrix\_transform\_k\_0[i][j] = matrix\_k\_0\_[i][j];

}

else {

matrix\_transform[i][j] = b\_[i];

matrix\_transform\_k\_0[i][j] = b\_k\_0\_[i];

}

}

}

// прямой ход гаусса

D\_.resize(line\_length);

D\_k\_0\_.resize(line\_length);

for (int k = 0; k < line\_length; k++) {

D\_[k].resize(line\_length);

D\_[k][k] = matrix\_transform[k][k];

D\_k\_0\_[k].resize(line\_length);

D\_k\_0\_[k][k] = matrix\_transform\_k\_0[k][k];

for (int i = k + 1; i < line\_length; i++) {

float t\_k\_0 = R(matrix\_transform\_k\_0[i][k] / matrix\_transform\_k\_0[k][k]);

matrix\_transform\_k\_0[i][k] = t\_k\_0;

float t = R(matrix\_transform[i][k] / matrix\_transform[k][k]);

matrix\_transform[i][k] = t;

for (int j = k + 1; j < line\_length; ++j) {

matrix\_transform[i][j] -= R(matrix\_transform[k][j] \* t);

matrix\_transform\_k\_0[i][j] -= R(matrix\_transform\_k\_0[k][j] \* t\_k\_0);

}

}

}

// заполняем L из D

std::vector<std::vector<float>> Lt(matrix\_.size()), Lt\_k\_0(matrix\_k\_0\_.size());

L\_.resize(matrix\_.size());

L\_k\_0\_.resize(matrix\_.size());

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

for (int j = 0; j < line\_length; ++j) {

if (i > j) {

L\_[i].push\_back(matrix\_transform[i][j]);

L\_k\_0\_[i].push\_back(matrix\_transform\_k\_0[i][j]);

} else {

if (i == j) {

L\_[i].push\_back(1.0f);

L\_k\_0\_[i].push\_back(1.0f);

} else {

L\_[i].push\_back(0.0f);

L\_k\_0\_[i].push\_back(0.0f);

}

}

}

}

// транспонируем матрицу L

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

Lt[i].resize(line\_length, 0);

Lt\_k\_0[i].resize(line\_length, 0);

for (int j = 0; j < line\_length; ++j) {

if (i < j) {

Lt[i][j] = L\_[j][i];

Lt\_k\_0[i][j] = L\_k\_0\_[j][i];

} else {

if (i == j) {

Lt[i][j] = 1;

Lt\_k\_0[i][j] = 1;

}

}

}

}

// выводим в консоль L

std::cout << "Matrixs L\_ and Lt with k = 2:";

PrintMatrix(L\_); PrintMatrix(Lt);

std::cout << std::endl << "Matrixs L\_ and Lt with k = 0:";

PrintMatrix(L\_k\_0\_); PrintMatrix(Lt\_k\_0);

// обратный ход гаусса

// решаем уравнение Ly = b

std::vector<float> y(L\_.size()), y\_k\_0(L\_k\_0\_.size());

for (int i = 0; i < L\_.size(); i++) {

y[i] = b\_[i];

y\_k\_0[i] = b\_k\_0\_[i];

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

y[i] -= R(L\_[i][j] \* y[j]);

y\_k\_0[i] -= R(L\_k\_0\_[i][j] \* y\_k\_0[j]);

}

}

// решаем уравнение Dz = y

for (int i = 0; i < L\_.size(); i++) {

y[i] /= R(D\_[i][i]);

y\_k\_0[i] /= R(D\_k\_0\_[i][i]);

}

// решаем уравнение Ltx = z

std::vector<float> answer(y.size()), answer\_k\_0(y\_k\_0.size());

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

answer[i] = y[i];

answer\_k\_0[i] = y\_k\_0[i];

for (int j = i + 1; j < L\_.size(); j++) {

answer[i] -= R(Lt[i][j] \* answer[j]);

answer\_k\_0[i] -= R(Lt\_k\_0[i][j] \* answer\_k\_0[j]);

}

}

// пишем в файл ответы

LogVector(answer);

LogVector(answer\_k\_0);

LogMatrix(L\_);

LogMatrix(L\_k\_0\_);

std::vector<float> d, d\_k\_0;

for (int i = 0; i < D\_.size(); i++) {

d.push\_back(D\_[i][i]);

d\_k\_0.push\_back(D\_k\_0\_[i][i]);

}

LogVector(d);

LogVector(d\_k\_0);

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

for (int j = 0; j < line\_length; ++j) {

std::cout << std::setw(7) << std::setprecision(2) << std::left << matrix\_transform[i][j] << " ";

}

std::cout << " " << matrix\_transform[i][line\_length] << std::endl;

}

// считаем погрешности

std::ofstream fout("logs.txt", std::ios::app);

fout << "k = 2:" << CountInfelicity(x\_, answer) << std::endl << "k = 0:" << CountInfelicity(x\_, answer\_k\_0);

fout.close();

return std::pair<std::vector<float>, std::vector<float>> {answer, answer\_k\_0};

}

private:

std::vector<std::vector<float>> matrix\_, L\_, D\_;

std::vector<std::vector<float>> matrix\_k\_0\_, L\_k\_0\_, D\_k\_0\_;

std::vector<float> b\_, x\_, b\_k\_0\_;

};

int main() {

remove("logs.txt");

Matrix m;

std::cout << std::endl;

std::pair<std::vector<float>, std::vector<float>> pair = m.Solve();

std::cout << std::endl << "Vector with k = -2" << std::endl;

for (const auto& el : pair.first) {

std::cout << std::setw(7) << std::setprecision(7) << el << " ";

}

std::cout << std::endl << "Vector with k = 0" << std::endl;

for (const auto& el : pair.second) {

std::cout << std::setw(7) << std::setprecision(5) << el << " ";

}

return 0;

}

**Выходные данные:**

Матрица L при k = 2:

1 0 0 0 0 0 0 0 0 0 0

-0.043 1 0 0 0 0 0 0 0 0 0

-0.043 -2.113 1 0 0 0 0 0 0 0 0

-0.043 -1.079 0.671 1 0 0 0 0 0 0 0

-0.174 -2.248 2.319 -0.228 1 0 0 0 0 0 0

-0.13 -0.134 0.419 0.497 0.097 1 0 0 0 0 0

-0.087 -0.09 0.976 0.009 0.43 -0.221 1 0 0 0 0

-0.174 -4.316 3.58 1.754 0.932 -1.008 0.428 1 0 0 0

-0.174 -1.214 1.389 -1.718 0.178 0.052 -0.06 1.542 1 0 0

0 0 0.597 0.253 0.227 -1.367 -1.324 -0.38 -0.22 1 0

-0.13 -3.237 2.311 0.509 0.522 0.095 0.209 0.088 -0.178 -0.485 1

Матрица L при k = 0:

1 0 0 0 0 0 0 0 0 0 0

-0.042 1 0 0 0 0 0 0 0 0 0

-0.042 -1.043 1 0 0 0 0 0 0 0 0

-0.042 -0.532 6.558 1 0 0 0 0 0 0 0

-0.167 -1.107 31.552 3.554 1 0 0 0 0 0 0

-0.125 -0.064 7.302 0.914 0.152 1 0 0 0 0 0

-0.083 -0.042 18.424 2.12 0.536 -0.051 1 0 0 0 0

-0.167 -2.128 43.674 5.214 1.059 -0.452 0.602 1 0 0 0

-0.167 -0.596 19.674 1.978 0.628 0.08 0.313 0.802 1 0 0

0 0 11.628 1.377 0.307 -0.838 -0.322 0.018 -0.039 1 0

-0.125 -1.596 25.488 2.95 0.647 0.156 0.228 0.178 0.78 -2.513 1

Матрица D при k = 2:

Diag { 23.01, 0.967, -3.35, 1.35, 16.359, 2.702, 1.86, 6.766, -12.241, -8.418, 7.772}

Матрица D при k = 0:

Diag { 24, 1.958, -0.172, 8.808, 62.052, 4.008, 3.828, 12.542, 2.859, -1.495, 14.878}

Ответ при k = 2:

16.9815 17.7764 18.5194 20.0155 21.1717 22.1508 22.7417 24.0626 25.0008 25.9512 27.0294

Ответ при k = 0:

17.2667 21.0618 22.1862 20.7287 21.0574 21.1407 23.529 23.6732 25.3676 25.7707 26.8786

Относительные погрешности:

При k = 0: 0.3766%, при k = 2: 0.0075328%.

**Вывод:**

Данный алгоритм решает СЛАУ. Кроме того, он позволяет выполнить это с достаточно малой погрешностью, т.е. наше решение отличается от точного совсем на маленькие значения.