Белорусский Государственный Университет

Факультет Прикладной Математики и Информатики

Отчет

Лабораторная работа №4

«Итерационные метод Данилевского»

Профессор кафедры вычислительной математики ФПМИ

Лиходед Николай Александрович

Студент 2 группы 2 курса

Сачек Илья Валерьевич

2019 год

**Постановка задачи**:

Необходимый для выполнения работы теоретический материал и формулы имеются в файле «Метод Данилевского».

**Задание.** Разработать программу приведения матрицы к канонической форме Фробениуса методом Данилевского (регулярный случай); сохранить матрицы, используемые для получения собственных векторов.

Для вычислений:

* *n=*4 – порядок матрицы;
* при заполнении матрицы *A* использовать случайные числа из диапазона от −50 до 50. Для получения случайных чисел использовать библиотечную функцию rand(), подключив хедер stdlib.h;
* использовать тип float;
* «ведущий элемент» ** на (*n*–*k*+1)-м шаге, *k=n*, *n*–1,... , 2, (*=*) должен быть отличным от нуля (рассматривается только такой регулярный случай);
* считать ** равным нулю, если ;
* если , то выдать соответствующее сообщение и заново заполнить матрицу *A*.

Программно реализовать для рассматриваемого примера алгоритм приведения матрицы к канонической форме Фробениуса. Порядок матрицы *n* должен быть в коде параметром (это требование не является обязательным). Сохранить матрицы *Mn–*1, *Mn–*2,... , *M*1, используемые для получения собственных векторов (понадобятся для другой работы). Вывести на печать матрицу *A* (входные данные), полученную каноническую форму Фробениуса Φ, матрицы *Mn–*1, *Mn–*2, ... , *M*1, коэффициент *p*1 (полученный из формы Фробениуса), след матрицы Sp *A* (для контроля вычислений: должно приближенно выполняться равенство *p*1*=*Sp *A=a*1,1+*a*2,2+... +*an,n*).

**Входные данные:**

Начальная матрица:

19 43 -29 21

-46 49 5 41

-48 21 40 25

44 47 -46 -31

**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 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]);

}

std::ofstream fout("errors.txt");

fout << sum\_x / sum\_x2 << std::endl;

fout.close();

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 auto& vec : matrix) {

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

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

}

std::cout << std::endl;

}

}

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

for (const auto& x\_ : vector) {

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

}

std::cout << std::endl;

}

class Matrix {

public:

// 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;

}

~Matrix() {

matrix\_.clear();

}

// создание матрицы А

Matrix() {

srand(time(NULL));

// случайное количество элементов матрицы

int n = 4;// rand() % 3 + 10;

matrix\_.resize(n, std::vector<float>(n));

for (auto& line : matrix\_) {

line.resize(n);

}

// заполняем матрицу А

// заполнение недиагональных элементов

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

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

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

matrix\_[i][j] = rand() % 100 - 50;

}

}

// пишем исходную матрицу в файл

LogMatrix(matrix\_);

PrintMatrix(matrix\_);

std::cout << std::endl;

}

void ImplementDanilevski() {

std::vector<std::vector<float>> M\_i, M\_i\_1;

while (true) {

for (int i = matrix\_.size() - 2; i >= 0; i--) {

std::cout << i << std::endl;

if (i == 0) {

int sfgvds = 1010234;

}

M\_i = CreateM\_i(i);

std::cout << std::endl;

PrintMatrix(M\_i);

LogMatrix(M\_i);

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

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

fout.close();

M\_i\_1 = CreateM\_i\_1(i);

std::cout << std::endl;

PrintMatrix(M\_i\_1);

LogMatrix(M\_i\_1);

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

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

fout.close();

matrix\_ = MultMatrix(matrix\_, M\_i);

matrix\_ = MultMatrix(M\_i\_1, matrix\_);

PrintMatrix(matrix\_);

LogMatrix(matrix\_);

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

fout << std::endl << std::endl << "----------------------------------";

fout.close();

if (i != 0) {

if (matrix\_[i][i - 1] == 0) {

continue;

}

}

}

break;

}

//PrintMatrix(matrix\_);

}

std::vector<std::vector<float>> CreateM\_i(int n) {

std::vector<std::vector<float>> M\_i(matrix\_.size(), std::vector<float>(matrix\_.size()));

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

if (i != n) {

M\_i[i][i] = 1;

}

if (i == n) {

M\_i[n][i] = 1 / matrix\_[n + 1][n];

} else {

M\_i[n][i] = -matrix\_[n + 1][i] / matrix\_[n + 1][n];

}

}

return M\_i;

}

std::vector<std::vector<float>> CreateM\_i\_1(int n) {

std::vector<std::vector<float>> M\_i\_1(matrix\_.size(), std::vector<float>(matrix\_.size()));

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

if (i != n) {

M\_i\_1[i][i] = 1;

}

M\_i\_1[n][i] = matrix\_[n + 1][i];

}

return M\_i\_1;

}

private:

std::vector<std::vector<float>> matrix\_;

std::vector<float> f\_, x\_, x\_curr\_;

int k\_max\_, k\_;

float E\_;

};

int main() {

remove("logs.txt");

Matrix m;

std::cout << std::endl;

m.ImplementDanilevski();

return 0;

}

**Выходные данные:**

Mn-1:

1 0 0 0

0 1 0 0

0.375 0.0833333 -0.0208333 0.979167

0 0 0 1

Mn-2:

1 0 0 0

-0.950182 -0.00047521 0.0157215 -0.33831

0 0 1 0

0 0 0 1

Mn-3:

6.81348e-06 -0.000220529 0.0248235 0.0974955

0 1 0 0

0 0 1 0

0 0 0 1

След A: 24

Форма Фробениуса:

24 -3326 51516 2.5062 8e+06

1 0 0 0

0 1 0 0

0 0 1 0

Собственные значения:

24 -3326 51516 2.5062 8e+06