Отчет по дисциплине: «Численные методы»

Лабораторная работа №7

## «Метод Данилевского»

Подготовил студент 3 курса 4 группы

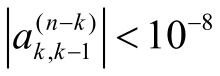
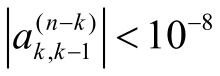
Кондратович Артём

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

Цель работы. Изучить основы метода Данилевского., разработать программу, реализующую основной случай метода.

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;
* 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*).

Листинг программы:

#include <iostream>

#include <random>

#include <vector>

#include <map>

#include "Operators.h"

#include <string>

const int n = 4;

const float epsilon = (float)std::pow(10, -8);

float GetRandomFromRange(int a, int b)

{

static std::random\_device rd;

static std::mt19937 generator(rd());

std::uniform\_real\_distribution<float> distribution(a, b);

return distribution(generator);

}

std::vector<std::vector<float>> GenerateE()

{

std::vector<std::vector<float>> e(n, std::vector<float>(n));

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

{

e[i][i] = 1;

}

return e;

}

std::vector<std::vector<float>> GenerateA()

{

std::vector<std::vector<float>> matrix(n, std::vector<float>(n));

for(auto& line : matrix)

{

for (auto& element : line)

{

element = GetRandomFromRange(-50, 50);

}

}

return matrix;

}

std::vector<std::vector<float>> MatrixM(const std::vector<std::vector<float>>& a, float leading, int i)

{

auto e = GenerateE();

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

{

if (j != i - 1)

{

e[i - 1][j] = -(a[i][j] / leading);

}

else

{

e[i - 1][j] = 1 / leading;

}

}

return e;

}

std::vector<std::vector<float>> InverseMatrixM(const std::vector<std::vector<float>>& a, int i)

{

auto e = GenerateE();

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

{

e[i - 1][j] = a[i][j];

}

return e;

}

float Trace(const std::vector<std::vector<float>>& matrix)

{

float trace = 0;

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

{

trace += matrix[i][i];

}

return trace;

}

bool DanilevskiMethod(std::vector<std::vector<float>> a, std::map<std::string, std::vector<std::vector<float>>>& matrices, std::vector<std::vector<float>>& frobenius)

{

for (int i = n - 1; i >= 1; i--)

{

if (std::fabs(a[i][i - 1]) < epsilon)

{

return false;

}

//std::cout << std::endl << a[i][i - 1] << " " << i << std::endl;

auto m = MatrixM(a, a[i][i - 1], i);

//std::cout << "M" << std::endl << m;

matrices.insert({ "M" + std::to\_string(i), m});

auto m1 = InverseMatrixM(a, i);

//std::cout << "M^-1" << std::endl << m1;

a = m1 \* a \* m;

}

frobenius = a;

return true;

}

float CheckResult(float sourceTrace, float resultTrace)

{

return std::fabs(sourceTrace - resultTrace);

}

void Printreport(const std::vector<std::vector<float>>& frobenius, const std::map<std::string, std::vector<std::vector<float>>>& matrices, const float& sourceTrace)

{

std::cout << "Frobenius" << std::endl;

std::cout << frobenius;

for (auto const& m : matrices)

{

std::cout << m.first << std::endl;

std::cout << m.second;

}

std::cout << "p1 = " << frobenius[0][0] << std::endl;

std::cout << "|sourceTrace - p1| = " << CheckResult(sourceTrace, frobenius[0][0]);

}

void Solve()

{

std::vector<std::vector<float>> frobenius(n, std::vector<float>(n));

std::map<std::string, std::vector<std::vector<float>>> matrices;

while (true)

{

auto a = GenerateA();

std::cout << "Start" << std::endl;

std::cout << "Source matrix A" << std::endl;

std::cout << a;

auto trace = Trace(a);

if (DanilevskiMethod(a, matrices, frobenius))

{

Printreport(frobenius, matrices, trace);

break;

}

std::cout << "Exception! One of the leading elements less than 10^-8" << std::endl;

if (!matrices.empty())

{

matrices.clear();

}

}

}

int main()

{

Solve();

}

Результат:

