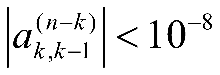
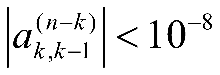
**Лабораторная работа 6**

**«Метод Данилевского»**

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*).

Содержание работы должно включать следующие пункты.

1. Постановка задачи.
2. Входные данные.
3. Листинг программы.
4. В листинге программы обязательны подробные комментарии.
5. Выходные данные.

**Входные данные**

n=4;

**Листинг программы**

#include <iostream>

#include <iomanip>

#include <random>

#include <fcntl.h>

#include <io.h>

using namespace std;

//Размерность матрицы

int n = 4;

//Нахождение случайного числа

int Rand(int L, int R) {

static random\_device rd;

static mt19937 gen(rd());

uniform\_int\_distribution<> dis(L, R);

return dis(gen);

}

//Вывод матрицы

void PrintMatrix(float\*\* A)

{

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

{

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

{

cout << fixed << setprecision(8) << setw(20) << A[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

//Возврат знака

inline wchar\_t sign(float x) {

if (x >= 0)

return L'+';

return L'-';

}

//Метод Данилевского

void Danilevsky(float\*\* A) {

float\*\* M = new float\* [n];

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

M[i] = new float[n];

}

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

if (fabs(A[i][i - 1]) < 0.00000001f) {

cout << "Bad matrix";

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

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

A[i][j] = Rand(-50, 50);

}

}

Danilevsky(A);

return;

}

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

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

if (j != i - 1) {

if (j == k) {

M[j][k] = 1;

} else {

M[j][k] = 0;

}

}

}

}

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

if (j == i - 1) {

M[i - 1][j] = 1 / A[i][i - 1];

} else {

M[i - 1][j] = -A[i][j] / A[i][i - 1];

}

}

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

if (j != i) {

A[j][i - 1] /= A[i][i - 1];

}

}

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

if (j != i - 1) {

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

if (k != i) {

A[k][j] -= A[k][i - 1] \* A[i][j];

}

}

}

}

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

float s = 0;

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

if (k != i) {

s += A[k][j] \* A[i][k];

}

}

A[i - 1][j] = s;

}

A[i - 1][i - 1] += A[i][i];

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

if (j != i - 1) {

A[i][j] = 0;

} else {

A[i][j] = 1;

}

}

cout << "M" << i << ":" << endl;

PrintMatrix(M);

}

}

int main() {

//Создание и инициализация матрицы

float\*\* A = new float\* [n];

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

A[i] = new float[n];

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

A[i][j] = Rand(-50, 50);

}

}

//Вывод матрицы

cout << "A:" << endl;

PrintMatrix(A);

//Расчет главного значения матрицы

float SpA = 0;

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

SpA += A[i][i];

}

//Метод Данилевского

Danilevsy(A);

//Вывод

cout << "p1:" << " " << A[0][0] << endl;

cout << "SpA:" << " " << SpA << endl;

cout << "Characteristic polynomial: ";

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

wcout << setprecision(8);

wcout << (wchar\_t)0x03BB << (wchar\_t)0x2074 << L' ' << sign(-A[0][0]) << L' ' << fabs(A[0][0]) << (wchar\_t)0x00B7 << (wchar\_t)0x03BB <<

(wchar\_t)0x00B3 << L' ' << sign(-A[0][1]) << L' ' << fabs(A[0][1]) << (wchar\_t)0x00B7 << (wchar\_t)0x03BB << (wchar\_t)0x00B2 << L' ' << sign(-A[0][2]) << L' ' << fabs(A[0][2])

<< (wchar\_t)0x00B7 << (wchar\_t)0x03BB << L' ' << sign(-A[0][3]) << L' ' << fabs(A[0][3]);

}

**Выходные данные**

