1. **Постановка Задачи.**

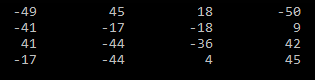
Разработать программу приведения матрицы к канонической форме Фробениуса методом Данилевского (регулярный случай); сохранить матрицы, используемые для получения собственных векторов.

Для вычислений:

* *n=*4 – порядок матрицы;
* при заполнении матрицы *A* использовать случайные числа из диапазона от −100 до 100. 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*).

1. **Входные данные**



1. **Листинг программы**

Файл Frobenius.h:

#pragma once

#include "Matrix.h"

#include <ctime>

using namespace std;

class frobenius

{

public:

frobenius(size\_t n) : n(n)

{

generate();

}

public:

void generate()

{

srand(time(nullptr));

A.resize(n);

for (auto& r : A)

r.resize(n);

for (auto& r : A)

for (auto& el : r)

el = rand() % 100 - 50;

M.resize(n - 1);

for (auto& mat : M)

{

mat.resize(n);

for (auto& r : mat)

r.resize(n, 0);

}

for (auto& mat : M)

{

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

mat[i][i] = 1;

}

M\_inv.resize(n - 1);

for (auto& mat : M\_inv)

{

mat.resize(n);

for (auto& r : mat)

r.resize(n, 0);

}

for (auto& mat : M\_inv)

{

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

mat[i][i] = 1;

}

}

bool norm()

{

for (int k = n - 2; k >= 0; --k)

{

float cur = A[k+1][k];

if (abs(cur) < 1.e-8)

return false;

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

M[k][k][i] = -A[k+1][i] / cur;

M[k][k][k] = 1 / cur;

for (int i = k+1; i < n; ++i)

M[k][k][i] = -A[k + 1][i] / cur;

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

M\_inv[k][k][i] = A[k+1][i];

A = M\_inv[k] \* A;

A = A \* M[k];

}

return true;

}

vector<vector<float>>& getMatrix()

{

return A;

}

void printM()

{

for (auto m : M)

cout << m << endl;

}

private:

vector<vector<vector<float>>> M;

vector<vector<vector<float>>> M\_inv;

vector<vector<float>> A;

size\_t n;

};

Файл Matrix.cpp:

#pragma once

#include <vector>

#include <iostream>

#include <iomanip>

using namespace std;

vector<vector<float>> operator\*(const vector<vector<float>>& A, const vector<vector<float>>& B) // for square matrices ONLY!

{

size\_t n = A.size();

vector <vector<float>> ans(n);

for (auto& r : ans)

r.resize(n);

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

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

{

float temp = 0;

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

temp += A[i][k] \* B[k][j];

ans[i][j] = temp;

}

return ans;

}

ostream& operator<<(ostream& os, const vector<vector<float>>& M)

{

for (auto r : M)

{

for (auto el : r)

os << setw(10) << el << ' ';

os << endl;

}

return os;

}

float trace(const vector<vector<float>>& M)

{

float tmp = 0;

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

tmp += M[i][i];

return tmp;

}

Файл main.cpp:

#include "Frobenius.h"

#include "Matrix.h"

#include <iostream>

using namespace std;

const int n = 4;

int main()

{

frobenius fb(n);

cout << setprecision(4) << fb.getMatrix() << endl;

cout << "Trace: " << trace(fb.getMatrix()) << "\n\n";

bool flag = false;

while (!fb.norm())

{

cout << "Regenerating matrix\n";

fb.generate();

}

cout <<"Frobenius normal form: \n" << setprecision(4) << fb.getMatrix()<<endl;

cout << "p1 = " << fb.getMatrix()[0][0];

cout << "M matrices:\n";

fb.printM();

system("pause");

}

1. **Выходные данные**

