Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Томский государственный университет систем управления и радиоэлектроники»

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

**Вычисление собственных чисел и собственных векторов**

**Отчет по лабораторной работе №3**

**По дисциплине**

**«Численные методы»**

Обучающийся гр. 431-3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Андреев Д.П.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Руководитель: ст. Преподаватель каф. АСУ

Косова А. Е.

« » 2023 г.

Томск 2023

Оглавление

[1 Цель и Задачи 3](#_Toc137165734)

[2 Теория 4](#_Toc137165735)

[3.Листинг программы 8](#_Toc137165736)

[4.Пример решения 14](#_Toc137165737)

[5.Вывод 16](#_Toc137165738)

# 1 Цель и Задачи

В данной практической работе необходимо найти матрицу Фробениуса, собственные числа и вектора входной матрицы.

**2 Теория**

При аналитическом решении собственные числа матрицы находятся из решения уравнения

D(λ) = 0, (2.3.4)

где D(λ) = det(A – λE) – характеристический полином матрицы. После этого, согласно (2.3.1), можно найти собственные вектора, решая СЛАУ

(A – λE)x = 0. (2.3.5)

**Вычисление собственных чисел методом Данилевского**

В данной практической работе для поиска собственных чисел и векторов мы будем использовать метод Данилевского. Суть его состоит в том, что исходная матрица A преобразуется в подобную ей матрицу Фробениуса P, имеющую следующий вид:

Делается это при помощи следующего преобразования подобия:

(2.3.6)

где

Таким образом, можно последовательно находить n–1 матрицу :

(2.3.7)

А можно найти матрицы S (прямую и обратную) и затем сразу вычислить P по формуле (2.3.6). Такой способ эффективнее, т.к. не нужно хранить множество матриц M, произведение которых еще понадобятся для вычисления собственных векторов.

Матрицы M строятся следующим образом:

(2.3.8)

(2.3.9)

Несложно доказать, что у подобных матриц собственные числа совпадают. Далее для матрицы P строится характеристический полином

D(λ) = det(P – λE) = . (2.3.10)

Это полином степени n. Очевидно, что он имеет n корней . Некоторые из них могут быть кратными, при этом выполняется соотношение (2.3.2). Необходимо не только найти все корни полинома, но и определить их кратность (см. п. 2.3.1.3).

**Вычисление собственных векторов методом Данилевского**

Далее для каждого собственного числа вычисляется соответствующий ему собственный вектор. Собственные вектора у подобных матриц не совпадают. Если yi – это собственный вектор матрицы P, соответствующий собственному числу λi , то

. (2.3.11)

При этом собственный вектор матрицы P выглядит следующим образом:

(2.3.12)

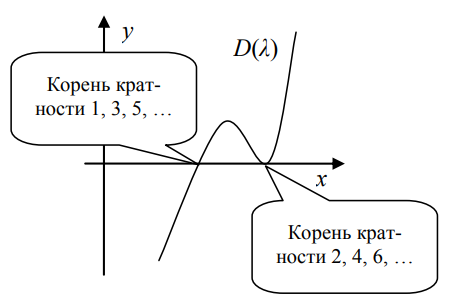
Определение кратности собственных числе и векторов

При поиске кратных корней возникают некоторые сложности. Дело в том, что если кратность корня четная, то в этой точке наблюдается экстремум (минимум или максимум) характеристического полинома, а если нечетная – то полином просто меняет знак. Пример приведен на рис. 2.3.1.

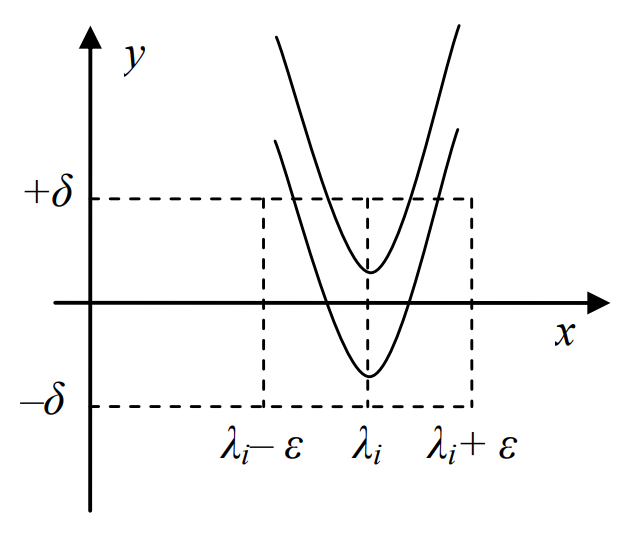
Согласно определению [1], корень уравнения ξ имеет кратность k, если не только функция в точке ξ принимает нулевое значение, но и k –1 ее производных:

. (2.3.13)

При i = 0 имеем саму функцию. Таким образом, получаем k нулей функции и ее производных.

Рисунок 2.3.1: Поведение характеристического полинома

Учитывая погрешности вычислений на ЭВМ, при четной кратности корня характеристический полином может пройти либо выше, либо ниже нулевой отметки (рис. 2.3.2).

Рисунок 2.3.2: Погрешности при вычислении собственных чисел

Здесь ε и δ – достаточно малые числа. Т.о., программа может либо вообще не найти корня, либо найти сразу два. Поэтому договоримся считать корнем любое число , для которого | f ()| < δ. При этом, если два корня расположены близко друг к другу (т.е. ), то корнем следует считать только один из них, либо за корень принять число, расположенное между ними:

. (2.3.14)

Поиск собственных чисел продолжается до тех пор, пока не будут найдены все, т.е. пока не выполнится условие (2.3.2).

# 3.Листинг программы

**Program.cs**

using System;

using System.Data;

using System.IO;

using чм\_3;

namespace чм\_3

{

class Program

{

static void Main(string[] args)

{

FileStream FileIn = new FileStream("IN.txt", FileMode.Open);

StreamReader reader = new StreamReader(FileIn);

int m = Convert.ToInt16(reader.ReadLine());

uint n = Convert.ToUInt32(reader.ReadLine());

double[] matData = new double[n\* n];

//Записываем матрицу

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

{

int ind = 0;

string nums = reader.ReadLine();

foreach (var num in nums.Split())

{

matData[i \* n + (ind++)] = Convert.ToDouble(num);

}

}

reader.Close();

Matrix taskMat = new Matrix(n, n, matData);

FileStream FileOut = new FileStream("OUT.txt", FileMode.Open);

StreamWriter writer = new StreamWriter(FileOut);

double[,] eigen = taskMat.EigenValues(in writer);

for (int i = 0; i < n; i += (int)eigen[i, 1])

{

writer.WriteLine("--------------------------------------------------------------");

writer.WriteLine("Собственное число {0}: {1}", i, eigen[i, 0]);

writer.WriteLine("Проверка собственного числа {0}: {1}", i, taskMat.CheckEigen(eigen[i, 0]));

if (m == 2)

{

Matrix vect = taskMat.EigenVector(eigen[i, 0]);

writer.WriteLine("Собственный вектор {0}: \n{1}", i, vect);

writer.WriteLine("Проверка собственного вектора {0}: \n{1}", i, taskMat.CheckEigenVect(vect, eigen[i, 0]));

}

writer.WriteLine("Кратность собственного числа {0}: {1} ", i, eigen[i, 1]);

}

Console.WriteLine("Готово!");

writer.Close();

}

}

}

**Class1.cs**

using System;

using System.Collections;

using System.Data;

using System.IO;

using System.Linq;

using System.Net.WebSockets;

using PolStrLib;

namespace чм\_3

{

class Matrix : IEnumerable, IFormattable, ICloneable

{

private double[] data;

private uint lines;

private uint columns;

private static uint newID = 0;

private uint id;

public double[] Data

{

get

{

return data;

}

private set

{

data = value;

}

}

public uint Lines

{

get

{

return lines;

}

private set

{

lines = value;

}

}

public uint Columns

{

get

{

return columns;

}

private set

{

columns = value;

}

}

//Конструктор по умолчанию

public Matrix(uint \_lines = 3, uint \_columns = 3, double[] \_data = null)

{

if ((\_lines == 0) || (\_columns == 0))

{

throw new ArgumentException("Количество строк и колонок должно быть отлично от 0");

}

if ((\_data != null) && (\_lines \* \_columns > \_data.Length))

{

throw new ArgumentException("Данных не достаточно для полного заполнения матрицы");

}

if ((\_data != null) && (\_lines \* \_columns < \_data.Length))

{

throw new ArgumentException("Выход за пределы размеров матрицы");

}

id = newID++;

lines = \_lines;

columns = \_columns;

data = new double[lines \* columns];

if (\_data != null)

{

\_data.CopyTo(data, 0);

}

else

{

for (int i = 0; i < columns \* lines; i++)

{

data[i] = 0;

}

}

}

//Конструктор квадратной матрицы

public Matrix(uint \_size, double[] \_data) : this(\_size, \_size, \_data)

{

}

//Конструктор копирования

public Matrix(Matrix \_matrix) : this(\_matrix.lines, \_matrix.columns, \_matrix.data)

{

}

//Соразмерная единичная матрица

public Matrix GetUnit()

{

double[] unit = new double[lines \* lines];

for (uint i = 0; i < lines; i++)

{

unit[i \* lines + i] = 1.0;

}

Matrix answ = new Matrix(lines, lines, unit);

return answ;

}

//Найти S

private Matrix[] S()

{

Matrix[] a = new Matrix[lines];

a[0] = this.Clone();

Matrix[,] m = new Matrix[lines - 1, 2];

for (uint i = 1; i < lines; i++)

{

m[i - 1, 0] = M(lines - i, false, a[i - 1]);

m[i - 1, 1] = M(lines - i, true, a[i - 1]);

a[i] = m[i - 1, 1] \* a[i - 1] \* m[i - 1, 0];

}

Matrix[] \_S = new Matrix[2];

\_S[0] = m[0, 0];

\_S[1] = m[lines - 2, 1];

for (uint i = 1; i < lines - 1; i++)

{

\_S[0] \*= m[i, 0];

\_S[1] \*= m[lines - 2 - i, 1];

}

return \_S;

}

//Найти M

private Matrix M(uint \_k, bool \_inv, Matrix \_a)

{

Matrix \_M = new Matrix(lines, lines);

--\_k;

if (\_inv)

{

for (uint i = 0; i < lines; i++)

{

for (uint j = 0; j < lines; j++)

{

if (i == \_k)

{

\_M[i, j] = \_a[i + 1, j];

continue;

}

if (i == j)

{

\_M[i, j] = 1;

continue;

}

\_M[i, j] = 0;

}

}

}

else

{

for (uint i = 0; i < lines; i++)

{

for (uint j = 0; j < lines; j++)

{

if ((i == \_k) && (j == \_k))

{

\_M[i, j] = 1 / \_a[\_k + 1, \_k];

continue;

}

if (i == \_k)

{

\_M[i, j] = -(\_a[\_k + 1, j] / \_a[\_k + 1, \_k]);

continue;

}

if (i == j)

{

\_M[i, j] = 1;

continue;

}

\_M[i, j] = 0;

}

}

}

return \_M;

}

//Матрица фробениуса

public Matrix Frobeniys()

{

Matrix[] s = S();

Matrix p = s[1] \* this \* s[0];

return p;

}

public double Det()

{

Matrix copy = this.Clone();

double det = 1;

for (uint i = 0; i < lines; i++)

{

double cof = copy[i, i];

det \*= cof;

for (uint j = i; j < lines; j++)

{

copy[i, j] /= cof;

}

for (uint j = i + 1; j < lines; j++)

{

cof = copy[j, i];

for (uint k = i; k < lines; k++)

{

copy[j, k] -= cof \* copy[i, k];

}

}

}

return det;

}

//Собственные числа

public double CheckEigen(double \_eigen)

{

Matrix tempMat = this - this.GetUnit() \* \_eigen;

return tempMat.Det();

}

public Matrix CheckEigenVect(Matrix \_vector, double \_eigen)

{

return (this \* \_vector) - (\_vector \* \_eigen);

}

public double[,] EigenValues(in StreamWriter \_writer = null)

{

double[] values = new double[lines];

Matrix p = this.Frobeniys();

if (\_writer != null)

{

\_writer.WriteLine("Матрица Фробениуса:");

\_writer.WriteLine(p);

}

for (uint i = 0; i < lines; i++)

{

values[i] = p[0, i];

}

return PolinRoot(values);

}

//Собственный вектор

public Matrix EigenVector(double \_eigen)

{

Matrix y = new Matrix(lines, 1);

for (uint i = 0; i < lines; i++)

{

y[i, 0] = Math.Round(Math.Pow(\_eigen, lines - i - 1),3);

}

return S()[0] \* y;

}

//Решение полинома

private static double[,] PolinRoot(double[] \_values, double \_start = -10, double \_end = 10, double \_disp = 0.001, double \_step = 0.0001)

{

string polinom;

string fx = GetPolin(\_values);

PolStr.StrToPolStr(fx, out polinom, 0);

double[,] answ = new double[\_values.Count(), 2];

bool wasFound = false;

double x0 = 0;

double f0 = PolStr.EvalPolStr(polinom, x0, 0);

f0 = \_values.Count() % 2 == 0 ? f0 : -f0;

double f1 = f0;

uint counter = 0;

for (double x1 = \_start; (counter < \_values.Count()) || (x1 < \_end); x1 += \_step)

{

f0 = f1;

f1 = PolStr.EvalPolStr(polinom, x1, 0);

f1 = \_values.Count() % 2 == 0 ? f1 : -f1;

if ((Math.Abs(f1) <= \_disp) && (Math.Abs(f1) < Math.Abs(f0)))

{

x0 = x1;

wasFound = true;

continue;

}

if (wasFound || ((f0 \* f1) < 0))

{

wasFound = false;

uint multiplicity = 1;

answ[counter, 0] = x0;

double tempDisp = \_disp;

while (Math.Abs(PolStr.EvalPolStr(polinom, x0, multiplicity)) < tempDisp)

{

++multiplicity;

tempDisp \*= 10;

}

for (uint i = 0; i < (multiplicity - 1); i++)

{

answ[counter, 1] = multiplicity;

answ[++counter, 0] = x0;

}

answ[counter++, 1] = multiplicity;

}

}

if (counter < \_values.Count())

throw new ArgumentOutOfRangeException("Какой либо из параметров препятсвует нахождению корней");

return answ;

}

//Получить полином в строке

private static string GetPolin(double[] values)

{

string f = "";

for (uint i = 0; i < values.Count() + 1; i++)

{

if (i != 0)

{

double buf = values[i - 1] \* -1;

if (!(buf < 0))

{

f += "+";

}

f += buf;

if (i != values.Count())

{

f += "\*";

}

}

for (uint j = 0; j < values.Count() - i; j++)

{

f += "x";

if (j == values.Count() - i - 1)

{

break;

}

f += "\*";

}

}

//if (values.Count() % 2 != 0)

//f += ")";

return f.Replace(',', '.');

}

//Проверка умножения

public bool CheckMul(Matrix \_operand)

{

if (\_operand == null)

{

throw new ArgumentException("Операнд не инициализрован");

}

if (columns == \_operand.lines)

{

return true;

}

return false;

}

//Проверка сложения

public bool CheckAdd(Matrix \_operand)

{

if (\_operand == null)

{

throw new ArgumentException("Операнд не инициализрован");

}

if ((columns == \_operand.columns) && (lines == \_operand.lines))

{

return true;

}

return false;

}

//Оператор умножения (матрица, матрица)

public static Matrix operator \*(Matrix \_operand1, Matrix \_operand2)

{

if (!\_operand1.CheckMul(\_operand2))

{

throw new ArgumentException("Данные матрицы нельзя перемножать");

}

double[] newData = new double[\_operand1.lines \* \_operand2.columns];

for (int i = 0; i < \_operand1.lines; i++)

{

for (int j = 0; j < \_operand2.columns; j++)

{

newData[i \* \_operand2.columns + j] = 0;

for (int k = 0; k < \_operand1.columns; k++)

{

newData[i \* \_operand2.columns + j] += (\_operand1.data[i \* \_operand1.columns + k] \* \_operand2.data[k \* \_operand2.columns + j]);

}

}

}

Matrix toOut = new Matrix(\_operand1.lines, \_operand2.columns, newData);

return toOut;

}

//Оператор умножения (матрица, скаляр)

public static Matrix operator \*(Matrix \_operand1, double \_operand2)

{

if (\_operand2 == null)

{

throw new ArgumentException("Второй операнд не инициализрован");

}

double[] newData = new double[\_operand1.lines \* \_operand1.columns];

\_operand1.data.CopyTo(newData, 0);

for (int i = 0; i < (\_operand1.columns \* \_operand1.lines); i++)

{

newData[i] \*= \_operand2;

}

Matrix toOut = new Matrix(\_operand1.lines, \_operand1.columns, newData);

return toOut;

}

//Оператор сложения (матрица, матрица)

public static Matrix operator +(Matrix \_operand1, Matrix \_operand2)

{

if (!\_operand1.CheckAdd(\_operand2))

{

throw new ArgumentException("Данные матрицы нельзя складывать");

}

double[] newData = new double[\_operand1.lines \* \_operand1.columns];

\_operand1.data.CopyTo(newData, 0);

for (int i = 0; i < \_operand1.lines \* \_operand1.columns; i++)

{

newData[i] += \_operand2.data[i];

}

Matrix toOut = new Matrix(\_operand1.lines, \_operand1.columns, newData);

return toOut;

}

//Оператор вычитания (матрица, матрица)

public static Matrix operator -(Matrix \_operand1, Matrix \_operand2)

{

if (!\_operand1.CheckAdd(\_operand2))

{

throw new ArgumentException("Данные матрицы нельзя вычитать");

}

double[] newData = new double[\_operand1.lines \* \_operand1.columns];

\_operand1.data.CopyTo(newData, 0);

for (int i = 0; i < \_operand1.lines \* \_operand1.columns; i++)

{

newData[i] -= \_operand2.data[i];

}

Matrix toOut = new Matrix(\_operand1.lines, \_operand1.columns, newData);

return toOut;

}

//Индексатор

public double this[uint i, uint j]

{

get { return data[i \* columns + j]; }

set { data[i \* columns + j] = value; }

}

//Перевод в строку

public override string ToString()

{

string toOut = "";

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

{

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

{

toOut += data[i \* columns + j].ToString();

toOut += " ";

}

toOut += "\n";

}

return toOut;

}

//Интерфейс клонирования

object ICloneable.Clone()

{

return Clone();

}

public Matrix Clone()

{

Matrix toOut = new Matrix(this.lines, this.columns, this.data);

return toOut;

}

//Интерфейс форматирования

string IFormattable.ToString(string format, IFormatProvider formatProvider)

{

string toOut = "";

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

{

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

{

toOut += data[i \* columns + j].ToString();

toOut += " ";

}

toOut += "\n";

}

return toOut;

}

//Интерфейс перечисления

IEnumerator IEnumerable.GetEnumerator()

{

return data.GetEnumerator();

}

};

}

# 4.Пример решения

Входной файл IN.txt изображён на рисунке 4.1.

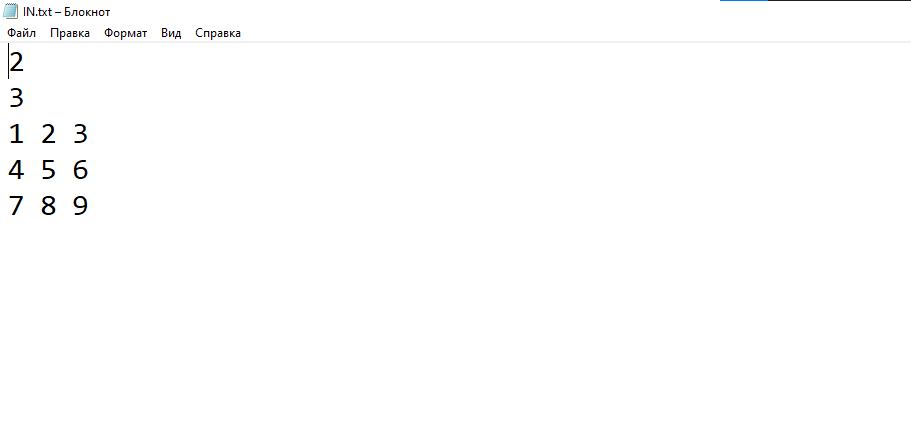


Рисунок 4.1— Входные данные из файла

Результат работы программы, записанный в файл OUT.txt изображён на рисунке 4.2, 4.3, 4,4.

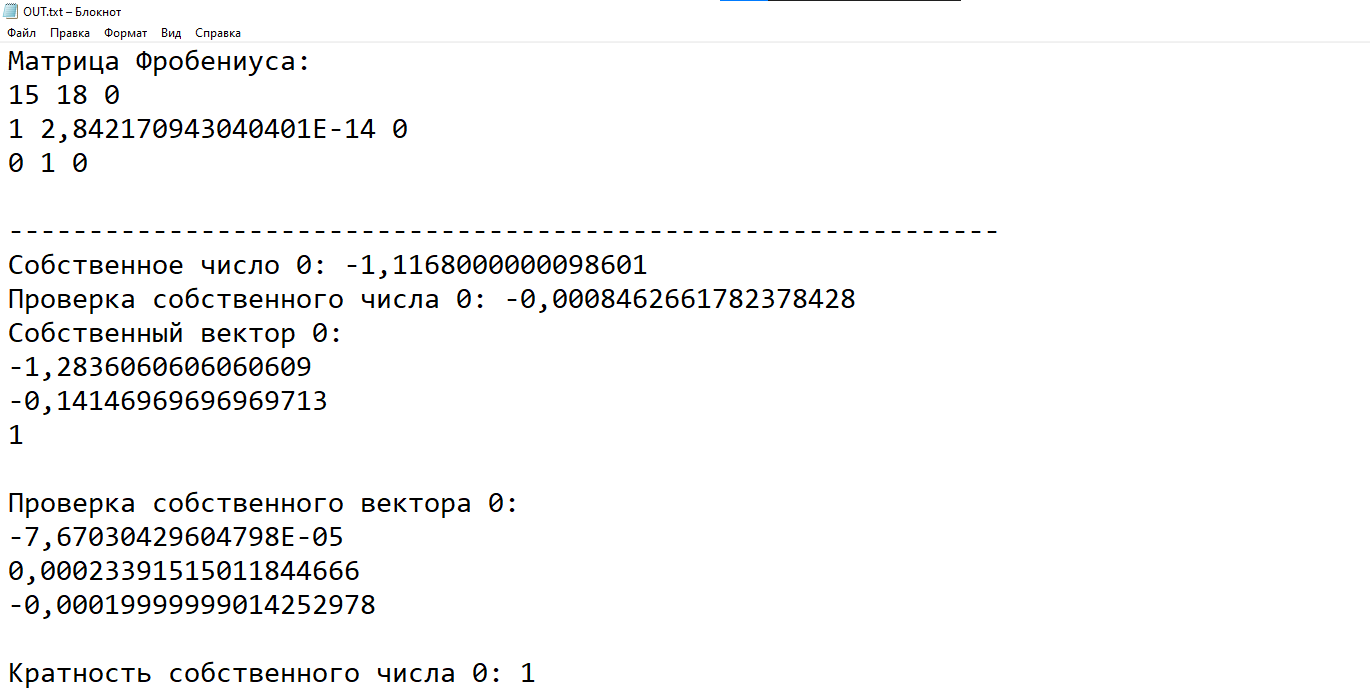


Рисунок 4.2 —Результат работы программы

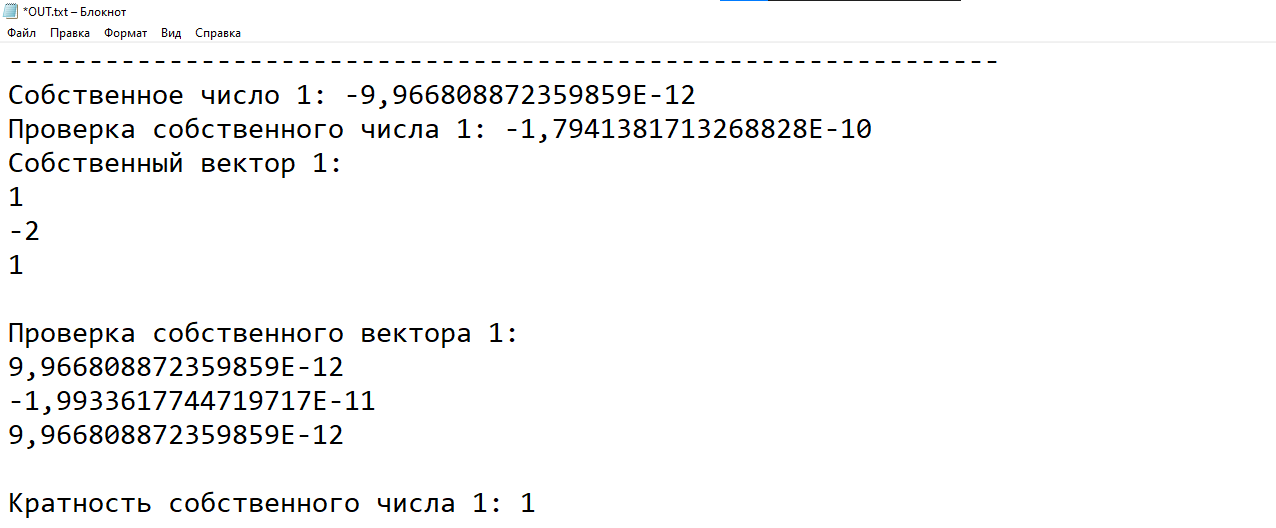


Рисунок 4.3 —Результат работы программы

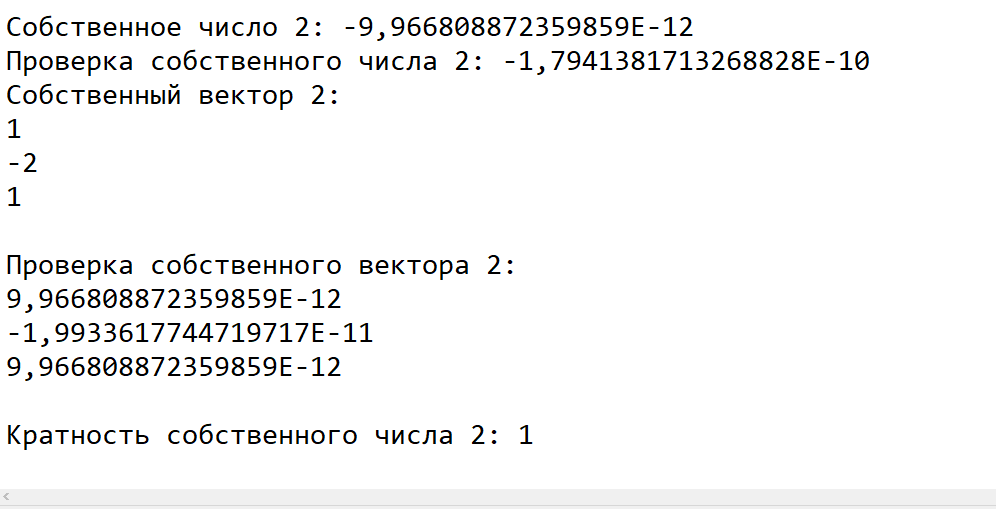


Рисунок 4.4 —Результат работы программы

# 5.Вывод

В результате лабораторной работы были изучены способы нахождения собственных чисел и векторов методом Данилевского, определение кратности чисел и векторов.