БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №4

**Методы решения проблемы собственных значений.**

**Метод Данилевского**

Выполнил:

Студент 2 курса 5 группы ФПМИ

Дунаев Виктор

Руководитель:

Радкевич Елена Владимировна

Минск, 2016 г.

**Оглавление**

Постановка задачи……………………………………………….……………..…..3

Построение собственного многочлена ………………………………….………..3

Вычисление собственных векторов матриц…..…………………………..…....…6

Листинг программы …………………………………………………….………….7

Входные данные …………………………………………………………….….….12

Выходные данные ………………………………………….……………………...12

**1.Постановка задачи**

Дана матрица , которая имеет следующий вид:

0.4997 -0.0658 0.0132 0.0263 0.0921

0.0684 0.7824 0.0000 -0.0526 0.0526

A = 0.0395 0.0000 0.6286 -0.1841 0.1052

-0.0789 0.1657 0.0000 0.6181 -0.0263

0.3288 0.0000 0.1184 0.0132 0.7364

1. Построить собственный многочлен матрицы  по методу Данилевского;
2. По возможности найти максимальное собственное значение  и собственный вектор матрицы , соответствующий собственному значению .

**2.Построение собственного многочлена**

Метод Данилевского использует тот факт, что подобные преобразования матрицы не изменяют её характеристический многочлен. Будем приводить матрицу  с помощью подобных преобразований к канонической форме Фробениуса ,

.

При этом 







Таким образом, элементы первой строки матрицы  - коэффициенты собственного многочлена .

Матрицу  целесообразно находить путём последовательного приведения строк к каноническому виду.

*А) Регулярный случай.*

Приводим строки к каноническому виду, начиная с последней. Пусть  Разделим -й столбец на . Умножим этот столбец на  и вычтем из -го столбца, , . Такое преобразование равносильно умножению матрицы  справа на матрицу ,

.

.   .

.

.

Коэффициенты  можно вычислить по следующим формулам:

, 

, 

, 

, 

Второй шаг состоит в приведении **-й строки матрицы  к виду Фробениуса. Пусть .

.

,

.

Если , , то



.

На шаге  коэффициенты  можно вычислить по следующим формулам:

, 

, 

, 

, 

*Б) Нерегулярный случай.*

Пусть процесс приведения матрицы  к виду Фробениуса доведён до строки , то есть выполнено  шагов, и при этом . Дальнейшее преобразование зависит от того, будут ли в -й строке левее  ненулевой элемент.

*Б1)* : . Вычисления можно свести к регулярному случаю перестановкой -го и -го столбцов. Чтобы преобразование было подобным, необходимо менять местами также соответствующие строки.

*Б2)* : . В этом случае матрица  имеет следующий вид:

.



По  можно построить собственный многочлен, с  процесс продолжается.

**3.Вычисление собственных векторов матрицы**

**Лемма.** Если  и  - подобные матрицы , и  - собственный вектор матрицы , соответствующий собственному значению , то  - собственный вектор матрицы , соответствующий тому же собственному значению .

Найдём собственный вектор матрицы , соответствующий собственному значению . Запишем систему в координатном виде:

.

Так как собственный вектор определяется с точностью до постоянного множителя, положим . Тогда из последних -го уравнения системы получим . Первое уравнение превращается в равенство, так как  является собственным значением матрицы . Исходя из построения матрицы , матрица  имеет вид .  - собственный вектор матрицы , соответствующий собственному значению . Его можно находить, последовательно умножая вектор  слева на , . При этом изменяется только -я компонента вектора. При втором варианте нерегулярного случая метода Данилевского использовать данный прием нельзя.

**4.Листинг программы**

**Main.cpp**

#include <iostream>

#include <fstream>

#include "cinoutFunctions.h"

#include "functions.h"

using namespace std;

int main ()

{

const int num = 5;

const double eps = 0.000001;

double \*\*A = new double\* [num],

\*\*S = new double\* [num],

\*\*M = new double\* [num],

\*eVect = new double [num],

\*r = new double [num],

eValue;

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

{

A[i] = new double [num];

S[i] = new double [num];

M[i] = new double [num];

}

ifstream isM ("matrix.txt");

ofstream os ("output.txt");

CinM (A, num, isM);

isM.close();

os << "Initial matrix:" << endl << endl;

CoutM (A, num, os);

os << endl;

if ( danilevsky (A, M, num) )

{

os << "Frobenius form:" << endl << endl;

CoutM (A, num, os);

eValue = findMaxEigenvalue (A, num, eps);

os << endl << "Max eigenvalue is " << eValue << endl << endl << "Eigenvector: " << endl << endl;

findEigenvector (M, eVect, eValue, num);

CoutV (eVect, num, os);

ifstream isM ("matrix.txt");

CinM (A, num, isM);

isM.close();

os << endl << "Residual:" << endl << endl;

findResidual (A, eVect, eValue, r, num);

CoutV (r, num, os);

}

os.close();

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

{

delete [] A[i];

delete [] S[i];

delete [] M[i];

}

delete [] A;

delete [] S;

delete [] M;

delete [] eVect;

delete [] r;

return 0;

}

**Functionc.h**

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

template <typename T>

void gaussTransformation (T\*\* A, int num)

{

T\*\* At = new T\* [num];

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

At[i] = new T [num];

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

for (int j = i; j < num; ++j)

{

At[i][j] = 0;

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

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

}

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

for (int j = i; j < num; ++j)

A[i][j] = A[j][i] = At[i][j];

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

delete [] At[i];

delete [] At;

}

**//Шаг метода Данилевского**

template <typename T>

bool danilevskyStep (T\*\* A, T\*\* B, T\*\* M, int num, int step)

{

int aboveNotUnitRow = num - step;

int notUnitRow = aboveNotUnitRow - 1;

if (A[aboveNotUnitRow][notUnitRow] == 0)

return false;

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

if (j != notUnitRow)

M[notUnitRow][j] = - A[aboveNotUnitRow][j] / A[aboveNotUnitRow][notUnitRow];

else

M[notUnitRow][j] = 1 / A[aboveNotUnitRow][notUnitRow];

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

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

{

B[i][j] = A[i][notUnitRow] \* M[notUnitRow][j];

if (j != notUnitRow)

B[i][j] += A[i][j];

}

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

if (j != notUnitRow)

B[aboveNotUnitRow][j] = 0;

else

B[aboveNotUnitRow][j] = 1;

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

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

A[i][j] = B[i][j];

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

{

A[notUnitRow][j] = 0;

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

A[notUnitRow][j] += A[aboveNotUnitRow][k] \* B[k][j];

}

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

if (j != notUnitRow)

A[aboveNotUnitRow][j] = 0;

else

A[aboveNotUnitRow][j] = 1;

return true;

}

**//Нахождение матрицы Фробениуса, подобной матрице **

template <typename T>

bool danilevsky (T\*\* A, T\*\* M, int num)

{

T\*\* B = new T\* [num];

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

B[i] = new T [num];

bool regular = true;

for (int i = 1; i < num; ++i)

if ( !danilevskyStep (A, B, M, num, i) )

{

regular = false;

break;

}

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

delete [] B[i];

delete [] B;

return regular;

}

**//Вычисление значения собственного многочлена в точке **

template <typename T>

T countPolynom (T\*\* F, int num, T x)

{

T res = pow (x, num);

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

res -= F[0][i] \* pow (x, num-i-1);

return res;

}

template <typename T>

T countDerivativePolynom (T\*\* F, int num, T x)

{

T res = num \* pow (x, num-1);

for (int i = 0; i < num-1; ++i)

res -= F[0][i] \* (num-i-1) \* pow (x, num-i-2);

return res;

}

**//Нахождение максимального собственного значения **

template <typename T>

T findMaxEigenvalue (T\*\* A, int num, T eps, T end = 5, T step = 0.1)

{

while (countPolynom (A, num, end) > 0)

end -= step;

T begin = end;

end += step;

// check derivative

T mid;

while (end - begin > eps)

{

mid = (end + begin) / 2;

if (countPolynom (A, num, mid) > 0)

end = mid;

else

begin = mid;

}

return mid;

}

**//Нахождение собственного вектора, соответствующего собственному значению **

template <typename T>

void findEigenvector (T\*\* M, T\* eVector, T eValue, int num)

{

T tmp;

eVector[num-1] = 1;

for (int i = num - 2; i >= 0; --i)

eVector[i] = eVector[i+1] \* eValue;

for (int i = 0; i < num - 1; ++i)

{

tmp = 0;

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

tmp += M[i][j] \* eVector[j];

eVector[i] = tmp;

}

}

template <typename T>

void findResidual (T\*\* A, T\* eVector, T eValue, T\* r, int num)

{

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

{

r[i] = - eValue \* eVector[i];

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

r[i] += A[i][j] \* eVector[j];

}

}

#endif

**cinoutFunctions.h**

#ifndef CINOUTFUNCTIONS\_H

#define CINOUTFUNCTIONS\_H

#include <iostream>

using namespace std;

template <typename T>

void CoutM (T\*\* m, int num, ostream& os)

{

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

{

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

{

os.width (25);

os.precision (15);

os << m[i][j];

}

os << endl;

}

}

template <typename T>

void CinM (T\*\* m, int num, istream& is)

{

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

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

is >> m[i][j];

}

template <typename T>

void CoutV (T\* v, int num, ostream& os)

{

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

{

os.precision (15);

os << v[i] << endl;

}

}

template <typename T>

void CoutLV (T\* v, int num, ostream& os)

{

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

{

os.precision (15);

os << "a[" << i << "][" << i << "] (" << i << ") = " << v[i] << endl;

}

}

template <typename T>

void CoutDV (T\* v, int num, ostream& os)

{

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

{

os.precision (15);

os << "dx[" << i << "] = " << v[i] << endl;

}

}

template <typename T>

void CinV (T\* v, int num, istream& is)

{

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

is >> v[i];

}

#endif

**5.Входные данные**

0.4997 -0.0658 0.0132 0.0263 0.0921

0.0684 0.7824 0.0000 -0.0526 0.0526

A = 0.0395 0.0000 0.6286 -0.1841 0.1052

-0.0789 0.1657 0.0000 0.6181 -0.0263

0.3288 0.0000 0.1184 0.0132 0.7364

**6.Выходные данные**

Данная матрица является регулярным случаем метода Данилевского.

Коэффициенты собственного многочлена:

P1 = 3.2652

P2 = -4.2124

P3 = 2.6810

P4 = -0.8403

P5 = 0.1034

Собственный многочлен:

 =L^5 + 3.2652\*L^4 - 4.2124\*L^3 + 2.6810\*L^2 - 0.8403\*L^1 + 0.1034.

Максимальное собственное значение U(max)= 0.4022.

Собственный вектор, соответствующий собственному значению U(max):

-0.842592117666229

-0.0113199683552523

-0.462176212614022

-0.177495527280946

1