**Лабораторная работа 9 «Нелинейные уравнения»**

Выполнил студент 3 курса 4 группы ФПМИ БГУ Видевич Александр

**Задача.**

Изучить основы итерационный степенного метода вычисления наибольшего по модулю собственного значения и соответствующего собственного вектора; разработать программу, реализующую основной случай метода.

Задание. Вычислить вещественные корни собственного многочлена четвертой степени *P*(λ), полученного из канонической формы Фробениуса лабораторной работы «Метод Данилевского». Проверить некоторые положения (далее уточняется какие), связанные с темами получения собственных значений и собственных векторов матрицы.

1. Произвести отделение корней многочлена *P*(λ).

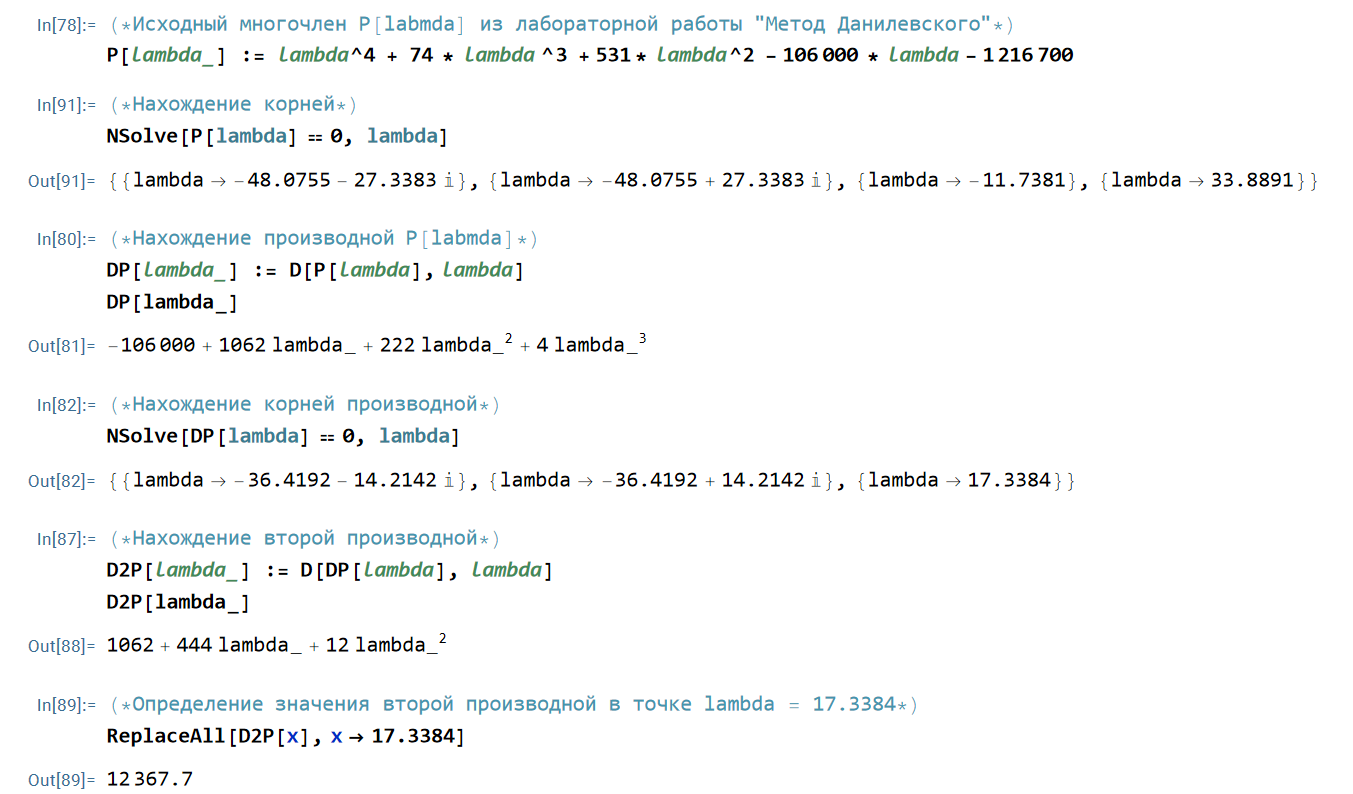
Для определения промежутков монотонности функции *P*(λ) решить уравнение *P'*(λ)=0: *а)* применить метод деления отрезка пополам, *б)* применить метод Ньютона. Предварительно произвести отделение корней многочлена *P'*(λ). Если требуется, при отделении корней многочлена *P'*(λ) найти промежутки монотонности функции *P'*(λ): решить квадратное уравнение *P''*(λ)*=*0.

Если отделения корней многочлена *P*(λ) не произошло, то вещественных корней многочлен *P*(λ) не имеет. Тогда два (наибольших по модулю) корня образуют комплексно-сопряженную пару и имеет место Случай 4 файла «Степенной метод». Проверить, справедливо ли Замечание 9: Характерным признаком Случая 4 является колебательный характер последовательности приближений.

2. Приближенно вычислить вещественные корни собственного многочлена методом Ньютона.

3. Хотя бы для одного найденного собственного значения (корня собственного многочлена): с помощью матриц *M*3, *M*2, *M*1, полученных в лабораторной работе «Метод Данилевского», найти соответствующий собственный вектор *u*. Проверить *Au*–λ*u≈*0.

**Определение промежутков монотонности функции.**





**Листинг программы.**

*#include* <iostream>

*#include* <vector>

*#include* <iomanip>

*#include* <optional>

*#include* <cmath>

*const* int n = *4*;

*const* double epsilon = *0.0000001*;

*// данные из лабораторной работы 7*

*const* std::*vector*<std::*vector*<double>> A = {

    {-*18*, *27*, -*4*, -*37*},

    {*18*, -*34*, -*38*, *10*},

    {-*23*, *6*, -*1*, *26*},

    {*6*, *0*, *43*, -*21*}};

*const* std::*vector*<std::*vector*<double>> M3 = {

    {*1*, *0*, *0*, *0*},

    {*0*, *1*, *0*, *0*},

    {-*0.13953*, -*0*, *0.023256*, *0.48837*},

    {*0*, *0*, *0*, *1*}};

*const* std::*vector*<std::*vector*<double>> M2 = {

    {*1*, *0*, *0*, *0*},

    {*2.5896*, *0.002381*, *0.05371*, -*2.0554*},

    {*0*, *0*, *1*, *0*},

    {*0*, *0*, *0*, *1*}};

*const* std::*vector*<std::*vector*<double>> M1 = {

    {-*1.1866e*-*05*, -*0.0015008*, -*0.020778*, *1.5247*},

    {*0*, *1*, *0*, *0*},

    {*0*, *0*, *1*, *0*},

    {*0*, *0*, *0*, *1*}};

double GetPLambda(double lambda)

{

*return* (lambda \* lambda \* lambda \* lambda + *74* \* lambda \* lambda \* lambda + *531* \* lambda \* lambda - *106000* \* lambda - *1216700*);

}

double GetDPdLambda(double lambda)

{

*return* (*4* \* lambda \* lambda \* lambda + *222* \* lambda \* lambda + *1062* \* lambda - *106000*);

}

double GetSign(double value)

{

*if* (value < *0*)

    {

*return* -*1*;

    }

*if* (value > *0*)

    {

*return* *1*;

    }

*return* *0*;

}

double RunNewtonMethod(double x0)

{

    double x1 = x0 - GetPLambda(x0) / GetDPdLambda(x0);

*while* (std::fabs(x1 - x0) >= epsilon)

    {

        x0 = x1;

        x1 = x0 - GetPLambda(x0) / GetDPdLambda(x0);

    }

*return* x1;

}

double RunDichotomyMethod(double x0, double x1)

{

    double x2 = (x0 + x1) / *2*;

    double delta2 = (x1 - x0) / *2*;

    double delta3 = delta2 / *2*;

    double x3 = x2 - delta3 \* GetSign(GetPLambda(x2));

*while* (std::fabs(x3 - x2) >= epsilon)

    {

        x2 = x3;

        delta2 = delta3;

        delta3 = delta2 / *2*;

        x3 = x2 - delta2 \* GetSign(GetPLambda(x2));

    }

*return* x3;

}

std::*vector*<double> GetY(double lambda)

{

    std::*vector*<double> y(n);

    y[n - *1*] = *1*;

*for* (int i = n - *2*; i >= *0*; --i)

    {

        y[i] = y[i + *1*] \* lambda;

    }

*return* y;

}

std::*vector*<std::*vector*<double>> MultiplyMatrices(*const* std::*vector*<std::*vector*<double>> *&*a,

*const* std::*vector*<std::*vector*<double>> *&*b)

{

    std::*vector*<std::*vector*<double>> result(n, std::*vector*<double>(n, *0*));

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

    {

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

        {

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

            {

                result[i][j] += a[i][k] \* b[k][j];

            }

        }

    }

*return* result;

}

std::*vector*<double> MultiplyMatrixVector(*const* std::*vector*<std::*vector*<double>> *&*matrix, *const* std::*vector*<double> *&*vector)

{

*const* int size = vector.size();

    std::*vector*<double> result(size);

*for* (int i = *0*; i < size; i++)

    {

        double sum = *0*;

*for* (int j = *0*; j < size; j++)

        {

            sum += matrix[i][j] \* static\_cast<double>(vector[j]);

        }

        result[i] = sum;

    }

*return* result;

}

std::*vector*<double> GetEigenvector(double lambda)

{

    std::*vector*<double> y = GetY(lambda);

    std::*vector*<double> eigenvector = MultiplyMatrixVector(MultiplyMatrices(MultiplyMatrices(M3, M2), M1), y);

*return* eigenvector;

}

void PrintVector(*const* std::*vector*<double> *&*v)

{

*for* (auto el : v)

    {

        std::cout << el << ' ';

    }

    std::cout << '*\n*';

}

std::*vector*<double> CheckEigenvector(*const* std::*vector*<double> *&*v, double lambda)

{

    std::*vector*<double> Au = MultiplyMatrixVector(A, v);

    std::*vector*<double> result(n);

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

    {

        result[i] = Au[i] - lambda \* v[i];

    }

*return* result;

}

*#define* INF *9999*

int main()

{

*// применяется метод деления отрезка пополам*

    double lambda1 = RunDichotomyMethod(*17.338*, INF);

    std::cout << "Lambda 1 (Dichotomy method): " << lambda1 << '*\n*';

    std::cout << "P(lambda1) (Dichotomy method): " << GetPLambda(lambda1) << '*\n*';

    double lambda2 = RunDichotomyMethod(*17.338*, -INF);

    std::cout << "Lambda 2 (Dichotomy method): " << lambda2 << '*\n*';

    std::cout << "P(lambda2) (Dichotomy method): " << GetPLambda(lambda2) << '*\n*';

*// приближенно вычисляются вещественные корни собственного многочлена методом Ньютона.*

    lambda1 = RunNewtonMethod(*5000*);

    std::cout << "Lambda 1 (Newton method): " << lambda1 << '*\n*';

    std::cout << "P(lambda1) (Newton method): " << GetPLambda(lambda1) << '*\n*';

    lambda2 = RunNewtonMethod(-*5000*);

    std::cout << "Lambda 2 (Newton method): " << lambda2 << '*\n*';

    std::cout << "P(lambda2) (Newton method): " << GetPLambda(lambda2) << '*\n*';

*// с помощью матриц M3, M2, M1, полученных в лабораторной работе «Метод Данилевского», находится соответствующий собственный вектор u.*

    std::*vector*<double> eigenvector = GetEigenvector(lambda1);

    std::cout << "Eigenvector : ";

    PrintVector(eigenvector);

*// проверяется Au–λu≈0.*

    std::cout << "A\*u - lambda\*u : ";

    std::*vector*<double> result = CheckEigenvector(eigenvector, lambda1);

    PrintVector(result);

*return* *0*;

}

**Выходные данные.**

Lambda 1 (Dichotomy method): 33.8891

P(lambda1) (Dichotomy method): -0.0143797

Lambda 2 (Dichotomy method): -11.7381

P(lambda2) (Dichotomy method): 0.00270458

Lambda 1 (Newton method): 33.8891

P(lambda1) (Newton method): 0

Lambda 2 (Newton method): -11.7381

P(lambda2) (Newton method): -2.32831e-10

Eigenvector : -1.3649 -1.03526 1.46694 1

A\*u - lambda\*u : 0.00368156 -0.0289662 0.0010208 -0.000105517

**Вывод**

Анализ этих данных позволяет сделать следующие выводы:

* Метод деления отрезка пополам и метод Ньютона дали одинаковые значения для λ1 и λ2, что указывает на их сходимость и точность результатов.
* Значение P(λ1) равно 0 в методе Ньютона, что говорит о том, что λ1является корнем собственного многочлена.
* Значение P(λ2) близко к нулю в методе Ньютона, что указывает на то, что λ2 также является корнем собственного многочлена, но с небольшой погрешностью.
* Проверка Au - λu показывает, что вектор u является собственным вектором матрицы A с соответствующим собственным значением λ.