**Лабораторная работа 8 «Итерационный степенной метод»**

Выполнил студент 3 курса 4 группы ФПМИ БГУ Видевич Александр

**Задача.**

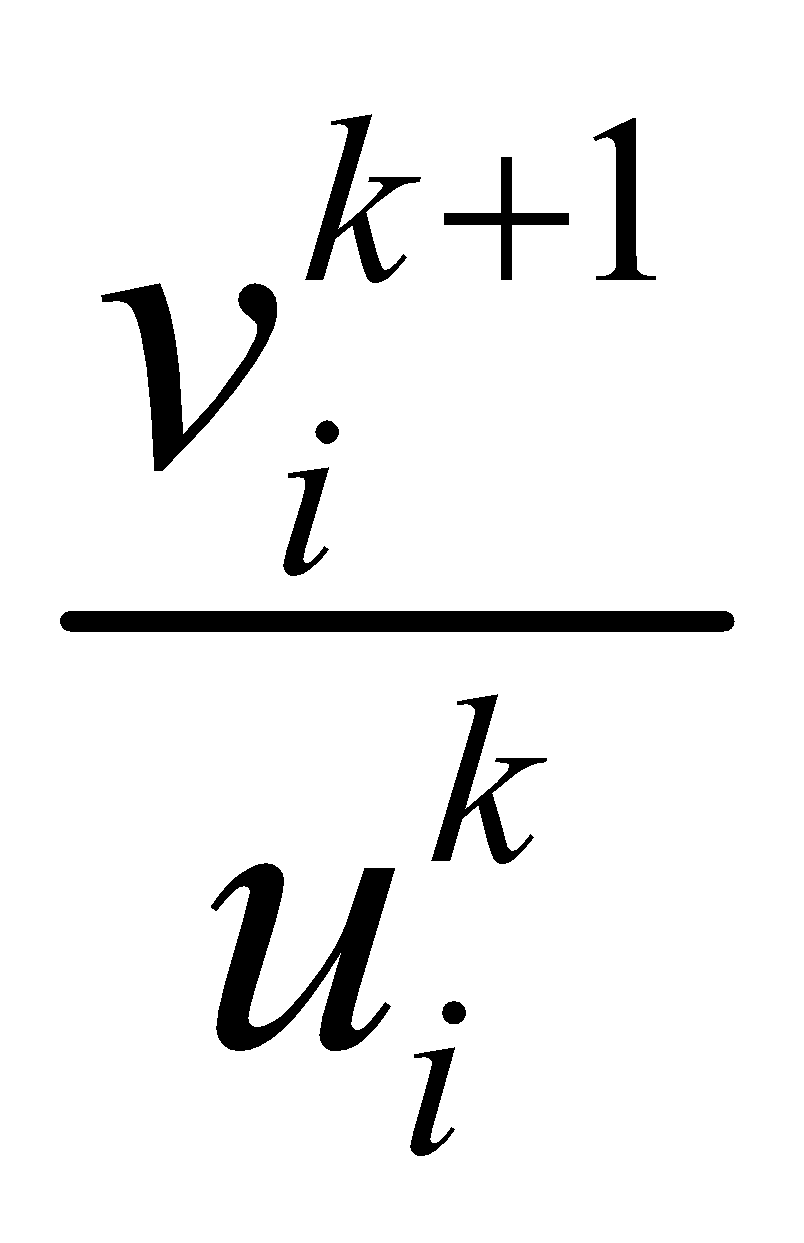
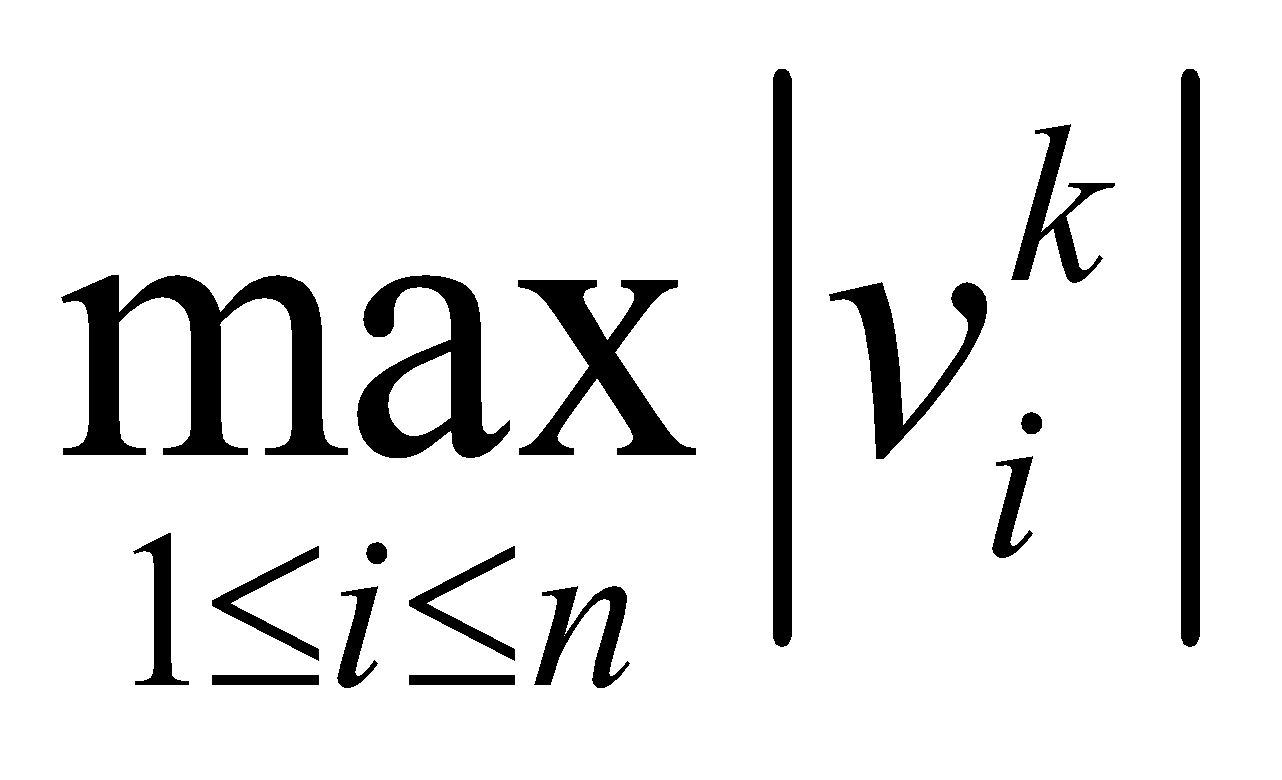
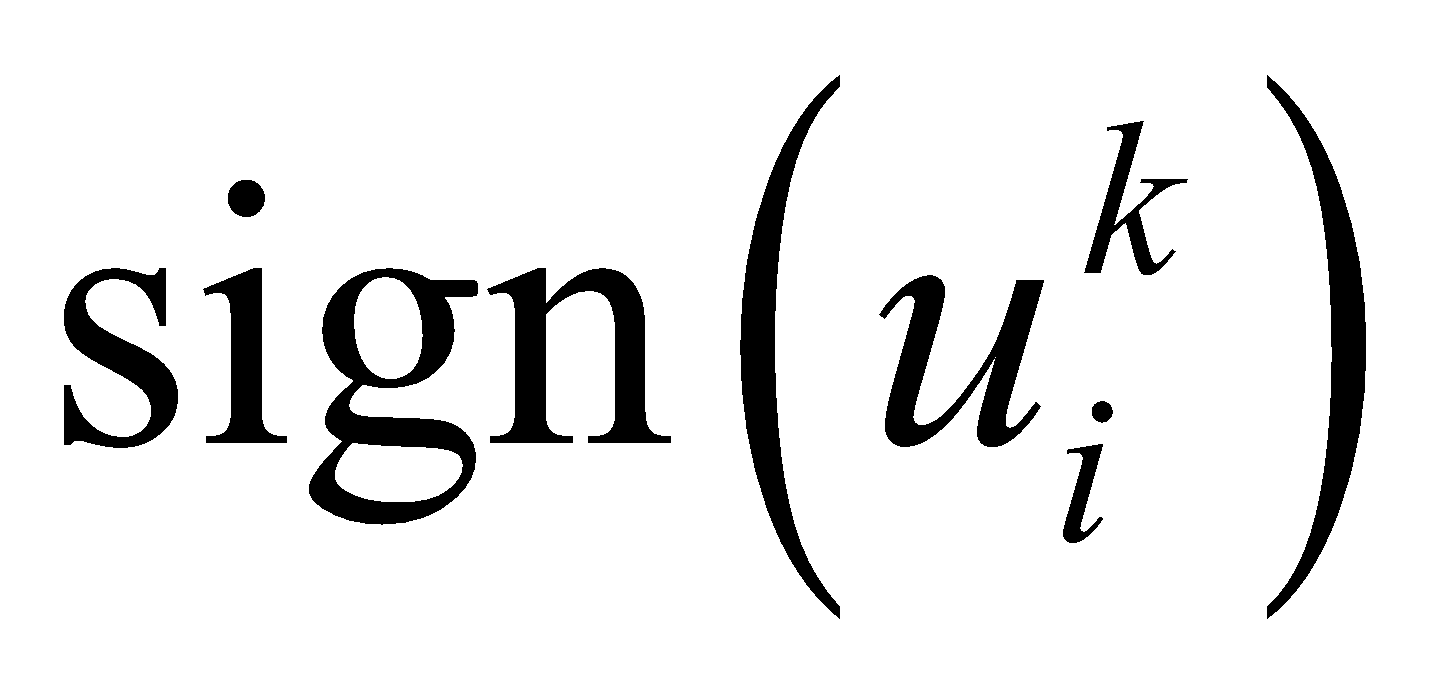
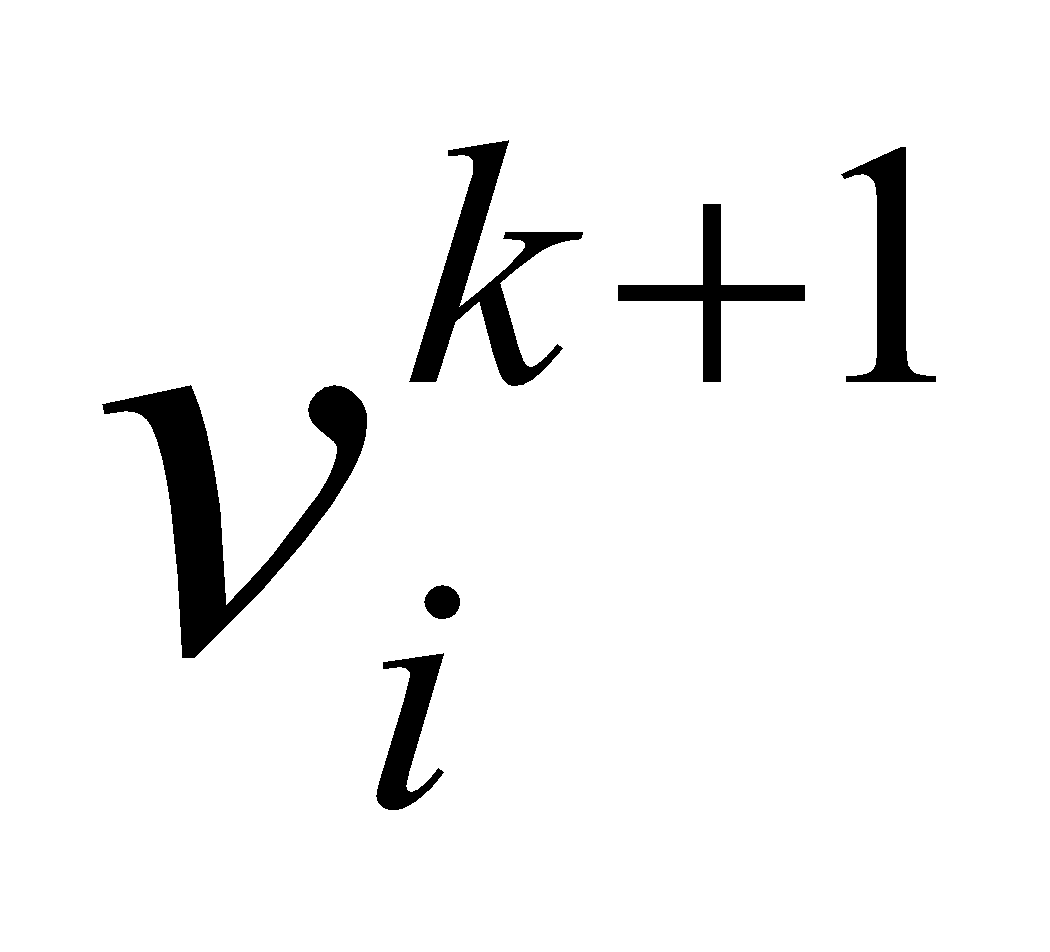
Изучить основы итерационный степенного метода вычисления наибольшего по модулю собственного значения и соответствующего собственного вектора; разработать программу, реализующую основной случай метода.

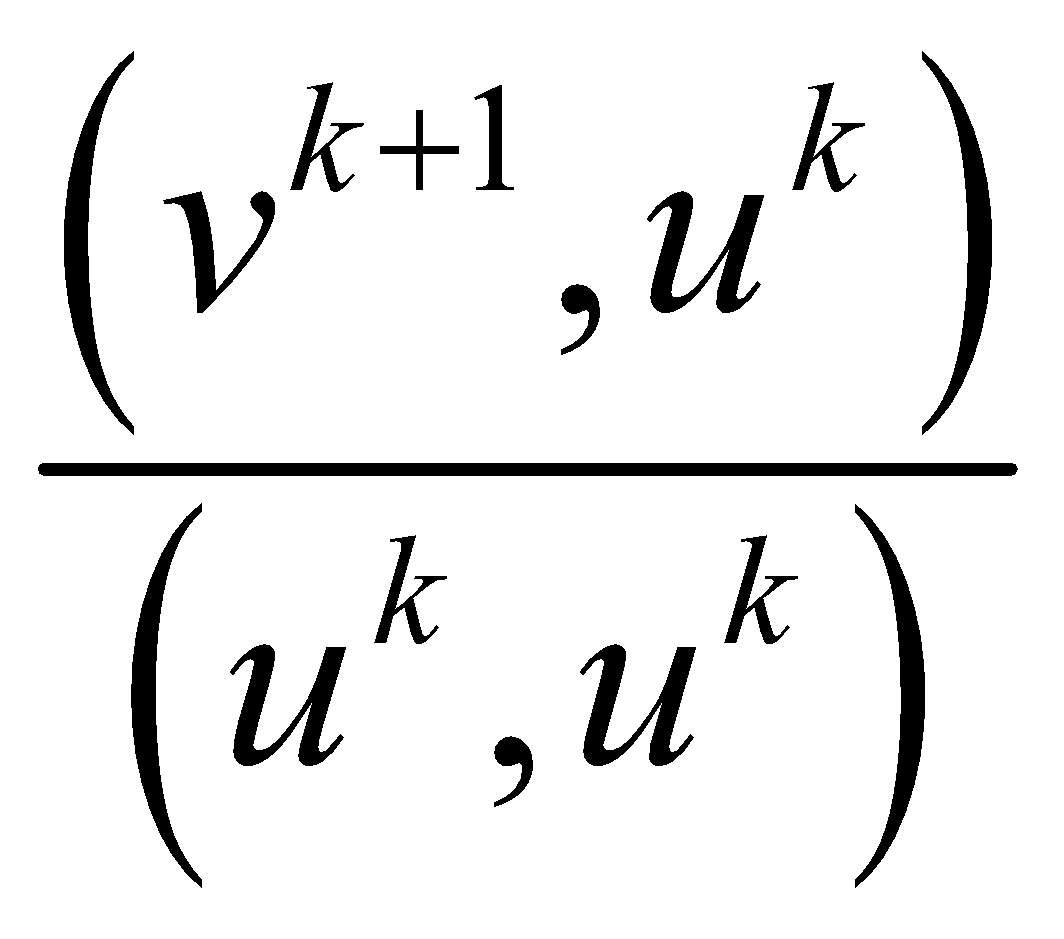
Задание. Разработать программу вычисления наибольшего по величине модуля собственного значения (случай вещественного не кратного собственного значения) и соответствующего ему собственного вектора симметричной матрицы.

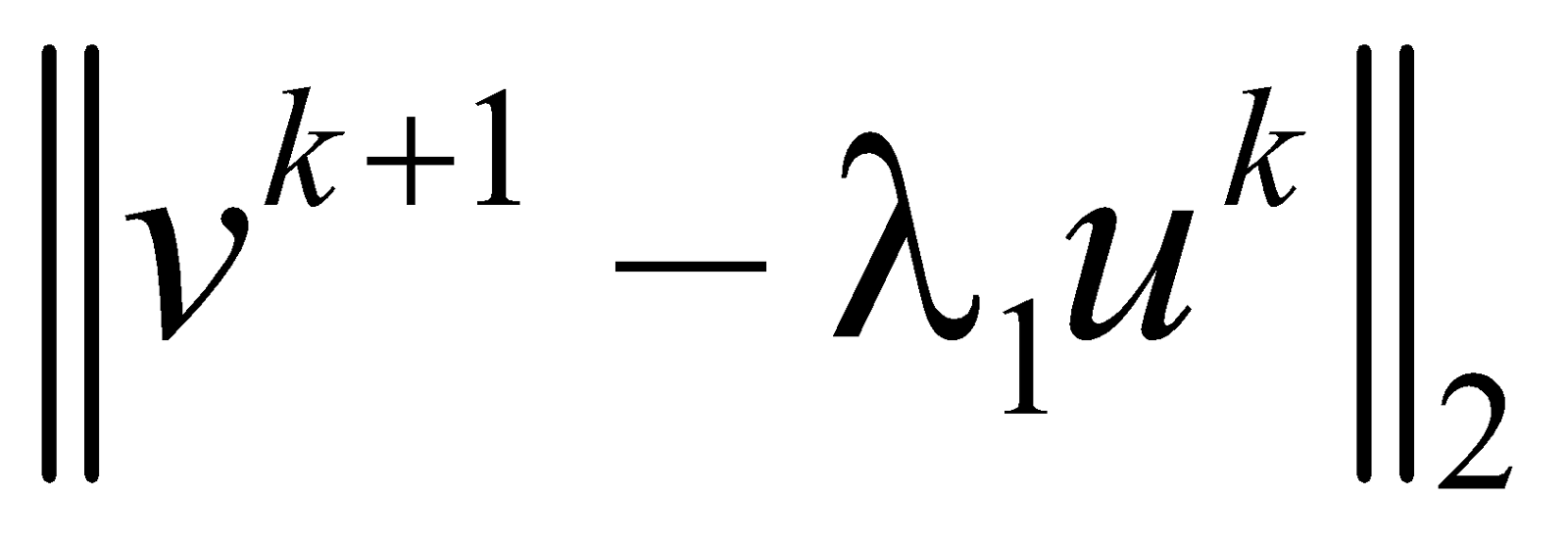
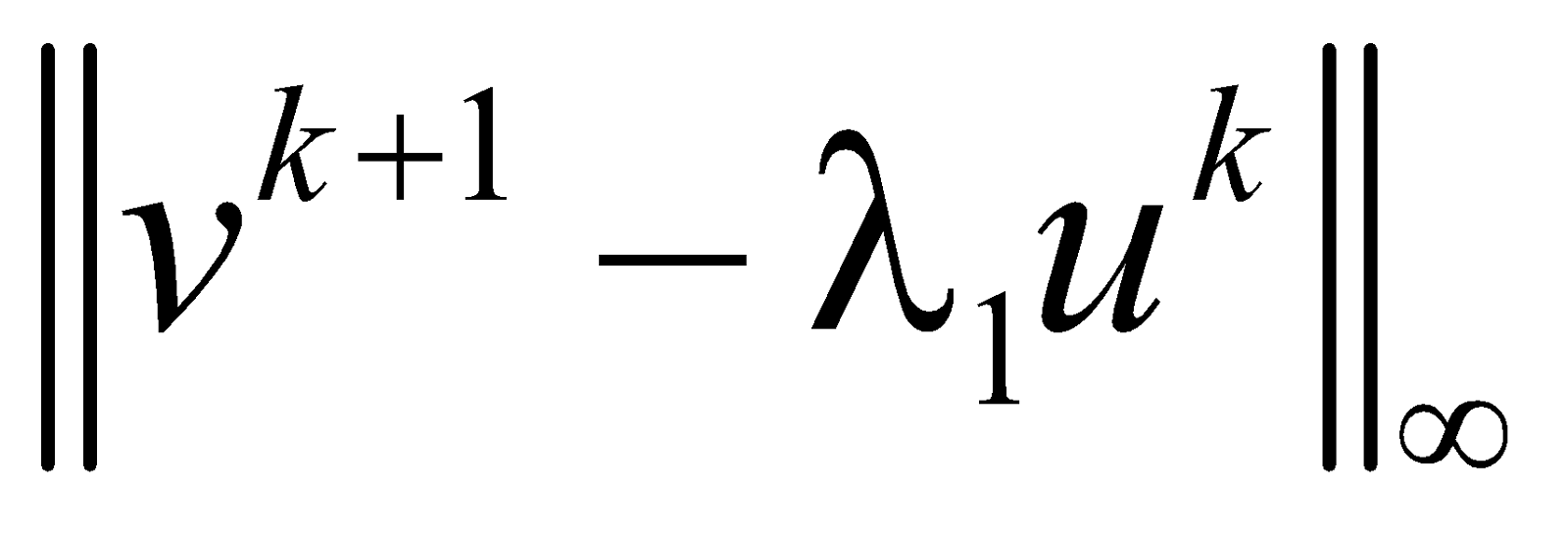
Матрицу (*n* – порядок матрицы, ξ – номер группы) задать как в лабораторной работе 2.

Для вычисления наибольшего по модулю собственного значения и соответствующего собственного вектора использовать формулы из пункта Случай 1 (файл «Степенной метод»).

Вывести на печать *uk* (приближенно это собственный вектор, соответствующий собственному значению λ1) при *k=*50.

В формуле λ1≈ (формула вспомогательная, не для счета) выбирать такое *i*, для которого достигается ). В этом случае справедливо (это уже для счета) λ1≈.

Для вычисления наибольшего по модулю собственного значения использовать также формулу λ1≈.

Для обоих случаев приближённого вычисления λ1: вывести на печать приближенное λ1 для *k=*50; вычислить вектор *vk+*1–λ1*uk* (это для проверки насколько оказались хороши вычисленные λ1 и *uk*: вектор *vk+*1–λ1*uk* равен вектору невязки *Auk*–λ1*uk*); для *k=*50 вычислить (для обоих случаев) и вывести на печать или .

**Входные данные.**

n = 1700, K =50

**Листинг программы.**

*#include* <vector>

*#include* <cmath>

*#include* <iostream>

*#include* <ctime>

*#include* <limits>

int n;

*// функция генерации матрицы*

*#define* KSI *4*

std::*vector*<std::*vector*<double>> GenerateMatrix(*const* int size)

{

    std::*vector*<std::*vector*<double>> matrix(size, std::*vector*<double>());

*for* (int i = size - *1*; i >= *0*; --i)

    {

        double diagElement = *0*;

*for* (int j = size - *1*; j > i; --j)

        {

            diagElement += matrix[j][i];

        }

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

        {

*if* (i == j)

            {

                matrix[i].push\_back(*0.0f*);

*continue*;

            }

            matrix[i].push\_back((double)rand() / RAND\_MAX \* -*1000.0f*);

            diagElement += matrix[i][j];

        }

*if* (i == *0*)

        {

            matrix[i][i] = -diagElement + std::pow(*10*, *2* - KSI);

        }

*else*

        {

            matrix[i][i] = -diagElement;

        }

    }

*return* matrix;

}

*// функция умножения матрицы на вектор*

template <typename T>

std::*vector*<double> GetMatrixVectorMultiplyResult(*const* std::*vector*<std::*vector*<double>> *&*matrix, *const* std::*vector*<T> *&*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 ? i : j][i >= j ? j : i] \* static\_cast<double>(vector[j]);

        }

        result[i] = sum;

    }

*return* result;

}

double GetMaximumNorm(*const* std::*vector*<double> *&*vector)

{

    double result = -*99999*;

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

    {

*if* (vector[i] > result)

        {

            result = vector[i];

        }

    }

*return* result;

}

template <typename T>

void PrintVector(*const* std::*vector*<T> *&*vector, *const* int numberOfElements = *0*, *const* std::string *&*message = "")

{

*if* (message != "")

    {

        std::cout << message << ' ';

    }

*const* int border = numberOfElements != *0* ? numberOfElements : vector.size();

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

    {

        std::cout << vector[i] << ' ';

    }

    std::cout << '*\n*';

}

int GetSign(*const* int number)

{

*if* (number == *0*)

*return* *0*;

*if* (number > *0*)

*return* *1*;

*return* -*1*;

}

*#define* K *50*

double GetScalarProduct(*const* std::*vector*<double> *&*v1, *const* std::*vector*<double> *&*v2)

{

    double res = *0*;

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

    {

        res += v1[i] \* v2[i];

    }

*return* res;

}

void RunMethod(*const* std::*vector*<std::*vector*<double>> *&*matrix)

{

    std::*vector*<double> u(n, *0*);

    u[*0*] = *1*;

    std::*vector*<double> v(n);

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

    {

        v = GetMatrixVectorMultiplyResult(matrix, u);

*const* double vNorm = GetMaximumNorm(v);

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

        {

            u[j] = v[j] / vNorm;

        }

    }

    int maxIndex = *0*;

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

    {

*if* (std::abs(v[maxIndex]) <= std::abs(v[i]))

        {

            maxIndex = i;

        }

    }

    PrintVector(u, u.size(), "u^k");

    double lambda = v[maxIndex] \* GetSign(u[maxIndex]);

    std::cout << "lambda1 = vi^k+1\* sign(ui^k) = " << lambda << '*\n*';

    std::*vector*<double> au = GetMatrixVectorMultiplyResult(matrix, u);

    std::*vector*<double> residualVector(n);

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

    {

        residualVector[i] = au[i] - lambda \* u[i];

    }

    PrintVector(residualVector, residualVector.size(), "v^k+1 - labmda \* u^k");

    std::cout << "Maximum norm = " << GetMaximumNorm(residualVector) << '*\n*';

    lambda = GetScalarProduct(v, u) / GetScalarProduct(u, u);

    std::cout << "lambda1 = (v^k+1, u^k)/(u^k,u^k) = " << lambda << '*\n*';

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

    {

        residualVector[i] = au[i] - lambda \* u[i];

    }

    PrintVector(residualVector, residualVector.size(), "v^k+1 - labmda \* u^k");

    std::cout << "Maximum norm = " << GetMaximumNorm(residualVector);

}

int main()

{

    std::ios\_base::sync\_with\_stdio(*false*);

    std::cin.tie(*nullptr*);

    std::srand(static\_cast<unsigned int>(std::time(*nullptr*)));

*// ввод данных*

    std::cin >> n;

*// генерация задачи*

    std::*vector*<std::*vector*<double>> matrix = GenerateMatrix(n);

*// std::cout* << *'\n';*

*// for (auto &r : matrix)*

*// {*

*//     for (auto c : r)*

*//     {*

*//         std::cout* << *c* << *' ';*

*//     }*

*//     std::cout* << *'\n';*

*// }*

*// std::cout* << *'\n';*

    RunMethod(matrix);

*return* *0*;

}

**Выходные данные.**

lambda1 = vi^k+1\* sign(ui^k) = 854146

Maximum norm = 2863.22

lambda1 = (v^k+1, u^k)/(u^k,u^k) = 854146

Maximum norm = 2863.22

**Вывод**

Итерационный степенной метод позволяет вычислить наибольшее по модулю собственное значение и соответствующий ему собственный вектор симметричной матрицы.