Отчет по дисциплине: «Численные методы»

Лабораторная работа №8

## «Итерационный степенной метод»

Подготовил студент 3 курса 4 группы

Кондратович Артём

Необходимый для выполнения работы теоретический материал и формулы имеются в файле «Степенной метод».

Цель работы. 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 (приближенно это собственный вектор, соответствующий собственному значению λ1) при *k=*50.

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

Для вычисления наибольшего по модулю собственного значения использовать также формулу .

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 .

Листинг программы:

#include <iostream>

#include <random>

#include <vector>

#include <chrono>

#include "Operators.h"

const int k = 4;

const int iterations = 50;

double realRand() /\* generate float/double value in range [-1000, 0] \*/

{

static std::random\_device rd;

static std::mt19937 gen(rd());

static std::uniform\_real\_distribution<double> dis(-1000, 0);

return dis(gen);

}

int intRand() /\* generate n value in range [1500, 2000] \*/

{

static std::random\_device rd;

static std::mt19937 gen(rd());

static std::uniform\_int\_distribution<int> dis(1500, 2000);

return dis(gen);

}

std::vector<std::vector<double>> GenerateMatrixA(int n)

{

std::vector<std::vector<double>> matrix(n, std::vector<double>(n));

for (size\_t i = 0; i < n; i++)

{

for (size\_t j = i + 1; j < n; j++)

{

matrix[i][j] = realRand();

matrix[j][i] = matrix[i][j];

}

}

matrix[0][0] = std::pow(10, 2 - k);

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

matrix[0][0] += -matrix[0][i];

}

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

matrix[i][i] = -matrix[i][0];

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

if (i != j)

matrix[i][i] += -matrix[i][j];

}

}

return matrix;

}

void PrintReport(const std::vector<std::vector<double>>& a,const std::vector<double>& u,double lamdba, double time)

{

std::cout << std::setprecision(15);

std::cout << "Lambda = " << lamdba << std::endl;

std::cout << "|A \* u - lambda \* u| = " << MaxNorm(a \* u - u \* lamdba) << std::endl;

std::cout << "Time = " << time << "s" << std::endl;

}

void SolveFirstSituation(const std::vector<std::vector<double>>& a)

{

std::vector<double> y(a.size());

y[0] = 1;

std::vector<double> u = y;

double lambda = 0;

auto t1 = std::chrono::steady\_clock::now();

for (auto i = 0; i < iterations ; i++)

{

// index

auto index = IndexOfMax(y);

// v k+1

auto v = a \* u;

lambda = v[index] \* Sign(u[index]);

u = v / MaxNorm(v);

y = v;

}

auto t2 = std::chrono::steady\_clock::now();

PrintReport(a, u,lambda, std::chrono::duration<double>(t2 - t1).count());

}

void SolveSecondSituation(const std::vector<std::vector<double>>& a)

{

std::vector<double> y(a.size());

y[0] = 1;

std::vector<double> u = y;

double lambda = 0;

auto t1 = std::chrono::steady\_clock::now();

for (auto i = 0; i < iterations; i++)

{

auto v = a \* u;

lambda = (v \* u) / (u \* u);

u = v / MaxNorm(v);

}

auto t2 = std::chrono::steady\_clock::now();

PrintReport(a, u, lambda, std::chrono::duration<double>(t2 - t1).count());

}

int main()

{

auto n = intRand();

std::cout << "Genrated n = " << n << std::endl;

auto a = GenerateMatrixA(n);

SolveFirstSituation(a);

SolveSecondSituation(a);

}

Операторы:

#include "Operators.h"

std::vector<std::vector<double>> operator\* (const std::vector<std::vector<double>>& a, const std::vector<std::vector<double>>& b)

{

std::vector<std::vector<double>> result(a.size(), std::vector<double>(a.size()));

for (size\_t i = 0; i < a.size(); i++) {

for (size\_t j = 0; j < a.size(); j++) {

for (size\_t k = 0; k < a.size(); k++) {

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

}

}

}

return result;

}

std::vector<double> operator\* (const std::vector<std::vector<double>>& m, const std::vector<double>& v)

{

std::vector<double> result(v.size());

for (auto i = 0; i < m.size(); i++)

{

auto sum = 0.;

for (auto j = 0; j < m[i].size(); j++)

{

sum += m[i][j] \* v[j];

}

result[i] = sum;

}

return result;

}

std::ostream& operator << (std::ostream& stream, const std::vector<std::vector<double>>& matrix)

{

stream << std::fixed << std::setprecision(8);

for (auto const& line : matrix) {

for (auto const& element : line) {

stream << std::setw(14) << element << " ";

}

stream << std::endl;

}

return stream;

}

double MaxNorm(const std::vector<double>& v)

{

auto max = 0.;

for (auto x : v)

max = std::max(max, std::fabs(x));

return max;

}

double Sign(double n)

{

return n >= 0. ? 1. : -1.;

}

std::vector<double> operator/ (const std::vector<double>& v, double s)

{

if (s == 0)

{

throw "Division by zero";

}

std::vector<double> result(v.size());

for (size\_t i = 0; i < v.size(); i++)

{

result[i] = v[i] / s;

}

return result;

}

std::vector<double> operator\* (const std::vector<double>& v, double s)

{

std::vector<double> result(v.size());

for (size\_t i = 0; i < v.size(); i++)

{

result[i] = v[i] \* s;

}

return result;

}

std::vector<double> operator- (const std::vector<double>& a, const std::vector<double>& b)

{

std::vector<double> result(a.size());

for (size\_t i = 0; i < a.size(); i++)

{

result[i] = a[i] - b[i];

}

return result;

}

double operator\* (const std::vector<double>& a, const std::vector<double>& b)

{

auto result = 0.;

for (size\_t i = 0; i < a.size(); i++)

{

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

}

return result;

}

size\_t IndexOfMax(const std::vector<double>& v)

{

auto max = 0.;

auto index = 0;

for (auto i = 0; i < v.size(); i++)

{

if (max < std::fabs(v[i]))

{

max = std::fabs(v[i]);

index = i;

}

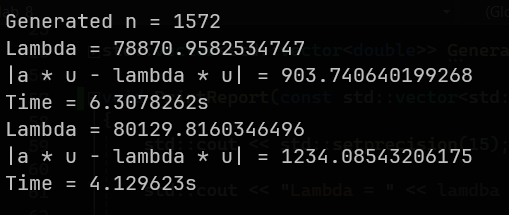
}

return index;

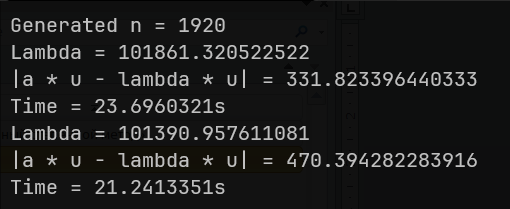
}

Результаты:

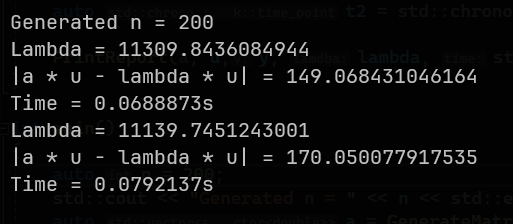
При iteration = 50

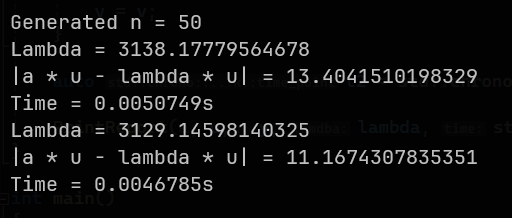


При Iterations = 200



Так как разняться данные попробуем уменьшить размерность





1000 итераций

