Белорусский Государственный Университет

Факультет Прикладной Математики и Информатики

Отчет

Лабораторная работа №6

«Итерационный степенной метод»

Профессор кафедры вычислительной математики ФПМИ

Лиходед Николай Александрович

Студент 2 группы 2 курса

Сачек Илья Валерьевич

2019 год

**Постановка задачи**:

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

**Задание 1.** Разработать программу вычисления наибольшего и второго по величине модуля собственных значений и соответствующих им собственных векторов симметричной матрицы.

Матрицу задать следующим образом (воспользоваться программой лабораторной работы «Решение систем на основе разложения симметричных матриц»):

* недиагональные элементы *ai,j*, *i<j*, выбираются из чисел 0, –1, –2, –3, *–*4 произвольным образом; если *i>j*, то полагается *ai,j*=*aj,i*.
* *ai,i=*, 2≤*i*≤*n*;
* *a*11*=*.

Для вычислений выбрать *n* – одно из чисел в пределах от 10 до 12. В качестве языка программирования выбрать C или C++, использовать тип float.

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

Вывести на печать *uk* на итерациях 46–50 (*k=*46,…,50).

Для обоих случаев: вывести на печать приближенное λ1 для *k=*46,…,50; вычислить вектор *vk+*1–λ1*uk* (это, для проверки насколько оказались хороши вычисленные λ1 и *uk*, вектор «погрешности» *Auk*–λ1*uk*) для *k=*50, вычислить и вывести на печать кубическую норму (максимум-норму) этого вектора.

Для вычисления второго по величине модуля собственного значения использовать формулу λ2≈. Выбирать такое *i*, для которого достигается ).

Рассмотреть три случая:

1) *m=*30, λ1 берется при *k=*50 по формуле для не обязательно симметричной матрицы;

2) *m=*50, λ1 берется при *k=*50 по формуле для не обязательно симметричной матрицы;

3) *m=*50, λ1 берется при *k=*50 по формуле для симметричной матрицы.

Во всех случаях λ1 уже было вычислено ранее.

Для трех случаев: вывести на печать приближенное λ2, вычислить и вывести на печать собственный вектор *u*, соответствующий собственному значению λ2, вычислить вектор *Au*–λ2*u*, вычислить и вывести на печать кубическую норму этого вектора.

**Задание 2.** (Задание дополнительное, для повышения оценки текущей успеваемости.) Вычислительные эксперименты для получения наибольшего по модулю собственного значения и соответствующего ему собственного вектора случайно заданной матрицы четвертого порядка.

Взять матрицу (именно ее), случайным образом заданную в лабораторной работе «Метод Данилевского», и попытаться получить собственное значение и соответствующий ему собственный вектор согласно разных случаев файла «Степенной метод».

**Входные данные:**

Начальная матрица:

24 -4 -2 -2 -1 0 -4 -2 0 -2 -3 -4

-4 4 -4 0 0 -4 -2 -1 -1 -4 -2 -1

-2 -4 2 0 -4 -2 0 -4 -2 -2 0 -4

-2 0 0 2 -2 -3 -1 -1 -1 -4 -3 -1

-1 0 -4 -2 1 0 -4 -4 0 -4 -3 -2

0 -4 -2 -3 0 0.01 -4 -3 -1 -1 0 -4

-4 -2 0 -1 -4 -4 4 -2 0 -4 -3 -1

-2 -1 -4 -1 -4 -3 -2 2 -1 -4 -1 -1

0 -1 -2 -1 0 -1 0 -1 0.01 -2 -2 -2

-2 -4 -2 -4 -4 -1 -4 -4 -2 2 -1 -1

-3 -2 0 -3 -3 0 -3 -1 -2 -1 3 -2

-4 -1 -4 -1 -2 -4 -1 -1 -2 -1 -2 4

**Листинг программы:**

#include <cmath>

#include <iomanip>

#include "Functions.h"

float Norm(const std::vector<float>& vec) {

float max = 0;

for (const auto& el : vec) {

if (max < abs(el)) {

max = abs(el);

}

}

return max;

}

void Normalization(std::vector<float>& vec) {

float max = 0;

for (const auto& el : vec) {

if (el > max) {

max = el;

}

}

for (auto& el : vec) {

el /= max;

}

}

float ScalarProduct(const std::vector<float>& vec\_1, const std::vector<float>& vec\_2) {

float ret\_val = 0;

for (int i = 0; i < vec\_1.size(); i++) {

ret\_val += vec\_1[i] \* vec\_2[i];

}

return ret\_val;

}

float Sign(float number) {

if (number < 0) {

return -1;

}

if (number > 0) {

return 1;

}

return 0;

}

std::vector<float> Eigenvectors(const std::vector<float>& u, const std::vector<float>& v, float lambda) {

std::vector<float> eigenvector(u.size());

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

eigenvector[i] = v[i] - lambda \* u[i];

}

return eigenvector;

}

float CountSecondEigenValue(std::vector<float> v\_m, std::vector<float> v\_m1,

std::vector<float> u\_m, std::vector<float> u\_m1, float lambda) {

float max = 0;

int index = 0;

for(int i = 0; i < v\_m.size(); i++) {

if (abs(v\_m[i] - lambda \* u\_m1[i]) > max) {

max = abs(v\_m[i] - lambda \* u\_m1[i]);

index= i;

}

}

return (v\_m1[index] \* Norm(v\_m) - lambda \* v\_m[index]) / (v\_m[index] - lambda \* u\_m1[index]);

}

class Matrix {

public:

~Matrix() {

matrix\_.clear();

}

// создание матрицы А

Matrix() {

srand(time(NULL));

// случайное количество элементов матрицы

int n = rand() % 3 + 10;

// запоминаем числа для не диагональных элементов от -4 до 0

std::vector<int> no\_diag;

for (int i = 0; i < (n \* n - n) / 2; i++) {

no\_diag.push\_back(rand() % 5 - 4);

}

matrix\_.resize(n);

for (auto& line : matrix\_) {

line.resize(n, -6);

}

// fill matrix\_

int p = 0, sum\_1\_line = 0;

float sum\_1\_col = 0.0f;

for (int i = 0; i < matrix\_.size(); i++) {

for (int j = 0; j < matrix\_.size(); j++) {

if (i != j) {

if (matrix\_[i][j] == -6) {

matrix\_[i][j] = no\_diag[p];

matrix\_[j][i] = no\_diag[p];

p++;

}

}

if (i == 0 && j != 0) {

sum\_1\_line += matrix\_[0][j];

}

if (j == 0 && i != 0) {

sum\_1\_col += matrix\_[i][0];

}

}

}

matrix\_[0][0] = -(float)sum\_1\_line + pow(10, -2);

sum\_1\_col += matrix\_[0][0];

for (int i = 1; i < matrix\_.size(); i++) {

matrix\_[i][i] = sum\_1\_col - matrix\_[i][0];

}

matrix\_ = { { -23, -18, -35, 28 },

{ 5, 10, 5, -9 },

{ -49, 12, 25, 16 },

{ 26, -48, -42, 21} };

// write matrices in the logs.txt

LogMatrix(matrix\_);

}

void StartIterations() {

int line\_length = matrix\_.size();

std::vector<float> x\_1(line\_length), x\_2(line\_length),

v\_30, v\_31, u\_30, u\_29, v\_49, v\_50, u\_49, u\_48;

x\_2[0] = 1;

float max = 0;

for (int i = 0; i < 46; i++) {

if (i == 30) {

v\_30 = x\_2;

}

if (i == 31) {

v\_31 = x\_2;

}

Normalization(x\_2);

if (i == 29) {

u\_29 = x\_2;

}

if (i == 30) {

u\_30 = x\_2;

}

std::cout << "k = " << i << ": ";

PrintVector(x\_2);

x\_2 = MultMat(matrix\_, x\_2);

max = 0;

}

std::cout << std::endl;

float lamda\_simple = 0, lamda\_scalar\_product = 0;

for (int i = 46; i < 51; i++) {

if (i == 49) {

v\_49 = x\_2;

}

if (i == 50) {

v\_50 = x\_2;

}

x\_1 = x\_2;

Normalization(x\_1);

if (i == 49) {

u\_49 = x\_1;

}

if (i == 48) {

u\_48 = x\_1;

}

std::cout << "k = " << i << ": ";

PrintVector(x\_1);

x\_2 = MultMat(matrix\_, x\_1);

lamda\_simple = Norm(x\_2) \* Sign(Norm(x\_1));

lamda\_scalar\_product = ScalarProduct(x\_2, x\_1) / ScalarProduct(x\_1, x\_1);

}

std::vector<float> eigenvector\_simple = Eigenvectors(x\_1, x\_2, lamda\_simple);

std::vector<float> eigenvector\_scalar = Eigenvectors(x\_1, x\_2, lamda\_scalar\_product);

PrintVector(eigenvector\_simple);

std::cout << std::endl << std::endl;

PrintVector(eigenvector\_simple);

std::cout << std::endl << std::endl;

std::cout << "Eigenvalue by lambda = v / u: " << lamda\_simple << std::endl

<< "Eigenvalue by lambda <v, u> / <u, u>: " << lamda\_scalar\_product;

float eigenvalue\_30 = CountSecondEigenValue(v\_30, v\_31, u\_30, u\_29, lamda\_simple);

float eigenvalue\_50 = CountSecondEigenValue(v\_49, v\_50, u\_49, u\_48, lamda\_simple);

float eigenvalue\_50\_scalar\_eigenval = CountSecondEigenValue(v\_49, v\_50, u\_49, u\_48, lamda\_scalar\_product);

std::cout << std::endl << std::endl << eigenvalue\_30 << " " << eigenvalue\_50 << " " << eigenvalue\_50\_scalar\_eigenval << std::endl;

}

private:

std::vector<std::vector<float>> matrix\_;

std::vector<float> x\_;

};

int main() {

remove("logs.txt");

Matrix m;

std::cout << std::endl;

m.StartIterations();

return 0;

}

**Выходные данные:**

k = 46: 1 -0.16 -0.033 -0.07 0.029 0.089 -0.16 -0.06 0.029 -0.007 -0.074 -0.17

Iteration 46: division 26 scalar product 26

k = 47: 1 -0.16 -0.033 -0.07 0.029 0.089 -0.16 -0.06 0.029 -0.007 -0.074 -0.17

Iteration 47: division 26 scalar product 26

k = 48: 1 -0.16 -0.033 -0.07 0.029 0.089 -0.16 -0.06 0.029 -0.007 -0.074 -0.17

Iteration 48: division 26 scalar product 26

k = 49: 1 -0.16 -0.033 -0.07 0.029 0.089 -0.16 -0.06 0.029 -0.007 -0.074 -0.17

Iteration 49: division 26 scalar product 26

k = 50: 1 -0.16 -0.033 -0.07 0.029 0.089 -0.16 -0.06 0.029 -0.007 -0.074 -0.17

Iteration 50: division 26 scalar product 26

0 -1.9e-06 -1.7e-06 -1.5e-06 -1.6e-06 -1.4e-06 -2.4e-06 -1.7e-06 -8.3e-07 -1.6e-06 -1.3e-06 -1.4e-06

0 -1.9e-06 -1.7e-06 -1.5e-06 -1.6e-06 -1.4e-06 -2.4e-06 -1.7e-06 -8.3e-07 -1.6e-06 -1.3e-06 -1.4e-06

Check lamda\_simple: 2.4e-06

Check lamda\_scalar\_product: 2.4e-06

Check eigenvalue\_30: 46

Check eigenvalue\_50: 42

Check eigenvalue\_50\_scalar\_eigenval: 46

**Вывод:**