Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования   
«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Отчет

по лабораторной работе № 3

"Частичная проблема собственных чисел"

по дисциплине

«Вычислительные алгоритмы»

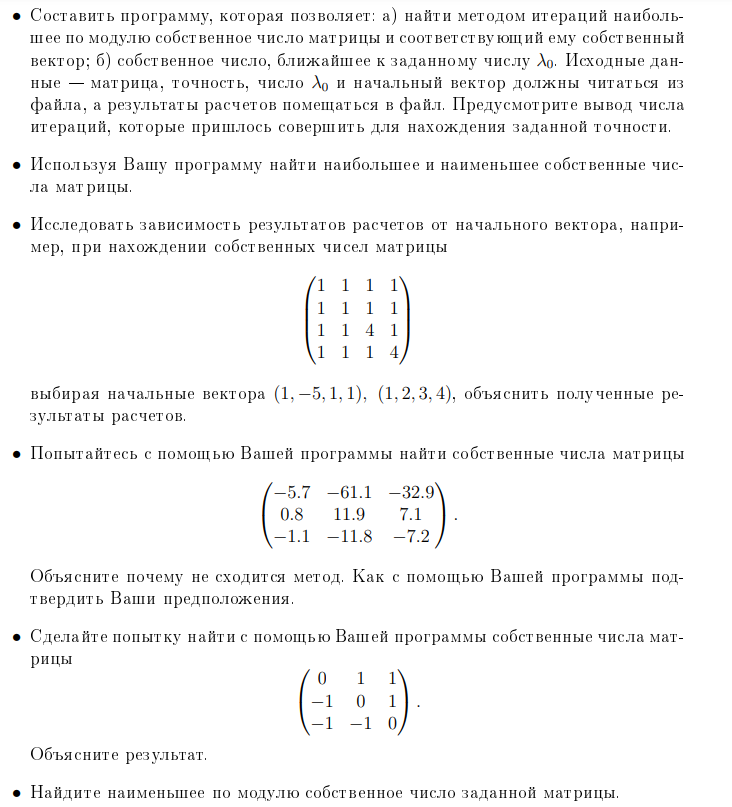
Студент гр. ПИ-92 Дьяков Д.В.

Борисов А.Е.

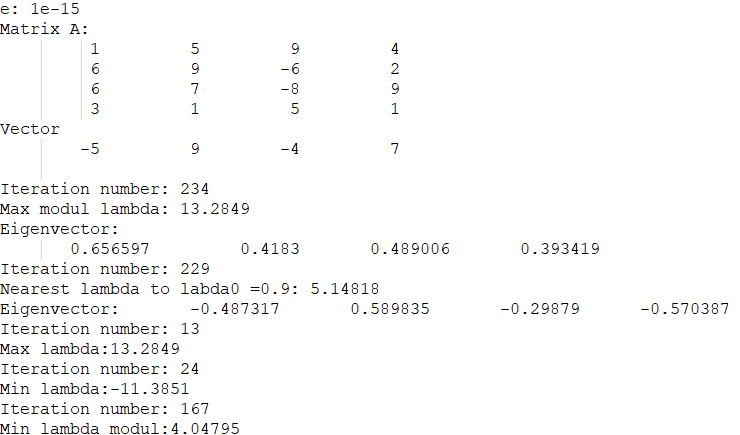
Преподаватель, к.т.н. Проскурин А.В.

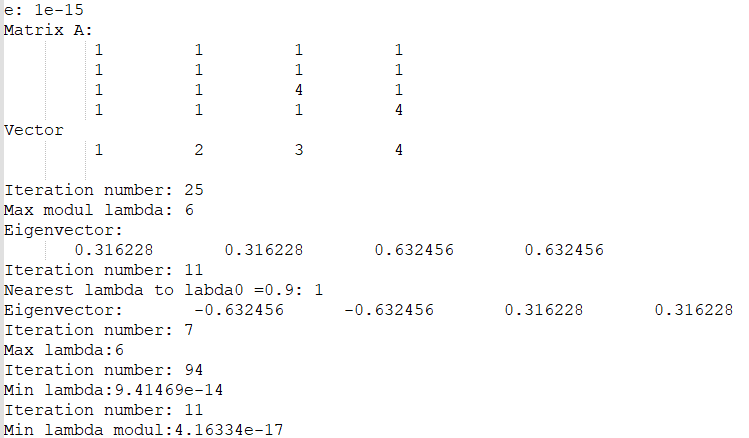
Барнаул 2022

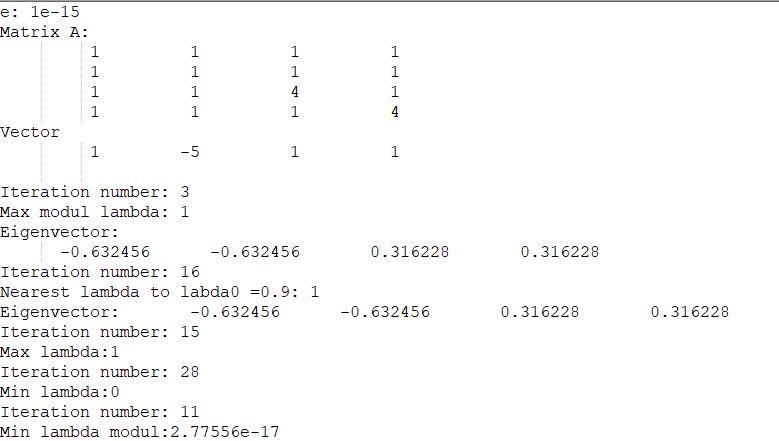
# Задание



# Тесты

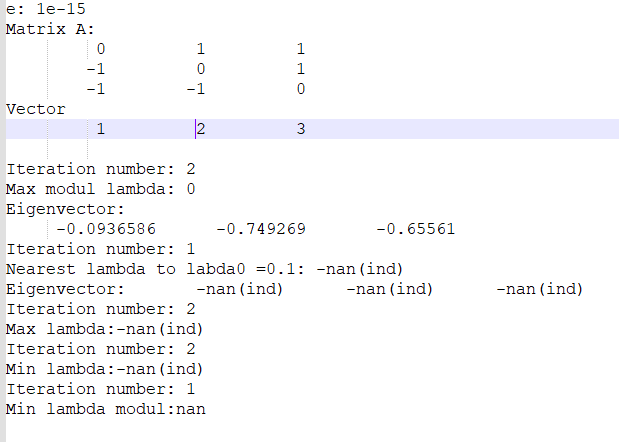
Тест 1: 

Тест 2: 



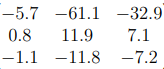
В тесте 2,3 при изменении начального приближения изменяется и наибольший модуль собственного значения. Это объясняется тем, что 1 как и 6 является модулем собственного значения и при выборе начального вектора (1, -5, 1, 1), последовательность сходится к 1 за меньшее количество итераций.

Тест 3:

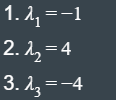


Если попытаться найти собственные числа вручную можно обнаружить, что собственные числа – комплексные числа, а программа может работать только с действительными числами, поэтому результат не сходится.

При:



Алгоритм циклится это можно объяснить тем, что матрица имеется дав наибольших по модулю собственных значения отличающихся только знаком.



# Текст программы

#include <iostream>

#include <vector>

#include <fstream>

#include <math.h>

#include <iomanip>

using Vector = std::vector<double>;

using Matrix = std::vector<Vector>;

std::pair<Vector, double> **NearestLambda**(const Matrix& mat, const Vector& x , double lambda0, double eps);

// Создание единичной матрицы

Matrix **E**(size\_t n) {

Matrix m(n, Vector(n, 0));

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

m[i][i] = 1;

return m;

}

// Умножение числа на матрицу

Matrix operator\* (double v, const Matrix& mat) {

Matrix matrix(mat);

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

for (int j = 0; j < mat.size(); ++j)

matrix[i][j] \*= v;

return matrix;

}

// Вычитание матриц

Matrix operator- (const Matrix& mat1, const Matrix& mat2) {

Matrix matrix(mat1);

for (int i = 0; i < mat1.size(); ++i)

for (int j = 0; j < mat1.size(); ++j)

matrix[i][j] = mat1[i][j] - mat2[i][j];

return matrix;

}

// скалярное произведение

double **DotProduct**(const Vector& v1, const Vector& v2)

{

double dotProduct = 0;

for (int i = 0; i < v1.size(); ++i)

dotProduct+= v1[i] \* v2[i];

return dotProduct;

}

// умножение матрицы на вектор

Vector operator \* (const Matrix& matrix, const Vector& vector) {

Vector mul(vector.size(), 0);

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

for (int j = 0; j < vector.size(); ++j)

mul[i] += matrix[i][j] \* vector[j];

return mul;

}

void **Normalize**(Vector& vector) {

double norma = sqrt( DotProduct(vector, vector) );

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

vector[i] = vector[i] / norma;

}

// Находим собственный вектор и наибольший модуль собственного значения

std::pair<Vector, double> **Solve**(const Matrix& matrix,

const Vector& startAproximation,

double eps)

{

int it = 0;

Vector x(startAproximation);

Vector prevX;

double prevLambda = 0, lambda = INFINITY;

while (std::abs(prevLambda - lambda) > eps) {

prevLambda = lambda;

prevX = x;

x = matrix \* prevX;

lambda = DotProduct(x, prevX) /

DotProduct(prevX, prevX);

Normalize(*x*);

it++;

}

std::cout << "Iteration number: " << it << std::endl;

return std::make\_pair(*x*, *lambda*);

}

int **main**()

{

setlocale(LC\_ALL, "ru\_RU.UTF-8");

freopen("output.dat", "w", stdout);

std::ifstream fin("input.dat");

if (!fin.is\_open())

return -1;

size\_t m;

float eps;

double lambda0;

fin >> m >> lambda0 >> eps;

std::cout << "e: " << eps << std::endl;

// Ввод из файла матрицы А

Matrix A(m, Vector(m));

Vector B(m);

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

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

fin >> A[i][j];

}

}

// Ввод из файла начального бриблежения

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

fin >> B[i];

}

fin.close();

// Вывод матрицы А

std::cout << "Matrix A: " << std::endl;

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

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

std::cout << std::setw(10) << A[i][j];

std::cout << std::endl;

}

// Вывод начального приблежения

std::cout << "Vector" << std::endl;

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

std::cout << std::setw(10) << B[i];

std::cout << std::endl;

std::cout << std::endl;

auto pair = Solve(A, B, eps);

std::cout << "Max modul lambda: " << pair.second << std::endl;

std::cout << "Eigenvector:" << std::endl;

for (int j = 0; j < pair.first.size(); ++j)

std::cout << std::setw(15) << pair.first[j];

std::cout << std::endl;

auto nearest = NearestLambda(A, B, lambda0, eps);

std::cout << "Nearest lambda to labda0 =" << lambda0 << ": "

<< nearest.second << std::endl << "Eigenvector: ";

for(auto& e : nearest.first)

std::cout << std::setw(15) << e;

std::cout << std::endl;

std::cout << "Max lambda:" << NearestLambda(A, B, pair.second + 0.1, eps).second << std::endl;

std::cout << "Min lambda:" << NearestLambda(A, B, -pair.second - 0.1, eps).second << std::endl;

std::cout << "Min lambda modul:" << std::abs(NearestLambda(A, B, 0.1, eps).second) << std::endl;

return 0;

}

// Получение обратной матрицы методом Гауса

Matrix **InvertMatrix**(const Matrix& mat) {

Matrix matrix(mat);

Matrix e(E(matrix.size()));

Matrix result(mat.size(), Vector(mat.size(), 0));

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

// находим главный элемент

double maxElement = 0;

int maxElementIndex = i;

for (int j = i; j < matrix.size(); ++j) {

if (maxElement < matrix[j][i]) {

maxElement = matrix[j][i];

maxElementIndex = j;

}

}

// проводим цикл исключений

for (int j = i; j < matrix.size(); ++j) {

if (j != maxElementIndex && matrix[j][i] != 0) {

double c = matrix[j][i] / maxElement;

for (int k = 0; k < matrix.size(); ++k)

matrix[j][k] -= matrix[maxElementIndex][k]\*c;

for (int k = 0; k < matrix.size(); ++k)

e[j][k] -= e[maxElementIndex][k]\*c;

}

}

// меняем строки

if (i != maxElementIndex) {

std::swap(*matrix[i]*, *matrix[maxElementIndex]*);

std::swap(*e[i]*, *e[maxElementIndex]*);

}

}

// выполняем обратный ход

for (int k = 0; k < matrix.size(); ++k) {

for (int i = result.size() - 1; i >= 0; --i) {

// S(Aij), j > i

double sum = 0;

for (int j = i + 1; j < matrix.size(); ++j)

sum += matrix[i][j] \* result[j][k];

if (std::abs(matrix[i][i]) < 0.00001)

return {};

// xi = (bi - S(Aij) ) / Aii, j > i

result[i][k] = (e[i][k] - sum) / matrix[i][i];

}

}

return result;

}

std::pair<Vector, double> **NearestLambda**(const Matrix& mat, const Vector& x,

double lambda0, double eps) {

auto matrix = mat - (lambda0 \* E(mat.size()));

Matrix inverse = InvertMatrix(matrix);

auto result = Solve(inverse, x, eps);

double nearLambda = (1. / result.second) + lambda0;

return std::make\_pair(*result.first*, *nearLambda*);

}