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»

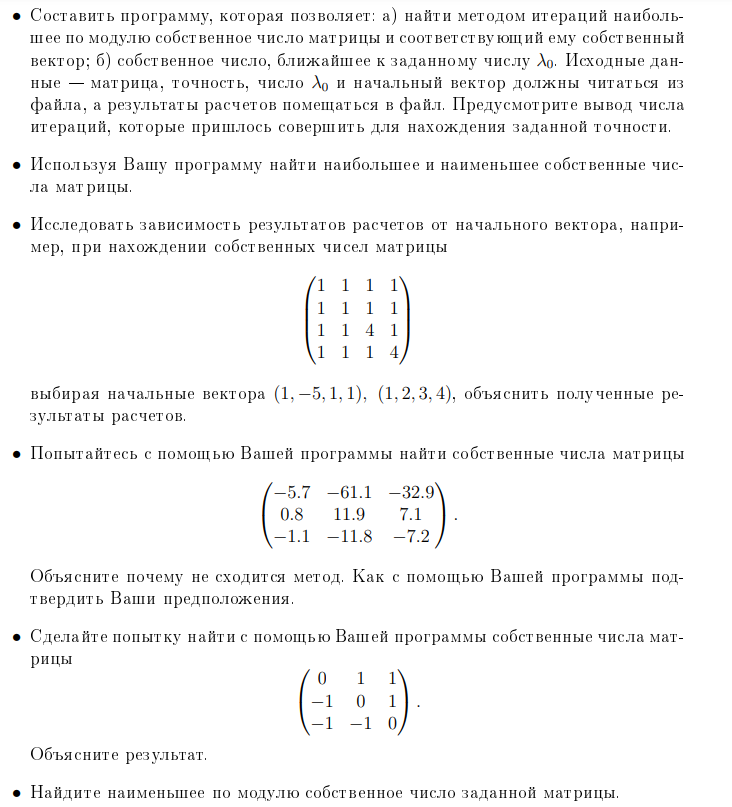
Студенты гр. ПИ-92 Шинтяпин И.И.

Шульпов В.М.

Преподаватель, к.т.н. Проскурин А.В.

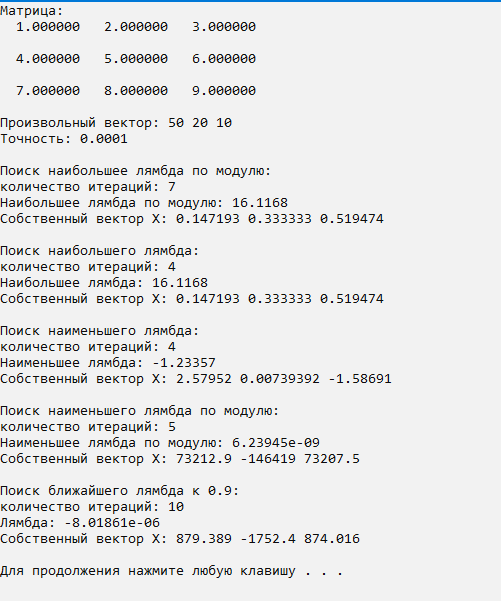
Барнаул 2022

Задание

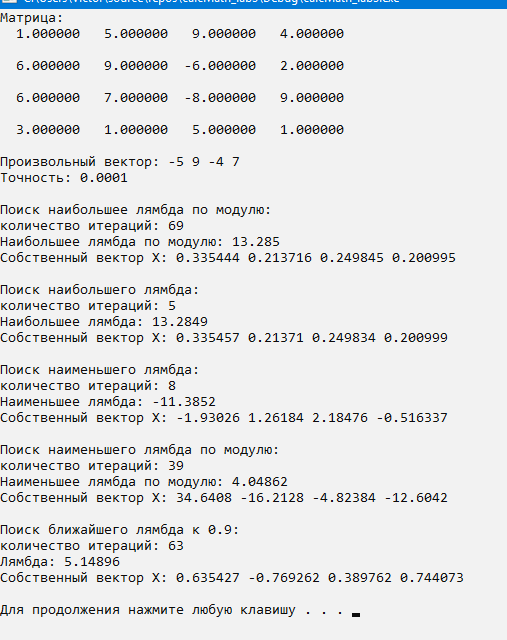


Тесты

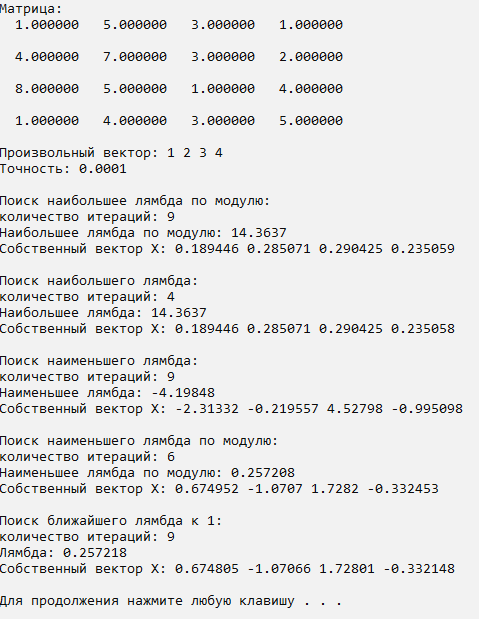
Тест 1:



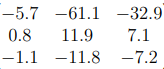
Тест 2:



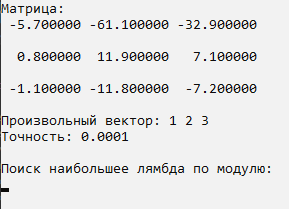
Тест 3



1. При



Программа зацикливается:

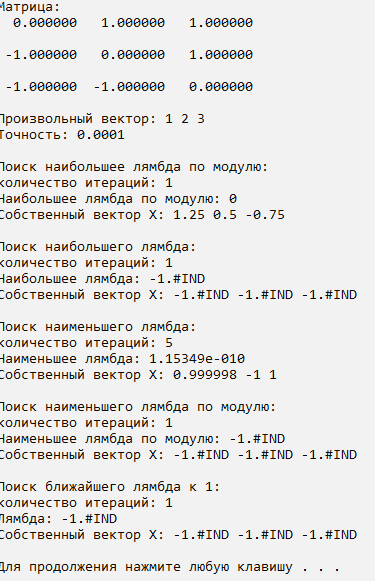


Это объясняется тем, что у матрицы два наибольших по модулю собственных значения, которые отличаются только знаком:

Лямбда1 = -1

Лямбда2 = 4

Лямбда3 = -4



Такой результат получается потому, что если попытаться решить данную задачу вручную, то можно обнаружить, что собственные числа – комплексные числа. Программа умеет работать только с действительными числами, поэтому результат не сходится

Код

#include <stdio.h>

#include <iostream>

#include <Conio.h>

#include <Windows.h>

#include <vector>

#include <fstream>

using namespace std;

void read(double\*\* matrix, double\* X, int\* n, ifstream\* fin);// функция чтения матрицы из файла

void print(double\*\* matrix, double\* X, int\* n, double\* e, FILE\* file);// функция вывода матрицы

void find\_y(double\*\* matrix, const double\* X, int\* n, double\* e, FILE\* file, double\* X\_new, double\* lambda);// функция поиска наибольшей лямбды

void mul\_matrix(double\*\* matrix, double\* X, double\* X\_previous, int\* n);// умножение вектора X на матрицу А

double scalar\_mul(double\* vector\_one, double\* vector\_two, int\* n);// скалярное умножение векторов

void normirovanie(double\* vector, int\* n);// нормирование вектора

void invert\_matrix(double\*\* matrix, int\* n);// поиск обратной матрицы методом Гаусса

double\*\* E(int\* n);// формирование единичной матрицы, заданного размера

void near\_lambda(double\*\* matrix, int\* n, double\* X, double\* lambda0, double\* e, double\* X\_new, double\* lambda, FILE\* file);//поиск лямбды ближйшей к заданному числу

void mult\_matrix\_by\_num(double\*\* matrix, int\* n, double mult);// умножение матрицы на число

void subtract\_matrix\_from\_matrix(double\*\* matrix1, double\*\* matrix2, int\* n);// вычиатние матриц

// функция чтения матрицы из файла

void read(double\*\* matrix, double\* X, int\* n, ifstream\* fin)

{

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

{

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

{

\*fin >> matrix[i][j];

}

}

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

{

\*fin >> X[i];

}

}

// функция вывода матрицы в консоль и в файл

void print(double\*\* matrix, double\* X, int\* n, double\* e, FILE\* file)

{

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

{

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

{

printf("%10lf ", matrix[i][j]);

fprintf(file, "%10lf ", matrix[i][j]);

}

printf("\n\n");

fprintf(file, "\n\n");

}

cout << "Произвольный вектор: ";

fprintf(file, "Произвольный вектор: ");

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

{

cout << X[i] << " ";

fprintf(file, "%lf ", X[i]);

}

cout << endl << "Точность: " << \*e << endl;

fprintf(file, "\nТочность: %lf", \*e);

cout << endl;

fprintf(file, "\n");

}

// умножение матрицы на число

void mult\_matrix\_by\_num(double\*\* matrix, int\* n, double\* mult) {

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

{

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

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

}

}

}

// вычитание матрицы из другой матрицы

void subtract\_matrix\_from\_matrix(double\*\* matrix1, double\*\* matrix2, int\* n) {

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

{

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

matrix2[i][j] = matrix1[i][j] - matrix2[i][j];

}

}

}

// алгоритм вычиления лямбда наибольшего по модулю и собственного вектора

void find\_y(double\*\* matrix, const double\* X, int\* n, double\* e, FILE\* file, double\* X\_new, double\* lambda)

{

double\* X\_previous = new double[\*n];// вектор предыдущей итерации

double y = 0;// значение лямбда

double y\_previous;// значение лямбда предыдущей итерации

double current\_e;// текущая точность

int k = 0;// кол-во итераций

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

{

X\_new[i] = X[i];

}

//цикл с постусловием пока не найдена лямбда необходимой точности

do {

k++;// следующая итерация

for (int i = 0; i < \*n; i++)// запоминание текущего вектора в предыдущий

{

X\_previous[i] = X\_new[i];

}

y\_previous = y;//запоминание текущей лямбды

mul\_matrix(matrix, X\_new, X\_previous, n);// умножение вектора на матрицу А

y = scalar\_mul(X\_new, X\_previous, n) / scalar\_mul(X\_previous, X\_previous, n);// деление скалярного произведение векторов

current\_e = abs(y - y\_previous);

normirovanie(X\_new, n);// нормирование вектора

} while (current\_e >= \*e);// проверка условия выхода из цикла

cout << "количество итераций: " << k << endl;

fprintf(file, "количество итераций: %d", k);

\*lambda = y;

delete[] X\_previous;

}

// умножение вектора X на матрицу А

void mul\_matrix(double\*\* matrix, double\* X, double\* X\_previous, int\* n)

{

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

{

X[i] = 0;

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

X[i] += matrix[i][j] \* X\_previous[j];

}

}

// скалярное произведение векторов

double scalar\_mul(double\* vector\_one, double\* vector\_two, int\* n)

{

double answer = 0;// сумма скалярного произведения

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

{

answer += vector\_one[i] \* vector\_two[i];

}

return answer;

}

//нормирование вектора

void normirovanie(double\* vector, int\* n)

{

double sum = 0;// норма

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

{

sum += vector[i];

}

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

{

vector[i] /= sum;

}

}

// получение единичной матрицы n\*n

double\*\* E(int\* n) {

double\*\* e\_matrix = new double\* [\*n];// выделение памяти под массив матрицы А

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

{

e\_matrix[i] = new double[\*n];

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

e\_matrix[i][j] = (i == j) ? 1 : 0;

}

}

return e\_matrix;

}

// Получение обратной матрицы методом Гаусса

void invert\_matrix(double\*\* matrix, int\* n) {

double\*\* matrix\_e = E(n);

double\*\* copy\_matrix = new double\* [\*n];// выделение памяти под массив матрицы А

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

{

copy\_matrix[i] = new double[\*n];

}

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

{

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

{

copy\_matrix[i][j] = matrix[i][j];

}

}

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

// находим главный элемент

double maxElement = 0;

int maxElementIndex = i;

for (int j = i; j < \*n; ++j) {

if (maxElement < copy\_matrix[j][i]) {

maxElement = copy\_matrix[j][i];

maxElementIndex = j;

}

}

// проводим цикл исключений

for (int j = i; j < \*n; ++j) {

if (j != maxElementIndex && copy\_matrix[j][i] != 0) {

double c = copy\_matrix[j][i] / maxElement;

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

copy\_matrix[j][k] -= copy\_matrix[maxElementIndex][k] \* c;

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

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

}

}

// замена строк

if (i != maxElementIndex) {

swap(copy\_matrix[i], copy\_matrix[maxElementIndex]);

swap(matrix\_e[i], matrix\_e[maxElementIndex]);

}

}

// обратный ход

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

for (int i = \*n - 1; i >= 0; --i) {

// S(Aij), j > i

double sum = 0;

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

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

if (std::abs(copy\_matrix[i][i]) < 0.00001)

{

return;

}

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

matrix[i][k] = (matrix\_e[i][k] - sum) / copy\_matrix[i][i];

}

}

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

{

delete[] copy\_matrix[i];

}

delete[] copy\_matrix;

}

void near\_lambda(double\*\* matrix, int\* n, double\* X, double\* lambda0, double\* e, double\* X\_new, double\* lambda, FILE\* file)

{

double\*\* matrix\_B = E(n);// доп матрица B (единичная)

mult\_matrix\_by\_num(matrix\_B, n, lambda0);// умножение матрицы B на лямбда0

subtract\_matrix\_from\_matrix(matrix, matrix\_B, n);// вычитание матрицы B из начальной матрицы, результат в B

invert\_matrix(matrix\_B, n);// поиск обртаной матрицы B

find\_y(matrix\_B, X, n, e, file, X\_new, lambda);// вычисление наибольшего лямбда относительно матрицы B

\*lambda = (1. / (\*lambda)) + \*lambda0;// нахождение ближайшего лямбда к заданной

// очистка памяти

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

{

delete[] matrix\_B[i];

}

delete[] matrix\_B;

}

int main()

{

SetConsoleCP(1251);// подключение русскоязычного ввода/вывода

SetConsoleOutputCP(1251);

system("Color F0");

int n;// размерность матрицы

double e;// точность вычислений

double\*\* matrix = nullptr;// указатель на динамический массив матрицы

double\* X = nullptr;// указатель на произвольный динамический вектор

double\* X\_new = nullptr;// массив для хранения собственных векторов

double\* lambda0 = new double;// заданное лямбда

double\* lambda = new double;// найденная лямбда

// открытие файла для чтения

ifstream fin("input.dat");

if (fin.is\_open() == false)

{

cout << "Ошибка открытия файла input.dat\nзавершение работы" << endl;

\_getch();

return 1;

}

// чтение данных из файла

fin >> \*lambda0;// чтение заданной лямбды

fin >> e;// чтение точности

fin >> n;//чтение размерности

matrix = new double\* [n];// выделение памяти под массив матрицы А

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

{

matrix[i] = new double[n];

}

X = new double[n];// выделение памяти под произвольный вектор

X\_new = new double[n];// собственный вектор

read(matrix, X, &n, &fin);// заполнение матрицы

fin.close();// закрытие файла

FILE\* file = fopen("output.dat", "w");

if (file == NULL)

{

cout << "Ошибка открытия файла output.dat\nзавершение работы" << endl;

\_getch();

return 1;

}

cout << "Матрица:" << endl;

fprintf(file, "Матрица:\n");

print(matrix, X, &n, &e, file);// вывод матрицы в консоль

cout << "Поиск наибольшее лямбда по модулю:" << endl;

fprintf(file, "Поиск наибольшее лямбда по модулю:\n");

find\_y(matrix, X, &n, &e, file, X\_new, lambda);// поиск наибольшего лямбда

cout << "Наибольшее лямбда по модулю: " << \*lambda << endl;

fprintf(file, "Наибольшее лямбда по модулю: %lf\n", \*lambda);

cout << "Собственный вектор X: ";

fprintf(file, "Собственный вектор X: ");

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

{

cout << X\_new[i] << " ";

fprintf(file, "%lf ", X\_new[i]);

}

cout << endl << endl;

fprintf(file, "\n\n");

double temp\_lambda = \*lambda + 0.1;

double temp\_lambda1 = -\*lambda - 0.1;

double temp\_lambda2 = 0.1;

// наибольшее лямбда

cout << "Поиск наибольшего лямбда:" << endl;

fprintf(file, "Поиск наибольшего лямбда:\n");

near\_lambda(matrix, &n, X, &temp\_lambda, &e, X\_new, lambda, file);

cout << "Наибольшее лямбда: " << \*lambda << endl;

fprintf(file, "Наибольшее лямбда: %lf\n", \*lambda);

cout << "Собственный вектор X: ";

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

{

cout << X\_new[i] << " ";

fprintf(file, "%lf ", X\_new[i]);

}

cout << endl << endl;

fprintf(file, "\n\n");

// наименьшее лямбда

cout << "Поиск наименьшего лямбда:" << endl;

fprintf(file, "Поиск наименьшего лямбда:\n");

near\_lambda(matrix, &n, X, &temp\_lambda1, &e, X\_new, lambda, file);

cout << "Наименьшее лямбда: " << \*lambda << endl;

fprintf(file, "Наименьшее лямбда: %lf\n", \*lambda);

cout << "Собственный вектор X: ";

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

{

cout << X\_new[i] << " ";

fprintf(file, "%lf ", X\_new[i]);

}

cout << endl << endl;

fprintf(file, "\n\n");

// наименьшее лямбда по модулю

cout << "Поиск наименьшего лямбда по модулю:" << endl;

fprintf(file, "Поиск наименьшего лямбда по модулю:\n");

near\_lambda(matrix, &n, X, &temp\_lambda2, &e, X\_new, lambda, file);

cout << "Наименьшее лямбда по модулю: " << abs(\*lambda) << endl;

fprintf(file, "Наименьшее лямбда по модулю: %lf\n", abs(\*lambda));

cout << "Собственный вектор X: ";

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

{

cout << X\_new[i] << " ";

fprintf(file, "%lf ", X\_new[i]);

}

cout << endl << endl;

fprintf(file, "\n\n");

//ближайшее лямбда к лямбда0

cout << "Поиск ближайшего лямбда к " << \*lambda0 << ": " << endl;

fprintf(file, "Поиск ближайшего лямбда к &lf:\n", \*lambda0);

near\_lambda(matrix, &n, X, lambda0, &e, X\_new, lambda, file);

cout << "Лямбда: " << \*lambda << endl;

fprintf(file, "Лямбда: %lf\n", \*lambda);

cout << "Собственный вектор X: ";

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

{

cout << X\_new[i] << " ";

fprintf(file, "%lf ", X\_new[i]);

}

cout << endl << endl;

fprintf(file, "\n\n");

//очистка памяти

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

{

delete[] matrix[i];

}

delete[] matrix;

delete[] X;

delete[] X\_new;

delete lambda0;

delete lambda;

fclose(file);

system("pause");

return 0;

}