**Московский авиационный институт**

# (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Дисциплина «Численные методы»

# Лабораторная работа №1

Тема: вычислительные методы линейной алгебры

|  |  |
| --- | --- |
| Студент: | Глушатов И.С. |
| Группа: | М8О-307Б-19 |
| Преподаватель: | Ревизников Д. Л. |
| Дата: |  |
| Оценка: |  |

**Лабораторная работа № 1.1**

**Задание**: реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

Изображение выглядит как текст

Автоматически созданное описание

**Листинг**

#include <iostream>

#include <initializer\_list>

#include <vector>

void print() {

std::cout << std::endl;

}

template<class T>

void print(T obj) {

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

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::vector<T>& v) {

for (auto& a : v) {

out << a << " ";

}

return out;

}

template <class T>

class Matrix {

static constexpr double MECH\_EPS = 0.0000001;

std::vector<std::vector<T>> matrix;

std::vector<Matrix<T>> plu;

public:

Matrix(size\_t \_n) {

matrix = std::vector<std::vector<T>>(\_n, std::vector<T>(\_n, T()));

}

Matrix(size\_t \_n, std::vector<std::vector<T>>& matr) : Matrix(\_n) {

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

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

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

}

}

}

Matrix(size\_t \_n, std::initializer\_list<T> list) : Matrix(\_n) {

if (list.size() != \_n \* \_n) {

throw "error";

}

auto it = list.begin();

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

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

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

it++;

}

}

}

size\_t size() const {

return matrix.size();

}

void SwapLines(size\_t i, size\_t j) {

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

std::swap(matrix[i][k], matrix[j][k]);

}

}

void SwapColumns(size\_t i, size\_t j) {

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

std::swap(matrix[k][i], matrix[k][j]);

}

}

Matrix<T> E(size\_t \_n) {

Matrix<T> result(\_n);

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

result[i][i] = 1;

}

return result;

}

std::vector<Matrix<T>> LUFactorizing(int\* count = nullptr) {

Matrix<T> P = E(size());

Matrix<T> L = E(size());

Matrix<T> U(size(), matrix);

/\*std::cout << P << std::endl;

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

std::cout << U << std::endl;\*/

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

// 1. Находим строку с максимальным по модулю элементом.

{

size\_t k = i;

T max = std::abs(U[i][i]);

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

if (std::abs(U[j][i]) > max) {

max = std::abs(U[j][i]);

k = j;

}

}

if (U[k][i] == 0) {

continue;

}

// 2. Меняем строки в U и обновляем L.

if (k != i) {

P.SwapColumns(i, k);

L.SwapLines(i, k);

L.SwapColumns(i, k);

U.SwapLines(i, k);

if (count != nullptr) (\*count) += 1;

}

}

// 3. Алгоритм Гаусса

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

double koef = U[j][i] / U[i][i];

U[j][i] = 0;

L[j][i] = koef;

for (size\_t t = i + 1; t < size(); t++) {

U[j][t] -= koef \* U[i][t];

}

}

}

/\*std::cout << P << std::endl;

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

std::cout << U << std::endl;\*/

return std::vector<Matrix<T>>({ P, L, U });

}

std::vector<T> Solve(const std::vector<T>& b) {

// A \* x = b => P \* L \* U \* x = b => L \* U \* x = P^(-1) \* b = P^(T) \* b

if (b.size() != size()) throw "размерность не совпадает";

// 1. Делаем LU - разложение

if (plu.size() == 0) {

plu = LUFactorizing();

}

// 2. Вычисляем P^(T) \* b = b \* P = y

auto y = b \* plu[0];

// 3. Вычисляем L \* z = y;

std::vector<T> z(size(), T());

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

z[i] = y[i];

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

z[i] -= plu[1][i][j] \* z[j];

}

z[i] /= plu[1][i][i];

}

// 4. Вычисляем U \* x = z

std::vector<T> x(size(), T());

for (long i = size() - 1; i >= 0; i--) {

x[i] = z[i];

for (long j = i + 1; j < size(); j++) {

x[i] -= plu[2][i][j] \* x[j];

}

x[i] /= plu[2][i][i];

}

return x;

}

double Determinant() {

int count = 0;

auto p = LUFactorizing(&count);

double result = 1;

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

result \*= p[2][i][i];

}

return count % 2 == 0 ? result : -result;

}

Matrix<T> Reverse() {

Matrix<T> result(size());

std::vector<T> b(size(), 0);

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

b[i] = 1;

auto res = Solve(b);

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

result[j][i] = res[j];

}

b[i] = 0;

}

return result;

}

Matrix<T>& operator= (std::initializer\_list<T> list) {

if (list.size() != size() \* size()) {

throw "error";

}

auto it = list.begin();

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

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

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

it++;

}

}

return \*this;

}

friend Matrix<T> operator\*(const Matrix<T>& m1, const Matrix<T>& m2) {

std::vector<std::vector<T>> vec(m1.size(), std::vector<T>(m1.size(), T()));

for (size\_t i = 0; i < m1.size(); i++) {

for (size\_t j = 0; j < m1.size(); j++) {

for (size\_t k = 0; k < m1.size(); k++) {

vec[i][j] +=

m1[i][k] \*

m2[k][j];

}

if (std::abs(vec[i][j]) < 2 \* MECH\_EPS) vec[i][j] = 0;

}

}

return Matrix<T>(m1.size(), vec);

}

friend std::vector<T> operator\*(const Matrix<T>& m1, const std::vector<T>& m2) {

if (m1.size() != m2.size()) {

throw "bad thing";

}

std::vector<T> result(m1.size(), T());

for (size\_t i = 0; i < m1.size(); i++) {

for (size\_t j = 0; j < m1.size(); j++) {

result[i] += m1[i][j] \* m2[j];

}

}

return result;

}

friend std::vector<T> operator\*(const std::vector<T>& m2, const Matrix<T>& m1) {

if (m1.size() != m2.size()) {

throw "bad thing";

}

std::vector<T> result(m1.size(), T());

for (size\_t i = 0; i < m1.size(); i++) {

for (size\_t j = 0; j < m1.size(); j++) {

result[i] += m1[j][i] \* m2[j];

}

}

return result;

}

std::vector<T>& operator[](const size\_t i) {

return matrix[i];

}

std::vector<T> operator[](const size\_t i) const {

return matrix[i];

}

friend std::ostream& operator<< (std::ostream& out, const Matrix<T>& matr) {

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

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

out << matr.matrix[i][j] << ' ';

}

out << std::endl;

}

return out;

}

};

void Task\_1\_1() {

Matrix<double> m(4, {

1, 2, -1, -7,

8, 0, -9, -3,

2, -3, 7, 1,

2, -5, -6, 8,

});

// 0. Исходная матрица

print("0. Matrix");

print(m);

print("Vector b: ");

print(std::vector<double>{ -23, 39, -7, 30 });

print();

// 1. LU - разложение

print("1. LU - Decomposition");

auto res = m.LUFactorizing();

for (auto& mm : res) print(mm);

print(res[0] \* res[1] \* res[2]);

// 2. Решение системы

print("2. System Solution");

auto solution = m.Solve({-23, 39, -7, 30});

print(solution);

print();

// 3. Определитель

print("3. Determinant");

auto determinant = m.Determinant();

print(determinant);

print();

// 4. Обратная матрица

print("4. Inverse Matrix");

auto reverse = m.Reverse();

print(reverse);

}

int main()

{

Task\_1\_1();

return 0;

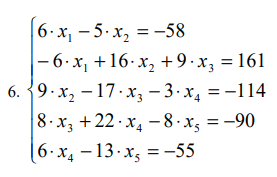
}

Изображение выглядит как текст

Автоматически созданное описание

**Лабораторная работа № 1.2**

**Задание**: реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.



**Листинг**

#include <iostream>

#include <vector>

void print() {

std::cout << std::endl;

}

template<class T>

void print(T obj) {

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

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::vector<T>& v) {

for (auto& a : v) {

out << a << " ";

}

return out;

}

template<class T>

std::vector<T> Solve(std::vector<std::vector<T>>& abc, std::vector<T> d) {

size\_t dimension = d.size();

std::vector<T> result(dimension);

std::vector<T> P(dimension, 0);

std::vector<T> Q(dimension, 0);

P[0] = -(abc[2][0] / abc[1][0]);

Q[0] = (d[0] / abc[1][0]);

for (size\_t i = 1; i < dimension - 1; i++) {

P[i] = -(abc[2][i] / (abc[1][i] + abc[0][i - 1] \* P[i - 1]));

Q[i] = ((d[i] - abc[0][i - 1] \* Q[i - 1]) / (abc[1][i] + abc[0][i - 1] \* P[i - 1]));

}

result[dimension - 1] = ((d[dimension - 1] - abc[0][dimension - 2] \* Q[dimension - 2]) / (abc[1][dimension - 1] + abc[0][dimension - 2] \* P[dimension - 2]));

for (size\_t i = 0; i < dimension - 1; i++) {

size\_t k = dimension - 2 - i;

result[k] = P[k] \* result[k + 1] + Q[k];

}

return result;

}

void Task\_1\_2() {

std::vector<std::vector<double>> abc {

{ -6, 9, 8, 6},

{ 6, 16, -17, 22, -13},

{-5, 9, -3, -8}

};

std::vector<double> d {-58, 161, -114, -90, -55};

// 0. Входные данные

print("0. Entry data");

for (int i = 0; i < 3; i++) {

print(std::vector<std::string>{"a:", "b:", "c:"}[i]);

print(abc[i]);

}

print("d:");

print(d);

print();

// 1. Решение

print("Solution");

print(Solve(abc, d));

}

int main()

{

Task\_1\_2();

return 0;

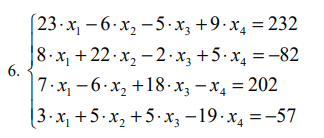
}

Изображение выглядит как текст

Автоматически созданное описание

**Лабораторная работа № 1.3**

**Задание**: реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.



**Листинг**

#include <iostream>

#include <vector>

using namespace std;

void print() {

std::cout << std::endl;

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::vector<T>& v) {

for (auto& a : v) {

out << a << " ";

}

return out;

}

template<class T>

void print(T obj) {

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

}

template<class T>

class Matrix {

vector<vector<T>> matrix;

public:

Matrix(size\_t \_n) {

matrix = vector<vector<T>>(\_n, vector<T>(\_n, T()));

}

Matrix(size\_t \_n, vector<T> v) : Matrix(\_n) {

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

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

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

}

}

}

void SwapColumns(size\_t i, size\_t j) {

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

std::swap(matrix[k][i], matrix[k][j]);

}

}

void SwapLines(size\_t i, size\_t j) {

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

std::swap(matrix[i][k], matrix[j][k]);

}

}

size\_t inline size() {

return matrix.size();

}

vector<T>& operator[] (size\_t i) {

return matrix[i];

}

vector<T> operator[] (size\_t i) const {

return matrix[i];

}

vector<T> operator\* (vector<T> v) {

vector<T> result(size(), T());

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

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

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

}

}

return result;

}

Matrix<T> operator+ (const Matrix<T>& m) {

Matrix<T> result(size());

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

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

result[i][j] = matrix[i][j] + m[i][j];

}

}

return result;

}

friend ostream& operator<<(ostream& out, Matrix<T> m) {

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

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

out << m[i][j] << " ";

}

out << endl;

}

return out;

}

};

template<class T>

vector<T> operator+ (vector<T> v1, vector<T> v2) {

vector<T> result(v1.size(), T());

for (size\_t i = 0; i < v1.size(); i++) {

result[i] = v1[i] + v2[i];

}

return result;

}

template<class T>

vector<T> operator- (vector<T>& v1, vector<T>& v2) {

vector<T> result(v1.size(), T());

for (size\_t i = 0; i < v1.size(); i++) {

result[i] = v1[i] - v2[i];

}

return result;

}

template<class T>

T Norm(vector<T> v) {

T max = std::abs(v[0]);

for (T& e : v) {

max = std::max(max, std::abs(e));

}

return max;

}

template<class T>

T Norm(Matrix<T> A) {

T max = -1;

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

T sum = 0;

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

sum += std::abs(A[i][j]);

}

max = std::max(max, std::abs(sum));

}

return max;

}

void DebugSolves(int type) {

static int count1 = 0;

static int count2 = 0;

if (type == 1) {

count1++;

}

else if (type == 2) {

count2++;

}

else if (type == 3) {

cout << "Yacobi iters: " << count1 << endl;

cout << "Zeidel iters: " << count2 << endl;

}

}

template<class T>

vector<T> SolveYacobi(Matrix<T> A, vector<T> b, T eps) {

vector<T> x(b);

// x = a2 + a1 \* x

Matrix<T> a1(A.size());

vector<T> a2(A.size(), T());

// 1. Вычисление a2

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

a2[i] = b[i] / A[i][i];

}

// 2. Вычисление a1

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

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

if (i == j) {

a1[i][j] = 0;

}

else {

a1[i][j] = -(A[i][j] / A[i][i]);

}

}

}

if (Norm(a1) >= 1) {

print("We have problems, cause a1 matrix's norm >= 1");

return vector<T>();

}

// 3. Итерации

T a1\_norm = Norm(a1);

vector<T> next\_x = a2 + a1 \* x;

T eps\_k = (a1\_norm) / (1 - a1\_norm) \* (Norm(next\_x - x));

while (eps\_k > eps) {

DebugSolves(1);

x = next\_x;

next\_x = a2 + (a1 \* next\_x);

eps\_k = (a1\_norm) / (1 - a1\_norm) \* (Norm(next\_x - x));

}

return next\_x;

}

template<class T>

vector<T> SolveZeidel(Matrix<T> A, vector<T> b, T eps) {

vector<T> x(b);

// x = a1 \* x + a2 \* x + b

Matrix<T> a(A.size());

Matrix<T> a2(A.size());

vector<T> new\_b(A.size(), T());

// 1. Вычисление new\_b

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

new\_b[i] = b[i] / A[i][i];

}

// 2. Вычисление a1 и a2

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

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

if (i == j) {

a[i][j] = 0;

}

else {

a[i][j] = -(A[i][j] / A[i][i]);

}

if (i < j) {

a2[i][j] = -(A[i][j] / A[i][i]);

}

}

}

if (Norm(a) >= 1) {

print("We have problems, cause a1 matrix's norm >= 1");

return vector<T>();

}

// 3. Итерации

T a\_norm = Norm(a);

T a2\_norm = Norm(a2);

vector<T> next\_x = new\_b + a \* x;

T eps\_k = (a2\_norm) / (1 - a\_norm) \* (Norm(next\_x - x));

while (eps\_k > eps) {

DebugSolves(2);

x = next\_x;

// Зейдельское улучшение

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

T buf = new\_b[i];

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

buf += a[i][j] \* next\_x[j];

}

for (size\_t j = i + 1; j < A.size(); j++) {

buf += a[i][j] \* next\_x[j];

}

next\_x[i] = buf;

}

eps\_k = (a2\_norm) / (1 - a\_norm) \* (Norm(next\_x - x));

}

return next\_x;

}

int main()

{

Matrix<double> m1 (4, {

6, -3, 19, -27,

-3, 19, -27, 115,

19, -27, 115, -243,

-27, 115, -243, 859,

});

auto m2 = vector<double>{ 11.6604100000000, 16.7085700000000, 33.6320500000000, 59.0361700000000 };

cout << SolveYacobi(m1, m2, 0.001) << endl;

cout << SolveZeidel(m1, m2, 0.001) << endl;

DebugSolves(3);

return 0;

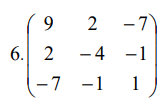
}

Изображение выглядит как текст

Автоматически созданное описание

**Лабораторная работа № 1.4**

**Задание**: реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.



**Листинг**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

void print() {

std::cout << std::endl;

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::vector<T>& v) {

for (auto& a : v) {

out << a << " ";

}

return out;

}

template<class T>

void print(T obj) {

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

}

template<class T>

class Matrix {

vector<vector<T>> matrix;

public:

Matrix(size\_t \_n) {

matrix = vector<vector<T>>(\_n, vector<T>(\_n, T()));

}

Matrix(size\_t \_n, vector<T> v) : Matrix(\_n) {

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

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

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

}

}

}

size\_t inline size() {

return matrix.size();

}

vector<T>& operator[] (size\_t i) {

return matrix[i];

}

vector<T> operator[] (size\_t i) const {

return matrix[i];

}

Matrix<T> operator\* (const Matrix<T>& m) {

Matrix<T> result(size());

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

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

result[i][j] = T();

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

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

}

}

}

return result;

}

friend ostream& operator<<(ostream& out, Matrix<T> m) {

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

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

out << m[i][j] << " ";

}

out << endl;

}

return out;

}

};

template<class T>

T Norm(Matrix<T> A) {

T sum = T();

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

for (size\_t j = i + 1; j < A.size(); j++) {

sum += A[i][j] \* A[i][j];

}

}

return std::sqrt(sum);

}

template<class T>

void JEA(Matrix<T> A, T eps) {

Matrix<T> Eigenvectors(A.size());

Matrix<T> U(A.size());

Matrix<T> U\_trans(A.size());

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

Eigenvectors[i][i] = 1;

U[i][i] = 1;

U\_trans[i][i] = 1;

}

while (Norm(A) > eps) {

T max = std::abs(A[0][1]);

size\_t l = 0, m = 1;

// 1. Поиск максимального по модулю недиагонального элемента

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

for (size\_t j = i + 1; j < A.size(); j++) {

if (std::abs(A[i][j]) > max) {

max = std::abs(A[i][j]);

l = i;

m = j;

}

}

}

// 2. Вычисление угла поворота

double phi = 0.5 \* (std::atan(2 \* A[l][m] / (A[l][l] - A[m][m])));

// 3. Составление матрицы поворота

U[l][l] = std::cos(phi);

U[m][m] = std::cos(phi);

U[l][m] = -std::sin(phi);

U[m][l] = std::sin(phi);

U\_trans[l][l] = std::cos(phi);

U\_trans[m][m] = std::cos(phi);

U\_trans[l][m] = std::sin(phi);

U\_trans[m][l] = -std::sin(phi);

// 4. Поворот

A = U\_trans \* A \* U;

Eigenvectors = Eigenvectors \* U;

U[l][l] = 1;

U[m][m] = 1;

U[l][m] = 0;

U[m][l] = 0;

U\_trans[l][l] = 1;

U\_trans[m][m] = 1;

U\_trans[l][m] = 0;

U\_trans[m][l] = 0;

}

// print(A);

// Печать результата

print("Eigenvalues:");

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

print(A[i][i]);

}

print();

print("Eigenvectors:");

print(Eigenvectors);

}

int main()

{

Matrix<double> m(3, {

9, 2, -7,

2, -4, -1,

-7, -1, 1

});

print("Matrix:");

print(m);

JEA(m, 0.000001);

return 0;

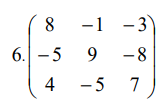
}

Изображение выглядит как текст

Автоматически созданное описание

**Лабораторная работа № 1.5**

**Задание**: реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.



**Листинг**

#include <iostream>

#include <vector>

#include <cmath>

#include <functional>

#include <iomanip>

using namespace std;

void print() {

std::cout << std::endl;

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::vector<T>& v) {

for (auto& a : v) {

out << a << " ";

}

return out;

}

template<class T>

std::ostream& operator<<(std::ostream& out, const std::pair<T, T>& v) {

return out << "{" << v.first << ", " << v.second << "}";

}

template<class T>

void print(T obj) {

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

}

template<class T, class ...Args>

void print(T obj, Args... args) {

std::cout << obj;

print(args...);

}

template<class T>

class Matrix {

vector<vector<T>> matrix;

public:

Matrix(size\_t \_n) {

matrix = vector<vector<T>>(\_n, vector<T>(\_n, T()));

}

Matrix(size\_t \_n, vector<T> v) : Matrix(\_n) {

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

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

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

}

}

}

size\_t inline size() {

return matrix.size();

}

Matrix<T> Transpose() {

Matrix<T> result(size());

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

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

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

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

}

}

return result;

}

vector<T>& operator[] (size\_t i) {

return matrix[i];

}

vector<T> operator[] (size\_t i) const {

return matrix[i];

}

Matrix<T> operator\* (const Matrix<T>& m) {

Matrix<T> result(size());

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

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

result[i][j] = T();

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

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

}

}

}

return result;

}

Matrix<T> operator- (const Matrix<T>& m) {

Matrix<T> result(size());

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

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

result[i][j] = matrix[i][j] - m[i][j];

}

}

return result;

}

Matrix<T> operator\* (const T& m) {

Matrix<T> result(size());

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

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

result[i][j] = matrix[i][j] \* m;

}

}

return result;

}

friend ostream& operator<<(ostream& out, Matrix<T> m) {

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

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

out << m[i][j] << " ";

}

out << endl;

}

return out;

}

};

template<class T>

T Norm(Matrix<T> A) {

T sum = T();

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

for (size\_t j = i + 1; j < A.size(); j++) {

sum += A[i][j] \* A[i][j];

}

}

return std::sqrt(sum);

}

template<class T>

T Norm(vector<T> v) {

T sum = T();

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

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

}

return std::sqrt(sum);

}

template<class T>

T ProdS(vector<T>& v1, vector<T>& v2) {

T result = T();

for (size\_t i = 0; i < std::min(v1.size(), v2.size()); i++) {

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

}

return result;

}

template<class T>

Matrix<T> ProdM(vector<T> v1, vector<T> v2) {

size\_t dim = std::min(v1.size(), v2.size());

Matrix<T> result(dim);

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

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

result[i][j] = v1[i] \* v2[j];

}

}

return result;

}

namespace std {

template<class T>

T sign(const T& obj) {

if (obj < 0) return -1;

if (obj == 0) return 0;

return 1;

}

}

template<class T>

pair<Matrix<T>, Matrix<T>> QR(Matrix<T> A) {

Matrix<T> E(A.size());

for (size\_t i = 0; i < A.size(); i++) E[i][i] = 1;

Matrix<T> Q(A.size());

for (size\_t i = 0; i < A.size(); i++) Q[i][i] = 1;

Matrix<T> H(A.size());

vector<T> v(A.size());

// норма j-ого столбца матрицы

auto l\_Norm = [&](size\_t j) {

T res = T();

for (size\_t i = j; i < A.size(); i++) {

res += A[i][j] \* A[i][j];

}

return std::sqrt(res);

};

for (size\_t k = 0; k < A.size() - 1; k++) {

// 1. Вычисление вектора v

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

if (i < k) {

v[i] = 0;

}

else if (i == k) {

v[i] = A[k][k] + std::sign(A[k][k]) \* l\_Norm(k);

}

else {

v[i] = A[i][k];

}

}

//print(v);

// 2. Построение матрицы H

H = E - ProdM(v, v) \* (2 / ProdS(v, v));

// 3. Обновление A

A = H \* A;

// 4. Сохранение H

Q = Q \* H;

}

return pair<Matrix<T>, Matrix<T>>{Q, A};

}

template<class T>

vector<pair<T, T>> Eigenvalues(Matrix<T>& A, T eps) {

auto error1 = [&] () -> T {

T max = -1;

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

T temp = 0;

for (size\_t i = j + 1; i < A.size(); i++) {

temp += A[i][j] \* A[i][j];

}

max = std::max(max, temp);

}

return std::sqrt(max);

};

auto error2 = [&]() -> T {

T max = -1;

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

for (size\_t i = j + 2; i < A.size(); i++) {

max = std::max(max, abs(A[i][j]));

}

}

return std::sqrt(max);

};

int cnt = 0;

do {

auto qr = QR(A);

A = qr.second \* qr.first;

cnt++;

} while (error1() > eps and error2() > eps);

vector<pair<T, T>> result;

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

if (i+1 < A.size() and abs(A[i+1][i]) > eps) {

T a = A[i][i], b = A[i][i + 1], c = A[i + 1][i], d = A[i + 1][i + 1];

T D = (a + d) \* (a + d) - 4 \* (a \* d - c \* b);

if (D < 0) {

result.push\_back({ (a + d) / 2, std::sqrt(abs(D)) / 2 });

result.push\_back({ (a + d) / 2, -std::sqrt(abs(D)) / 2 });

}

else {

result.push\_back({ (a + d) / 2 + std::sqrt(abs(D)) / 2, 0 });

result.push\_back({ (a + d) / 2 - std::sqrt(abs(D)) / 2, 0 });

}

i += 2;

}

else {

result.push\_back({ A[i][i], 0 });

i++;

}

}

return result;

}

int main()

{

// 13,40254105; 8,77858761; 1,81887134

Matrix<double> m1(3, {

8, -1, -3,

-5, 9, -8,

4, -5, 7

});

Matrix<double> m2(5, {

8, -1, -3, 4, 6,

-5, 9, -8, 5, 0,

4, -5, 7, -3, 4,

4, -7, 2, 9, 4,

0, 0, 2, -1, -1

});

print("Matrix without complex eigenvalues:");

print(std::setprecision(5), m1);

print("After QR - algorithm:");

auto res1 = Eigenvalues(m1, 0.00000001);

print(res1);

print();

print("Matrix with complex eigenvalues:");

print(m2);

print("After QR - algorithm:");

auto res2 = Eigenvalues(m2, 0.00000001);

print(res2);

return 0;

}

Изображение выглядит как текст

Автоматически созданное описание