# Московский авиационный институт
# (национальный исследовательский университет)

**Институт №8 «Информационные технологии и прикладная математика»**

**Кафедра 806 «Вычислительная математика и программирование»**

**Лабораторные работы по курсу «Численные методы»**

| | |
|---|---|
| Студент: | Первухин А.С. |
| Преподаватель: | Пивоваров Д.Е. |
| Группа: | М8О-303Б-21 |
| Дата: | |
| Оценка: | |
| Подпись: | |

**Москва, 2024**

# 1.1 LU - разложение матриц

## 1 Постановка задачи

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.

**Вариант:** 17

$$\begin{cases} 8x_1 + 8x_2 - 5x_3 - 8x_4 = 13 \\ 8x_1 - 5x_2 + 9x_3 - 8x_4 = 38 \\ 5x_1 - 4x_2 - 6x_3 - 2x_4 = 14 \\ 8x_1 + 3x_2 + 6x_3 + 6x_4 = -95 \end{cases}$$

## 2 Результаты работы

```
Определитель = 20050
Обратная матрица:
        0.02            0.01            0.06            0.06
        0.06            -0.03           -0.08           0.02
        -0.01           0.04            -0.07           0.02
        -0.04           -0.05           0.03            0.07
Вектор решений
-4.00
-3.00
-1.00
-8.00
Матрица U
        8.00            8.00            -5.00           -8.00
        0.00            -13.00          14.00           0.00
        0.00            0.00            -12.57          3.00
        0.00            0.00            0.00            15.34
Матрица L
        1.00            0.00            0.00            0.00
        1.00            1.00            0.00            0.00
        0.62            0.69            1.00            0.00
        1.00            0.38            -0.45           1.00
Матрица L*U
        8.00            8.00            -5.00           -8.00
        8.00            -5.00           9.00            -8.00
        5.00            -4.00           -6.00           -2.00
        8.00            3.00            6.00            6.00
```

Рис. 1: Вывод программы в консоли

## 3 Исходный код

```
#include <iostream>
#include <vector>
#include <fstream>
#include <iomanip>

using namespace std;

void LU(vector <vector <double>> A, vector <vector <double>> &L, vector <vector <double>> &U, int n)
{
    U=A;
    for(int i = 0; i < n; i++)
        for(int j = i; j < n; j++)
            L[j][i]=U[j][i]/U[i][i];

    for(int k = 1; k < n; k++)
    {
        for(int i = k-1; i < n; i++)
            for(int j = i; j < n; j++)
                L[j][i]=U[j][i]/U[i][i];

        for(int i = k; i < n; i++)
            for(int j = k-1; j < n; j++)
                U[i][j]=U[i][j]-L[i][k-1]*U[k-1][j];
    }
}

void proisv(vector <vector <double>> A, vector <vector <double>> B,
            vector <vector <double>> &R, int n)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            for(int k = 0; k < n; k++)
                R[i][j] += A[i][k] * B[k][j];
}

void print_matrix(vector <vector <double>> A, int n, ofstream& fout)
{
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            fout <<"\t"<< fixed << setprecision(2) << A[i][j] << "\t";
        }
        fout << endl;
    }
}
```

```
vector<double> first_slau(vector<vector<double>>& L, vector<double>& b, int n) {
    vector<double> z(n, 0);
    for (int i = 0; i < n; i++) {
        z[i] = b[i];
        for (int j = 0; j < i; j++)
            z[i] -= L[i][j] * z[j];
    }
    return z;
}

vector<double> second_slau(vector<vector<double>>& U, vector<double>& z, int n) {
    vector<double> x(n, 0);
    for (int i = n - 1; i >= 0; i--) {
        x[i] = z[i];
        for (int j = i + 1; j < n; j++)
            x[i] -= U[i][j] * x[j];
        x[i] /= U[i][i];
    }
    return x;
}

vector<vector<double>> getMinor(vector<vector<double>>& matrix, int row, int col) {
    vector<vector<double>> minor(matrix.size() - 1, vector<double>(matrix.size() - 1));
    for (int i = 0, r = 0; i < matrix.size(); ++i) {
        if (i == row) continue;
        for (int j = 0, c = 0; j < matrix.size(); ++j) {
            if (j == col) continue;
            minor[r][c++] = matrix[i][j];
        }
        ++r;
    }
    return minor;
}

int determinant(vector<vector<double>>& matrix) {
    int size = matrix.size();
    if (size == 1) return matrix[0][0];
    if (size == 2) return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];

    int det = 0;
    for (int i = 0; i < size; ++i) {
        vector<vector<double>> minor = getMinor(matrix, 0, i);
        int sign = (i % 2 == 0) ? 1 : -1;
        det += sign * matrix[0][i] * determinant(minor);
    }
    return det;
}
```

```cpp

vector<vector<double>> inverseMatrix(vector<vector<double>>& matrix) {
    int n = matrix.size();
    vector<vector<double>> augmentedMatrix(n, vector<double>(2 * n, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            augmentedMatrix[i][j] = matrix[i][j];
        }
        augmentedMatrix[i][i + n] = 1;
    }
    for (int i = 0; i < n; ++i) {
        if (augmentedMatrix[i][i] == 0) {
            for (int j = i + 1; j < n; ++j) {
                if (augmentedMatrix[j][i] != 0) {
                    swap(augmentedMatrix[i], augmentedMatrix[j]);
                    break;
                }
            }
        }
        double divisor = augmentedMatrix[i][i];
        for (int j = i; j < 2 * n; ++j) {
            augmentedMatrix[i][j] /= divisor;
        }
        for (int j = 0; j < n; ++j) {
            if (j != i) {
                double factor = augmentedMatrix[j][i];
                for (int k = i; k < 2 * n; ++k) {
                    augmentedMatrix[j][k] -= factor * augmentedMatrix[i][k];
                }
            }
        }
    }
    vector<vector<double>> inverse(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            inverse[i][j] = augmentedMatrix[i][j + n];
        }
    }
    return inverse;
}

int main()
{
    ofstream fout;
    fout.open("output.txt");
    int n = 4;
    vector <vector <double>> A (n,vector <double>(n, 0)), L (n,vector <double>(n, 0)),
        U(n,vector <double>(n, 0)), R(n,vector <double>(n, 0));
    for(int i = 0; i < n; i++)
```

```
    {
        for(int j = 0; j < n; j++)
        {
            L[i].push_back(0);
            U[i].push_back(0);
            R[i].push_back(0);
        }
    }
    A = {
            {8, 8, -5, -8},
            {8, -5, 9, -8 },
            {5, -4, -6, -2},
            {8, 3, 6, 6}
    };
    vector<double> b = {13, 38, 14, -95};
    LU(A,L,U,n);
    vector <double> z1 = first_slau(L, b, n);
    vector<double> x1 = second_slau(U, z1, n);
    int det = determinant(A);
    fout << " = " << det << endl;
    vector<vector<double>> invMatrix = inverseMatrix(A);
    fout << " :" << endl;
    print_matrix(invMatrix, n, fout);
    fout << " " << endl;
    for (int i = 0; i < n; i++){
        fout << x1[i] << endl;
    }
    fout << " U" << endl;
    print_matrix(U,n, fout);
    fout << " L" << endl;
    print_matrix(L,n, fout);
    proisv(L,U,R,n);
    fout << " L*U" << endl;
    print_matrix(R,n, fout);
    return 0;
}
```

# 1.2 Метод прогонки

## 4 Постановка задачи

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.

**Вариант:** 4

$$\begin{cases} -6x_1 + 5x_2 = 51 \\ -x_1 + 13x_2 + 6x_3 = 100 \\ -9x_2 - 15x_3 - 4x_4 = -12 \\ -x_3 - 7x_4 + x_5 = 47 \\ 9x_4 - 18x_5 = -90 \end{cases}$$

## 5 Результаты работы

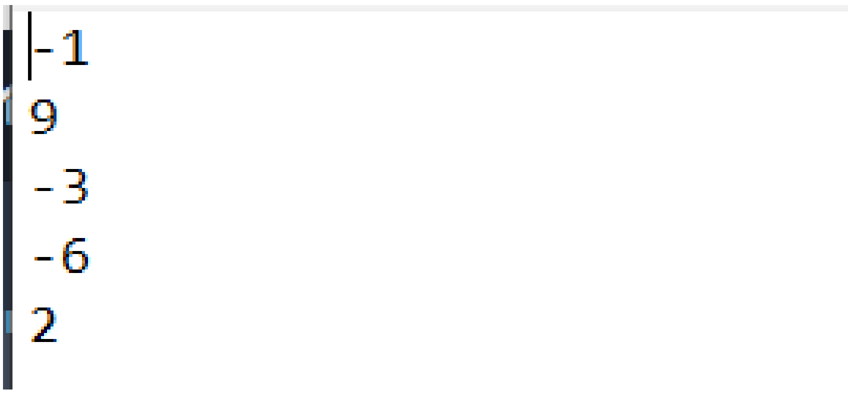



Рис. 2: Вывод программы

# 6 Исходный код

```
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;

int main() {
    int n = 5;
    ofstream fout;
    fout.open("output.txt");
    vector<vector<double>> A = {
            {-6, 5, 0, 0, 0},
            {-1, 13, 6, 0 ,0},
            {0, -9, -15, -4, 0},
            {0, 0, -1, -7, 1},
            {0, 0, 0, 9, -18}
    };
    vector<double> d = {51, 100, -12, 47, -90};
    vector<double> P (n, 0);
    vector<double> Q (n, 0);
    vector<double> x (n, 0);
    P[0] = - A[0][1] / A[0][0];
    Q[0] = d[0] / A[0][0];
    for (int i = 1; i < n; i++){
        P[i] = - A[i][i+1] / (A[i][i] + A[i][i-1] * P[i-1]);
        Q[i] = (d[i] - A[i][i-1] * Q[i-1]) / (A[i][i] + A[i][i-1] * P[i-1]);
    }
    for (int i = n - 1; i >= 0; i--){
        x[i] = P[i] * x[i+1] + Q[i];
    }
    for (int i = 0; i < n; i++){
        fout << x[i] << endl;
    }
    return 0;
}
```

# 1.3 Метод простых итераций. Метод Зейделя

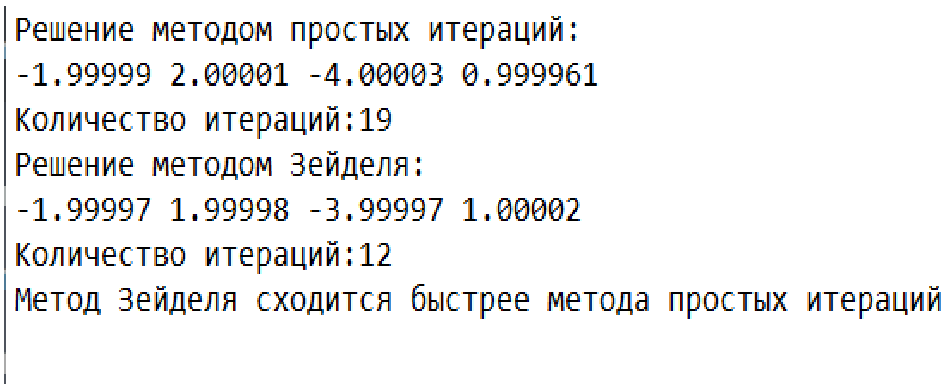
## 7 Постановка задачи

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.

**Вариант:** 17

$$\begin{cases} -19x_1 + 2x_2 - x_3 - 8x_4 = 38 \\ 2x_1 + 14x_2 - 4x_4 = 20 \\ 6x_1 - 5x_2 - 20x_3 - 6x_4 = 52 \\ -6x_1 + 4x_2 - 2x_3 + 15x_4 = 76 \end{cases}$$

## 8 Результаты работы



Решение методом простых итераций:
-1.99999 2.00001 -4.00003 0.999961
Количество итераций:19
Решение методом Зейделя:
-1.99997 1.99998 -3.99997 1.00002
Количество итераций:12
Метод Зейделя сходится быстрее метода простых итераций

Рис. 3: Вывод программы

## 9 Исходный код

```
#include <iostream>
#include <vector>
#include <fstream>
#include <cmath>

using namespace std;

double Norm(vector<double>& A, int n){
```

```
    double sum = 0;
    for (int i = 0; i < n; i++){
        sum += A[i] * A[i];
    }
    double norm = sqrt(sum);
    return norm;
}

int SimpleIter(vector<vector<double>>& A, vector<double>& b, int n, double eps, ofstream& fout){
    vector<double> x_old (n, 0);
    vector<double> x (n, 0);
    vector<double> subtract (n, 0);
    double e = 1;
    for (int i = 0; i < n; i++){
        x_old[i] = b[i] / A[i][i];
    }
    int k = 0;
    while (e > eps){
        for (int i = 0; i < n; i++){
            subtract[i] = 0;
        }
        for (int i = 0; i < n; i++){
            x[i] = b[i] / A[i][i];
            for (int j = 0; j < n; j++){
                if (i == j)
                    continue;
                else
                    x[i] -= A[i][j] / A[i][i] * x_old[j];
            }
        }
        for (int i = 0; i < n; i++){
            subtract[i] = fabs(x[i] - x_old[i]);
        }
        for (int i = 0; i < n; i++){
            x_old[i] = x[i];
        }
        e = Norm(subtract, n);
        k++;
    }
    fout << "    :" << endl;
    for (int i = 0; i < n; i++)
        fout << x_old[i] << " ";
    fout <<endl << " :"<< k << endl;
    return k;
}

int Zeidel(vector<vector<double>>& A, vector<double>& b, int n, double eps, ofstream& fout){
```

```
    vector<double> x (n, 0);
    vector<double> subtract (n, 0);
    vector<double> x_new (n, 0);
    double s1 = 0, s2 = 0;
    double e = 1;
    int k = 0;
    while (e >= eps){
        for (int i = 0; i < n; i++)
            x[i] = x_new[i];
        for (int i = 0; i < n; i++){
            s1 = 0;
            s2 = 0;
            for (int j = 0; j < i; j++){
                s1 += A[i][j] * x_new[j];
            }
            for (int j = i + 1; j < n; j++){
                s2 += A[i][j] * x[j];
            }
            x_new[i] = (b[i] - s1 - s2) / A[i][i];
        }
        for (int i = 0; i < n; i++){
            subtract[i] = fabs(x[i] - x_new[i]);
        }
        e = Norm(subtract, n);
        k++;
    }
    fout << "  :" << endl;
    for (int i = 0; i < n; i++)
        fout << x[i] << " ";
    fout <<endl << " :"<< k << endl;
    return k;
}

int main() {
    int n = 4;
    double eps = 0.0001;
    ofstream fout;
    fout.open("output.txt");
    vector<vector<double>> A = {
            {-19, 2, -1, -8},
            {2, 14, 0, -4},
            {6, -5, -20, -6},
            {-6, 4, -2, 15},
    };
    vector<double> b = {38, 20, 52, 43};
    int k1 = SimpleIter(A, b, n, eps, fout);
    int k2 = Zeidel(A, b, n, eps, fout);
    if (k1 < k2)
        fout << "       "<< endl;
```

```
    else if (k1 == k2)
        fout << "  " << endl;
    else
        fout << "      " << endl;
    return 0;
}
```

# 1.4 Метод вращений

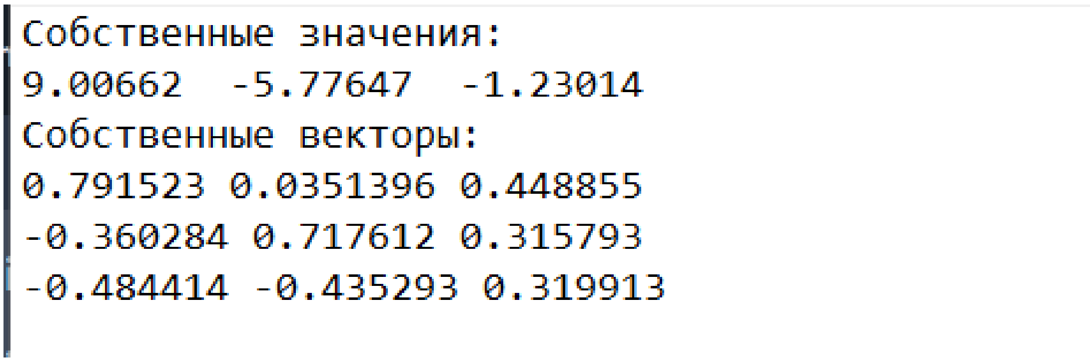
## 10 Постановка задачи

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.

**Вариант:** 17

$$\begin{pmatrix} 5 & -3 & -4 \\ 3 & -3 & 4 \\ -4 & 4 & 0 \end{pmatrix}$$

## 11 Результаты работы



```
Собственные значения:
9.00662  -5.77647  -1.23014
Собственные векторы:
0.791523 0.0351396 0.448855
-0.360284 0.717612 0.315793
-0.484414 -0.435293 0.319913
```

Рис. 4: Вывод программы

## 12 Исходный код

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>

using namespace std;

vector<vector<double>> SingleMatrix(int n) {
    vector<vector<double>> identity(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        identity[i][i] = 1.0;
```

```
    return identity;
}

pair<vector<double>, vector<vector<double>>> Jacobi(vector<vector<double>>& A, double eps) {
    int n = A.size();
    vector<vector<double>> sv = SingleMatrix(n);
    for (int iter = 0; iter < 10000; ++iter) {
        double max_off_diag = 0;
        int p = 0, q = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (abs(A[i][j]) > max_off_diag) {
                    max_off_diag = abs(A[i][j]);
                    p = i;
                    q = j;
                }
            }
        }

        if (max_off_diag < eps)
            break;

        double phi;
        if (A[p][p] == A[q][q])
            phi = M_PI / 4;
        else
            phi = 0.5 * atan(2 * A[p][q] / (A[p][p] - A[q][q]));

        double c = cos(phi);
        double s = sin(phi);

        vector<vector<double>> U = SingleMatrix(n);
        U[p][p] = c;
        U[p][q] = -s;
        U[q][p] = s;
        U[q][q] = c;

        vector<vector<double>> U_T(n, vector<double>(n));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                U_T[i][j] = U[j][i];
            }
        }

        vector<vector<double>> A_temp(n, vector<double>(n));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                double sum = 0;
```

```
                for (int k = 0; k < n; ++k) {
                    sum += U_T[i][k] * A[k][j];
                }
                A_temp[i][j] = sum;
            }
        }

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                double sum = 0;
                for (int k = 0; k < n; ++k) {
                    sum += A_temp[i][k] * U[k][j];
                }
                A[i][j] = sum;
            }
        }

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                double sum = 0;
                for (int k = 0; k < n; ++k) {
                    sum += sv[i][k] * U[k][j];
                }
                sv[i][j] = sum;
            }
        }
    }

    vector<double> sz(n);
    for (int i = 0; i < n; ++i)
        sz[i] = A[i][i];

    return make_pair(sz, sv);
}

int main() {
    ofstream fout;
    fout.open("output.txt");
    double eps = 0.0001;
    vector<vector<double>> A = {{5, -3, -4},
                                {-3, -3, 4},
                                {-4, 4, 0}};
    auto result = Jacobi(A, eps);
    vector<double> sob_val = result.first;
    vector<vector<double>> sob_vec = result.second;

    fout << " :" << endl;
    for (double i : sob_val)
        fout << i << " ";
```

```
    fout << endl;

    fout << " :" << endl;
    for (int i = 0; i < sob_vec.size(); ++i) {
        for (int j = 0; j < sob_vec[i].size(); ++j) {
            fout << sob_vec[i][j] << " ";
        }
        fout << endl;
    }

    return 0;
}
```

# 1.5 QR – разложение матриц

## 13 Постановка задачи

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.

**Вариант:** 17

$$\begin{pmatrix} -6 & 1 & -4 \\ -6 & 8 & -2 \\ 2 & -9 & 5 \end{pmatrix}$$

## 14 Результаты работы

```
Матрица Q:
-0.69 -0.53 -0.49
-0.69 0.26 0.68
0.23 -0.81 0.55
Матрица R:
8.72 -8.26 5.28
0.00 8.82 -2.43
0.00 0.00 3.36
Собственные значения:
8.32 -6.27 4.95
```

Рис. 5: Вывод программы

## 15 Исходный код

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <iomanip>

```

```
using namespace std;

pair<vector<vector<double>>, vector<vector<double>>> QR_decompose(vector<vector<double>>& matrix) {
    int m = matrix.size();
    int n = matrix[0].size();
    vector<vector<double>> q(m, vector<double>(n, 0));
    vector<vector<double>> r(n, vector<double>(n, 0));

    for (int i = 0; i < n; ++i) {
        vector<double> v;
        for (int k = 0; k < m; ++k) {
            v.push_back(matrix[k][i]);
        }

        for (int j = 0; j < i; ++j) {
            double sum = 0.0;
            for (int k = 0; k < m; ++k) {
                sum += q[k][j] * matrix[k][i];
            }
            r[j][i] = sum;
            for (int k = 0; k < m; ++k) {
                v[k] -= r[j][i] * q[k][j];
            }
        }

        double norm_v = 0.0;
        for (double val : v) {
            norm_v += val * val;
        }
        r[i][i] = sqrt(norm_v);
        for (int k = 0; k < m; ++k) {
            q[k][i] = v[k] / r[i][i];
        }
    }
    return make_pair(q, r);
}

vector<vector<double>> Multiplication(vector<vector<double>>& A, vector<vector<double>>& B) {
    int m = A.size();
    int n = B[0].size();
    int p = B.size();
    vector<vector<double>> result(m, vector<double>(n, 0));

    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < p; ++k) {
                result[i][j] += A[i][k] * B[k][j];
```

```
            }
        }
    }
    return result;
}

vector<double> QR_algorithm(vector<vector<double>>& matrix, double eps, ofstream& fout
    ) {
    vector<vector<double>> current_matrix = matrix;
    int n = matrix.size();
    for (int iter = 0; iter < 1000; ++iter) {
        auto [q, r] = QR_decompose(current_matrix);
        current_matrix = Multiplication(r, q);

        double upper_triangular_norm = 0.0;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                upper_triangular_norm += pow(current_matrix[i][j], 2);
            }
        }
        if (sqrt(upper_triangular_norm) < eps) {
            break;
        }
    }
    auto[q1,r1] = QR_decompose(matrix);
    fout << " Q:"<< endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            fout << fixed << setprecision(2)<<q1[i][j] << " ";
        fout << endl;
    }
    fout << " R:"<< endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            fout << fixed << setprecision(2)<< r1[i][j] << " ";
        fout << endl;
    }

    vector<double> sv(n);
    for (int i = 0; i < n; ++i) {
        sv[i] = current_matrix[i][i];
    }
    return sv;
}


int main() {
    ofstream fout;
    fout.open("output.txt");
```

```
    vector<vector<double>> A = {{-6, 1, -4},
                                {-6, 8, -2},
                                {2, -9, 5}};
    double eps = 0.0001;
    vector<double> sob_val = QR_algorithm(A, eps,fout);
    fout << " :\n";
    for (double value : sob_val) {
        fout << value << " ";
    }

    return 0;
}
```