

**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

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <iomanip>
5
6  using namespace std;
7
8  void LU(vector <vector <double>> A, vector <vector <double>> &L, vector <vector <
    double>> &U, int n)
9  {
10     U=A;
11     for(int i = 0; i < n; i++)
12         for(int j = i; j < n; j++)
13             L[j][i]=U[j][i]/U[i][i];
14
15     for(int k = 1; k < n; k++)
16     {
17         for(int i = k-1; i < n; i++)
18             for(int j = i; j < n; j++)
19                 L[j][i]=U[j][i]/U[i][i];
20
21         for(int i = k; i < n; i++)
22             for(int j = k-1; j < n; j++)
23                 U[i][j]=U[i][j]-L[i][k-1]*U[k-1][j];
24     }
25 }
26
27 void proisv(vector <vector <double>> A, vector <vector <double>> B,
28             vector <vector <double>> &R, int n)
29 {
30     for(int i = 0; i < n; i++)
31         for(int j = 0; j < n; j++)
32             for(int k = 0; k < n; k++)
33                 R[i][j] += A[i][k] * B[k][j];
34 }
35
36 void print_matrix(vector <vector <double>> A, int n, ofstream& fout)
37 {
38     for(int i = 0; i < n; i++)
39     {
40         for(int j = 0; j < n; j++)
41         {
42             fout <<"\t"<< fixed << setprecision(2) << A[i][j] << "\t";
43         }
44         fout << endl;
45     }
46 }
```

```

47 |
48 | vector<double> first_slau(vector<vector<double>>& L, vector<double>& b, int n) {
49 |     vector<double> z(n, 0);
50 |     for (int i = 0; i < n; i++) {
51 |         z[i] = b[i];
52 |         for (int j = 0; j < i; j++)
53 |             z[i] -= L[i][j] * z[j];
54 |     }
55 |     return z;
56 | }
57 |
58 | vector<double> second_slau(vector<vector<double>>& U, vector<double>& z, int n) {
59 |     vector<double> x(n, 0);
60 |     for (int i = n - 1; i >= 0; i--) {
61 |         x[i] = z[i];
62 |         for (int j = i + 1; j < n; j++)
63 |             x[i] -= U[i][j] * x[j];
64 |         x[i] /= U[i][i];
65 |     }
66 |     return x;
67 | }
68 |
69 | vector<vector<double>> getMinor(vector<vector<double>>& matrix, int row, int col) {
70 |     vector<vector<double>> minor(matrix.size() - 1, vector<double>(matrix.size() - 1));
71 |     for (int i = 0, r = 0; i < matrix.size(); ++i) {
72 |         if (i == row) continue;
73 |         for (int j = 0, c = 0; j < matrix.size(); ++j) {
74 |             if (j == col) continue;
75 |             minor[r][c++] = matrix[i][j];
76 |         }
77 |         ++r;
78 |     }
79 |     return minor;
80 | }
81 |
82 | int determinant(vector<vector<double>>& matrix) {
83 |     int size = matrix.size();
84 |     if (size == 1) return matrix[0][0];
85 |     if (size == 2) return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
86 |
87 |     int det = 0;
88 |     for (int i = 0; i < size; ++i) {
89 |         vector<vector<double>> minor = getMinor(matrix, 0, i);
90 |         int sign = (i % 2 == 0) ? 1 : -1;
91 |         det += sign * matrix[0][i] * determinant(minor);
92 |     }
93 |     return det;
94 | }
95 |

```

```

96
97 vector<vector<double>> inverseMatrix(vector<vector<double>>& matrix) {
98     int n = matrix.size();
99     vector<vector<double>> augmentedMatrix(n, vector<double>(2 * n, 0));
100     for (int i = 0; i < n; ++i) {
101         for (int j = 0; j < n; ++j) {
102             augmentedMatrix[i][j] = matrix[i][j];
103         }
104         augmentedMatrix[i][i + n] = 1;
105     }
106     for (int i = 0; i < n; ++i) {
107         if (augmentedMatrix[i][i] == 0) {
108             for (int j = i + 1; j < n; ++j) {
109                 if (augmentedMatrix[j][i] != 0) {
110                     swap(augmentedMatrix[i], augmentedMatrix[j]);
111                     break;
112                 }
113             }
114         }
115         double divisor = augmentedMatrix[i][i];
116         for (int j = i; j < 2 * n; ++j) {
117             augmentedMatrix[i][j] /= divisor;
118         }
119         for (int j = 0; j < n; ++j) {
120             if (j != i) {
121                 double factor = augmentedMatrix[j][i];
122                 for (int k = i; k < 2 * n; ++k) {
123                     augmentedMatrix[j][k] -= factor * augmentedMatrix[i][k];
124                 }
125             }
126         }
127     }
128     vector<vector<double>> inverse(n, vector<double>(n, 0));
129     for (int i = 0; i < n; ++i) {
130         for (int j = 0; j < n; ++j) {
131             inverse[i][j] = augmentedMatrix[i][j + n];
132         }
133     }
134     return inverse;
135 }
136
137 int main()
138 {
139     ofstream fout;
140     fout.open("output.txt");
141     int n = 4;
142     vector <vector <double>> A (n,vector <double>(n, 0)), L (n,vector <double>(n, 0)),
143         U(n,vector <double>(n, 0)), R(n,vector <double>(n, 0));
144     for(int i = 0; i < n; i++)

```

```

144     {
145         for(int j = 0; j < n; j++)
146         {
147             L[i].push_back(0);
148             U[i].push_back(0);
149             R[i].push_back(0);
150         }
151     }
152     A = {
153         {8, 8, -5, -8},
154         {8, -5, 9, -8 },
155         {5, -4, -6, -2},
156         {8, 3, 6, 6}
157     };
158     vector<double> b = {13, 38, 14, -95};
159     LU(A,L,U,n);
160     vector<double> z1 = first_slau(L, b, n);
161     vector<double> x1 = second_slau(U, z1, n);
162     int det = determinant(A);
163     fout << " = " << det << endl;
164     vector<vector<double>> invMatrix = inverseMatrix(A);
165     fout << " : " << endl;
166     print_matrix(invMatrix, n, fout);
167     fout << " " << endl;
168     for (int i = 0; i < n; i++){
169         fout << x1[i] << endl;
170     }
171     fout << " U" << endl;
172     print_matrix(U,n, fout);
173     fout << " L" << endl;
174     print_matrix(L,n, fout);
175     proisv(L,U,R,n);
176     fout << " L*U" << endl;
177     print_matrix(R,n, fout);
178     return 0;
179 }

```

## 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

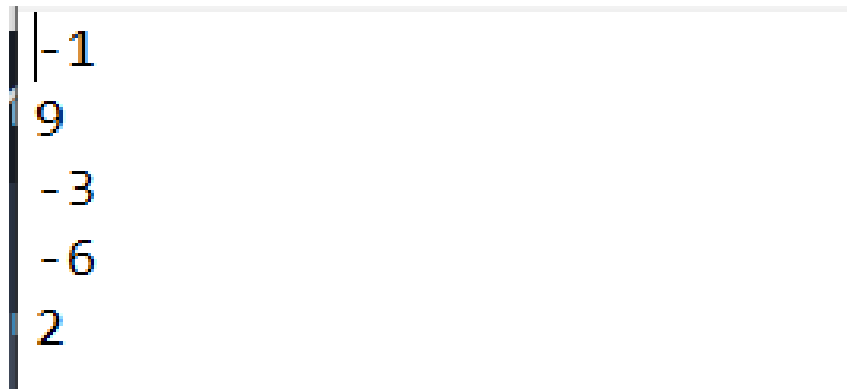


Рис. 2: Вывод программы

## 6 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <fstream>
4 |
5 | using namespace std;
6 |
7 | int main() {
8 |     int n = 5;
9 |     ofstream fout;
10 |    fout.open("output.txt");
11 |    vector<vector<double>> A = {
12 |        {-6, 5, 0, 0, 0},
13 |        {-1, 13, 6, 0, 0},
14 |        {0, -9, -15, -4, 0},
15 |        {0, 0, -1, -7, 1},
16 |        {0, 0, 0, 9, -18}
17 |    };
18 |    vector<double> d = {51, 100, -12, 47, -90};
19 |    vector<double> P (n, 0);
20 |    vector<double> Q (n, 0);
21 |    vector<double> x (n, 0);
22 |    P[0] = - A[0][1] / A[0][0];
23 |    Q[0] = d[0] / A[0][0];
24 |    for (int i = 1; i < n; i++){
25 |        P[i] = - A[i][i+1] / (A[i][i] + A[i][i-1] * P[i-1]);
26 |        Q[i] = (d[i] - A[i][i-1] * Q[i-1]) / (A[i][i] + A[i][i-1] * P[i-1]);
27 |    }
28 |    for (int i = n - 1; i >= 0; i--){
29 |        x[i] = P[i] * x[i+1] + Q[i];
30 |    }
31 |    for (int i = 0; i < n; i++){
32 |        fout << x[i] << endl;
33 |    }
34 |    return 0;
35 | }
```



## 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

```
1 | #include <iostream>  
2 | #include <vector>  
3 | #include <fstream>  
4 | #include <cmath>  
5 |  
6 | using namespace std;  
7 |  
8 | double Norm(vector<double>& A, int n){
```

```

9      double sum = 0;
10     for (int i = 0; i < n; i++){
11         sum += A[i] * A[i];
12     }
13     double norm = sqrt(sum);
14     return norm;
15 }
16
17 int SimpleIter(vector<vector<double>>& A, vector<double>& b, int n, double eps,
18               ofstream& fout){
19     vector<double> x_old (n, 0);
20     vector<double> x (n, 0);
21     vector<double> subtract (n, 0);
22     double e = 1;
23     for (int i = 0; i < n; i++){
24         x_old[i] = b[i] / A[i][i];
25     }
26     int k = 0;
27     while (e > eps){
28         for (int i = 0; i < n; i++){
29             subtract[i] = 0;
30         }
31         for (int i = 0; i < n; i++){
32             x[i] = b[i] / A[i][i];
33             for (int j = 0; j < n; j++){
34                 if (i == j)
35                     continue;
36                 x[i] -= A[i][j] / A[j][j] * x_old[j];
37             }
38         }
39         for (int i = 0; i < n; i++){
40             subtract[i] = fabs(x[i] - x_old[i]);
41         }
42         for (int i = 0; i < n; i++){
43             x_old[i] = x[i];
44         }
45         e = Norm(subtract, n);
46         k++;
47     }
48     fout << "      : " << endl;
49     for (int i = 0; i < n; i++)
50         fout << x_old[i] << " ";
51     fout << endl << " : " << k << endl;
52     return k;
53 }
54
55 int Zeidel(vector<vector<double>>& A, vector<double>& b, int n, double eps, ofstream&
56            fout){

```

```

56     vector<double> x (n, 0);
57     vector<double> subtract (n, 0);
58     vector<double> x_new (n, 0);
59     double s1 = 0, s2 = 0;
60     double e = 1;
61     int k = 0;
62     while (e >= eps){
63         for (int i = 0; i < n; i++)
64             x[i] = x_new[i];
65         for (int i = 0; i < n; i++){
66             s1 = 0;
67             s2 = 0;
68             for (int j = 0; j < i; j++){
69                 s1 += A[i][j] * x_new[j];
70             }
71             for (int j = i + 1; j < n; j++){
72                 s2 += A[i][j] * x[j];
73             }
74             x_new[i] = (b[i] - s1 - s2) / A[i][i];
75         }
76         for (int i = 0; i < n; i++){
77             subtract[i] = fabs(x[i] - x_new[i]);
78         }
79         e = Norm(subtract, n);
80         k++;
81     }
82     fout << "   :" << endl;
83     for (int i = 0; i < n; i++)
84         fout << x[i] << " ";
85     fout << endl << "   :" << k << endl;
86     return k;
87 }
88
89 int main() {
90     int n = 4;
91     double eps = 0.0001;
92     ofstream fout;
93     fout.open("output.txt");
94     vector<vector<double>> A = {
95         {-19, 2, -1, -8},
96         {2, 14, 0, -4},
97         {6, -5, -20, -6},
98         {-6, 4, -2, 15},
99     };
100     vector<double> b = {38, 20, 52, 43};
101     int k1 = SimpleIter(A, b, n, eps, fout);
102     int k2 = Zeidel(A, b, n, eps, fout);
103     if (k1 < k2)
104         fout << "           " << endl;

```

```

105 |     else if (k1 == k2)
106 |         fout << " " << endl;
107 |     else
108 |         fout << " " << endl;
109 |     return 0;
110 | }

```

## 1.4 Метод вращений

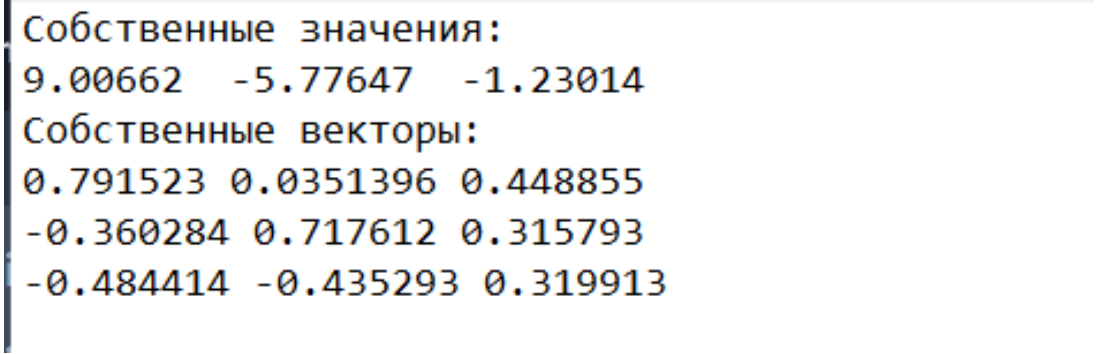
### 10 Постановка задачи

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

```
1 | #include <iostream>  
2 | #include <cmath>  
3 | #include <vector>  
4 | #include <fstream>  
5 |  
6 | using namespace std;  
7 |  
8 | vector<vector<double>> SingleMatrix(int n) {  
9 |     vector<vector<double>> identity(n, vector<double>(n, 0));  
10 |     for (int i = 0; i < n; ++i)  
11 |         identity[i][i] = 1.0;
```

```

12     return identity;
13 }
14
15 pair<vector<double>, vector<vector<double>>> Jacobi(vector<vector<double>>& A, double
    eps) {
16     int n = A.size();
17     vector<vector<double>> sv = SingleMatrix(n);
18     for (int iter = 0; iter < 10000; ++iter) {
19         double max_off_diag = 0;
20         int p = 0, q = 0;
21         for (int i = 0; i < n; ++i) {
22             for (int j = i + 1; j < n; ++j) {
23                 if (abs(A[i][j]) > max_off_diag) {
24                     max_off_diag = abs(A[i][j]);
25                     p = i;
26                     q = j;
27                 }
28             }
29         }
30
31         if (max_off_diag < eps)
32             break;
33
34         double phi;
35         if (A[p][p] == A[q][q])
36             phi = M_PI / 4;
37         else
38             phi = 0.5 * atan(2 * A[p][q] / (A[p][p] - A[q][q]));
39
40         double c = cos(phi);
41         double s = sin(phi);
42
43         vector<vector<double>> U = SingleMatrix(n);
44         U[p][p] = c;
45         U[p][q] = -s;
46         U[q][p] = s;
47         U[q][q] = c;
48
49         vector<vector<double>> U_T(n, vector<double>(n));
50         for (int i = 0; i < n; ++i) {
51             for (int j = 0; j < n; ++j) {
52                 U_T[i][j] = U[j][i];
53             }
54         }
55
56         vector<vector<double>> A_temp(n, vector<double>(n));
57         for (int i = 0; i < n; ++i) {
58             for (int j = 0; j < n; ++j) {
59                 double sum = 0;

```

```

60         for (int k = 0; k < n; ++k) {
61             sum += U_T[i][k] * A[k][j];
62         }
63         A_temp[i][j] = sum;
64     }
65 }
66
67 for (int i = 0; i < n; ++i) {
68     for (int j = 0; j < n; ++j) {
69         double sum = 0;
70         for (int k = 0; k < n; ++k) {
71             sum += A_temp[i][k] * U[k][j];
72         }
73         A[i][j] = sum;
74     }
75 }
76
77 for (int i = 0; i < n; ++i) {
78     for (int j = 0; j < n; ++j) {
79         double sum = 0;
80         for (int k = 0; k < n; ++k) {
81             sum += sv[i][k] * U[k][j];
82         }
83         sv[i][j] = sum;
84     }
85 }
86 }
87
88 vector<double> sz(n);
89 for (int i = 0; i < n; ++i)
90     sz[i] = A[i][i];
91
92 return make_pair(sz, sv);
93 }
94
95 int main() {
96     ofstream fout;
97     fout.open("output.txt");
98     double eps = 0.0001;
99     vector<vector<double>> A = {{5, -3, -4},
100                                {-3, -3, 4},
101                                {-4, 4, 0}};
102     auto result = Jacobi(A, eps);
103     vector<double> sob_val = result.first;
104     vector<vector<double>> sob_vec = result.second;
105
106     fout << " : " << endl;
107     for (double i : sob_val)
108         fout << i << " ";

```

```

109 |     fout << endl;
110 |
111 |     fout << " ." << endl;
112 |     for (int i = 0; i < sob_vec.size(); ++i) {
113 |         for (int j = 0; j < sob_vec[i].size(); ++j) {
114 |             fout << sob_vec[i][j] << " ";
115 |         }
116 |         fout << endl;
117 |     }
118 |
119 |     return 0;
120 | }

```



## 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

```
1 | #include <iostream>  
2 | #include <vector>  
3 | #include <cmath>  
4 | #include <fstream>  
5 | #include <iomanip>  
6 |
```

```

7 using namespace std;
8
9 pair<vector<vector<double>>, vector<vector<double>>>> QR_decompose(vector<vector<double
    >>& matrix) {
10     int m = matrix.size();
11     int n = matrix[0].size();
12     vector<vector<double>> q(m, vector<double>(n, 0));
13     vector<vector<double>> r(n, vector<double>(n, 0));
14
15     for (int i = 0; i < n; ++i) {
16         vector<double> v;
17         for (int k = 0; k < m; ++k) {
18             v.push_back(matrix[k][i]);
19         }
20
21         for (int j = 0; j < i; ++j) {
22             double sum = 0.0;
23             for (int k = 0; k < m; ++k) {
24                 sum += q[k][j] * matrix[k][i];
25             }
26             r[j][i] = sum;
27             for (int k = 0; k < m; ++k) {
28                 v[k] -= r[j][i] * q[k][j];
29             }
30         }
31
32         double norm_v = 0.0;
33         for (double val : v) {
34             norm_v += val * val;
35         }
36         r[i][i] = sqrt(norm_v);
37         for (int k = 0; k < m; ++k) {
38             q[k][i] = v[k] / r[i][i];
39         }
40     }
41     return make_pair(q, r);
42 }
43
44 vector<vector<double>> Multiplication(vector<vector<double>>& A, vector<vector<double
    >>& B) {
45     int m = A.size();
46     int n = B[0].size();
47     int p = B.size();
48     vector<vector<double>> result(m, vector<double>(n, 0));
49
50     for (int i = 0; i < m; ++i) {
51         for (int j = 0; j < n; ++j) {
52             for (int k = 0; k < p; ++k) {
53                 result[i][j] += A[i][k] * B[k][j];

```

```

54     }
55 }
56 }
57 return result;
58 }
59
60 vector<double> QR_algorithm(vector<vector<double>>& matrix, double eps, ofstream& fout
    ) {
61     vector<vector<double>> current_matrix = matrix;
62     int n = matrix.size();
63     for (int iter = 0; iter < 1000; ++iter) {
64         auto [q, r] = QR_decompose(current_matrix);
65         current_matrix = Multiplication(r, q);
66
67         double upper_triangular_norm = 0.0;
68         for (int i = 0; i < n; ++i) {
69             for (int j = i + 1; j < n; ++j) {
70                 upper_triangular_norm += pow(current_matrix[i][j], 2);
71             }
72         }
73         if (sqrt(upper_triangular_norm) < eps) {
74             break;
75         }
76     }
77     auto[q1,r1] = QR_decompose(matrix);
78     fout << " Q:"<< endl;
79     for (int i = 0; i < n; i++) {
80         for (int j = 0; j < n; j++)
81             fout << fixed << setprecision(2)<<q1[i][j] << " ";
82         fout << endl;
83     }
84     fout << " R:"<< endl;
85     for (int i = 0; i < n; i++) {
86         for (int j = 0; j < n; j++)
87             fout << fixed << setprecision(2)<< r1[i][j] << " ";
88         fout << endl;
89     }
90
91     vector<double> sv(n);
92     for (int i = 0; i < n; ++i) {
93         sv[i] = current_matrix[i][i];
94     }
95     return sv;
96 }
97
98
99 int main() {
100     ofstream fout;
101     fout.open("output.txt");

```

```

102     vector<vector<double>> A = {{-6, 1, -4},
103                                {-6, 8, -2},
104                                {2, -9, 5}};
105     double eps = 0.0001;
106     vector<double> sob_val = QR_algorithm(A, eps,fout);
107     fout << " :\n";
108     for (double value : sob_val) {
109         fout << value << " ";
110     }
111
112     return 0;
113 }

```