

**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: В. В. Хрушкова  
Преподаватель: Д. Е. Пивоваров  
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: 27**

$$\begin{cases} -2x_1 - 1x_2 - 9x_3 - 5x_4 = 93 \\ -4x_1 + 4x_2 - 2x_3 + 6x_4 = 16 \\ 5x_2 + 7x_3 - 4x_4 = -80 \\ 9x_2 + 7x_3 + 7x_4 = -119 \end{cases}$$

### 2 Результаты работы

```
U =
-2 -1 -9 -5
0 6 16 16
0 0 -6.33333 -17.3333
0 0 0 29.5263

L =
1 0 0 0
2 1 0 0
-0 0.833333 1 0
-0 1.5 2.68421 1

Determinant = 2244

Result =
-9
-7
-7
-1

Inverted =
0.125668 -0.312834 -0.181818 0.254011
0.137255 -0.0686275 7.40149e-17 0.156863
-0.12656 0.0632799 0.0909091 -0.0926916
-0.0499109 0.0249554 -0.0909091 0.0338681
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1 | #include <bits/stdc++.h>
2 |
3 | using namespace std;
4 |
5 |
6 | pair<vector<vector<double>>, vector<vector<double>>>> LU(vector<vector<double>>& X,
7 |     vector<vector<double>>& Y, int n) {
8 |     vector<vector<double>> L(n);
9 |     vector<vector<double>> U = X;
10 |
11 |     //
12 |     for (int i=0; i<n; i++)
13 |         for (int j=0; j<n; j++)
14 |             L[i].push_back(0);
15 |
16 |     for (int k=0; k<n; k++) {
17 |         //
18 |         if (U[k][k] == 0) {
19 |             for (int i=k+1; i<n; i++) {
20 |                 if (U[i][k] != 0) {
21 |                     swap(X[k], X[i]);
22 |                     swap(Y[k], Y[i]);
23 |                     swap(U[k], U[i]);
24 |                     swap(L[k], L[i]);
25 |                     break;
26 |                 }
27 |             }
28 |         }
29 |         //
30 |         //
31 |         L[k][k] = 1;
32 |         for (int i=k+1; i<n; i++) {
33 |             L[i][k] = U[i][k]/U[k][k];
34 |             if (U[i][k] == 0)
35 |                 continue;
36 |             for(int j=k; j<n; j++)
37 |                 U[i][j] -= L[i][k]*U[k][j];
38 |
39 |         }
40 |     }
41 |
42 |     return make_pair(L, U);
43 | }
44 |
45 |
```

```

46 vector<vector<double>> calculate_result(vector<vector<double>>& X, vector<vector<
    double>>& Y, int n) {
47     vector<vector<double>> L, U;
48     tie(L, U) = LU(X, Y, n);
49     vector<vector<double>> res = Y;
50
51     int m = res[0].size();
52     //
53     for (int k=0; k<m; k++)
54         for (int i=0; i<n; i++)
55             for (int j=0; j<i; j++)
56                 res[i][k] -= res[j][k]*L[i][j];
57     for (int k=0; k<m; k++) {
58         for (int i=n-1; i>-1; i--) {
59             for (int j=i+1; j<n; j++) {
60                 res[i][k] -= res[j][k]*U[i][j];
61             }
62             res[i][k] /= U[i][i];
63         }
64     }
65
66     return res;
67 }
68
69
70 void write_matrix_to_file(const vector<vector<double>>& matrix, ofstream& out) {
71     //
72     int n = matrix.size(), m = matrix[0].size();
73     for(int i=0; i<n; ++i){
74         for(int j=0; j<m; j++)
75             out << matrix[i][j] << " ";
76         out << endl;
77     }
78 }
79
80
81 int main() {
82     ifstream in("input.txt");
83     int n; //
84     in >> n;
85     vector<vector<double>> X(n), Y(n);
86
87     //
88     for (int i=0; i<n; ++i)
89         for (int j=0; j<n; ++j) {
90             int number;
91             in >> number;
92             X[i].push_back(number);
93         }

```

```

94
95 //
96 for (int i=0; i<n; ++i) {
97     int number;
98     in >> number;
99     Y[i].push_back(number);
100 }
101
102 in.close();
103
104 ofstream out("output.txt");
105 vector<vector<double>> L, U;
106 tie(L, U) = LU(X, Y, n);
107
108 out << "U =" << endl;
109 write_matrix_to_file(U, out);
110
111 out << endl << "L =" << endl;
112 write_matrix_to_file(L, out);
113
114
115 //
116 double determinant = 1;
117 for (int i=0; i<n; i++)
118     determinant *= U[i][i];
119 out << endl << "Determinant = " << determinant << endl;
120
121
122 //
123 vector<vector<double>> result = calculate_result(X, Y, n);
124 out << endl << "Result =" << endl;
125 write_matrix_to_file(result, out);
126
127
128 //
129 vector<vector<double>> E(n);
130 for (int i=0; i<n; i++)
131     for (int j=0; j<n; j++)
132         if (i != j)
133             E[i].push_back(0);
134         else
135             E[i].push_back(1);
136 vector<vector<double>> invert = calculate_result(X, E, n); //
137 out << endl << "Inverted =" << endl;
138 write_matrix_to_file(invert, out);
139
140 out.close();
141 return 0;
142 }

```

## 1.2 Метод прогонки

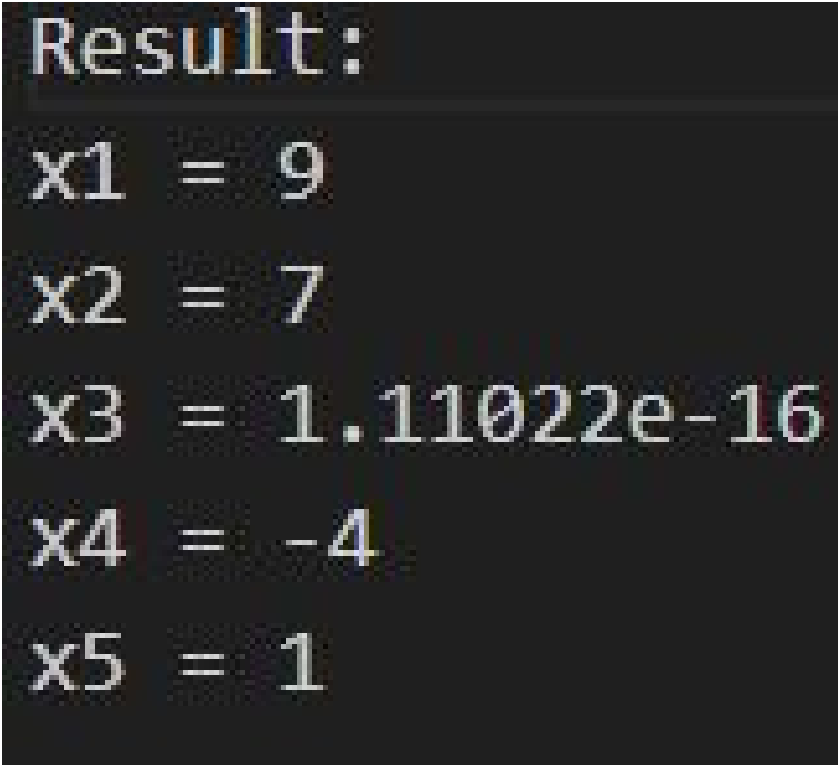
### 4 Постановка задачи

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: 27**

$$\begin{cases} -6x_1 + 3x_2 = -33 \\ 6x_1 - 23x_2 - 9x_3 = -107 \\ 2x_2 + 7x_3 - x_4 = 18 \\ 4x_3 + 15x_4 - 9x_5 = -69 \\ -6x_4 + 14x_5 = -31 \end{cases}$$

### 5 Результаты работы



```
Result:
x1 = 9
x2 = 7
x3 = 1.11022e-16
x4 = -4
x5 = 1
```

Рис. 2: Вывод программы в консоли

## 6 Исходный код

```
1  #include <vector>
2  #include <iostream>
3  #include <fstream>
4  #include <cstdio>
5
6  using namespace std;
7
8  //
9  vector<vector<double>>> tridiagonal(vector<vector<double>>>& coefficients, vector<vector
    <double>>>& results) {
10     int n = coefficients.size();
11     double a = 0, b = coefficients[0][0], c = coefficients[0][1], d = results[0][0];
12     vector<double> P(n, 0), Q(n, 0);
13
14     P[0] = -c/b;
15     Q[0] = d/b;
16     for (int i=1; i < n-1; i++){
17         a = coefficients[i][i-1];
18         b = coefficients[i][i];
19         c = coefficients[i][i+1];
20         d = results[i][0];
21
22         P[i] = -c/(b + a*P[i-1]);
23         Q[i] = (d - a*Q[i-1])/(b + a*P[i-1]);
24     }
25
26     a = coefficients[n-1][n-2];
27     b = coefficients[n-1][n-1];
28     c = 0;
29     d = results[n-1][0];
30
31     Q[n-1] = (d - a * Q[n-2]) / (b + a * P[n-2]);
32
33     vector<vector<double>>> result(n);
34     for(int i=0; i<n; i++)
35         result[i].push_back(0);
36
37     result[n-1][0] = Q[n-1];
38     for (int i = n-2; i > -1; i--)
39         result[i][0] = P[i]*result[i+1][0] + Q[i];
40
41     return result;
42 }
43
44 int main() {
45     ifstream in("input.txt");
46     int n; //
```

```

47 | in >> n;
48 | vector<vector<double>> X(n), Y(n);
49 |
50 | //
51 | for (int i=0; i<n; ++i)
52 |     for (int j=0; j<n; ++j) {
53 |         int number;
54 |         in >> number;
55 |         X[i].push_back(number);
56 |     }
57 |
58 | //
59 | for (int i=0; i<n; ++i) {
60 |     int number;
61 |     in >> number;
62 |     Y[i].push_back(number);
63 | }
64 |
65 | in.close();
66 |
67 | vector<vector<double>> result = tridiagonal(X, Y);
68 | ofstream out("output.txt");
69 |
70 | //
71 | out << "Result:" << endl;
72 | for (int i=0; i<n; i++)
73 |     out << "x" << i+1 << " = " << result[i][0] << endl;
74 |
75 | out.close();
76 | return 0;
77 | }

```



## 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: 27**

$$\begin{cases} -26x_1 - 7x_2 - 8x_3 - 2x_4 = -51 \\ 2x_1 - 17x_2 - 6x_3 - 2x_4 = 85 \\ -7x_1 - 6x_2 - 23x_3 - 3x_4 = 71 \\ 3x_1 - 2x_2 - 7x_3 - 13x_4 = 91 \end{cases}$$

### 8 Результаты работы

```
Iterations result:
3.99966
-3.00021
-3.00035
-4.00026

Seidel result:
4.00008
-3
-3.00004
-3.99996
```

Рис. 3: Вывод программы в консоли

## 9 Исходный код

```
1  #include <vector>
2  #include <iostream>
3  #include <fstream>
4  #include <cstdio>
5  #include <cmath>
6
7  using namespace std;
8
9  vector<vector<double>> plus_matrix(const vector<vector<double>>& matrix1, const vector<
    vector<double>>& matrix2) {
10     int n = matrix1.size(), m = matrix1[0].size();
11     vector<vector<double>> res(n, vector<double>(m));
12     for (int i = 0; i < n; ++i)
13         for (int j = 0; j < m; ++j)
14             res[i][j] = matrix1[i][j] + matrix2[i][j];
15     return res;
16 }
17
18 vector<vector<double>> multiply(const vector<vector<double>>& matrix1, const vector<
    vector<double>>& matrix2) {
19     int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
20     vector<vector<double>> res(n1, vector<double>(m2, 0));
21     for (int i = 0; i < n1; ++i) {
22         for (int j = 0; j < m2; ++j) {
23             double cntr = 0;
24             for (int k = 0; k < m1; ++k) {
25                 cntr += matrix1[i][k] * matrix2[k][j];
26             }
27             res[i][j] = cntr;
28         }
29     }
30     return res;
31 }
32
33 double get_current_eps(const vector<vector<double>>& v1, const vector<vector<double>>&
    v2) {
34     double eps = 0;
35     int n = v1.size();
36     for(int i=0; i<n; i++)
37         eps += pow(v1[i][0] - v2[i][0], 2);
38     return sqrt(eps);
39 }
40
41 vector<vector<double>> simple_iterations(vector<vector<double>>& a, vector<vector<
    double>>& b, double EPS0) {
42     vector<vector<double>> x_previous = b, x_current;
```

```

43     while (get_current_eps(x_current = plus_matrix(b, multiply(a, x_previous)),
44         x_previous) > EPS0)
45         x_previous = x_current;
46     return x_previous;
47 }
48
49 vector<vector<double>> seidel_method(const vector<vector<double>>& a, const vector<
    vector<double>>& b, double EPS0) {
50     vector<vector<double>> x_previous = b, x_current = x_previous;
51     bool flag = true;
52     double eps = 0;
53     int n = b.size();
54     while (flag or eps > EPS0) {
55         flag = false;
56         for(int i = 0; i < n; i++) {
57             x_current[i][0] = 0;
58             for(int j=0; j < n; j++) {
59                 if (i == j)
60                     x_current[i][0] += b[i][0];
61                 else if (i < j)
62                     x_current[i][0] += a[i][j]*x_previous[j][0];
63                 else
64                     x_current[i][0] += a[i][j]*x_current[j][0];
65             }
66         }
67         eps = get_current_eps(x_current, x_previous);
68         x_previous = x_current;
69     }
70     return x_previous;
71 }
72
73
74 int main() {
75     ifstream in("input.txt");
76     int n; //
77     in >> n;
78     vector<vector<double>> X(n), Y(n);
79
80     //
81     for (int i=0; i<n; ++i)
82         for (int j=0; j<n; ++j) {
83             int number;
84             in >> number;
85             X[i].push_back(number);
86         }
87
88     //
89     for (int i=0; i<n; ++i) {

```

```

90     int number;
91     in >> number;
92     Y[i].push_back(number);
93 }
94
95 in.close();
96
97 ofstream out("output.txt");
98
99 for(int i=0; i < n; i++) {
100     double a = X[i][i];
101     if (a == 0)
102         continue;
103     Y[i][0] /= a;
104     for (int j=0; j < n; j++)
105         X[i][j] = (i == j) ? 0 : -X[i][j]/a;
106 }
107
108 out << "Iterations result:\n";
109 for (auto row: simple_iterations(X, Y, 0.001)){
110     for (auto el: row)
111         out << el << " ";
112     out << endl;
113 }
114 out << endl;
115
116 out << "Seidel result:\n";
117 for (auto row: seidel_method(X, Y, 0.001)){
118     for (auto el: row)
119         out << el << " ";
120     out << endl;
121 }
122 out.close();
123 return 0;
124 }

```

## 1.4 Метод вращений

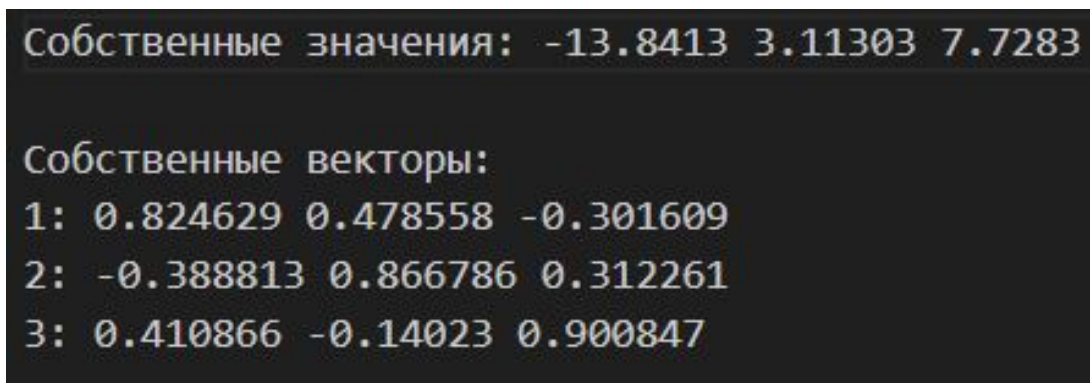
### 10 Постановка задачи

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: 27**

$$\begin{pmatrix} -8 & 5 & -7 \\ 5 & 1 & 4 \\ -7 & 4 & 4 \end{pmatrix}$$

### 11 Результаты работы



```
Собственные значения: -13.8413 3.11303 7.7283

Собственные векторы:
1: 0.824629 0.478558 -0.301609
2: -0.388813 0.866786 0.312261
3: 0.410866 -0.14023 0.900847
```

Рис. 4: Вывод программы в консоли

## 12 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <fstream>
5
6  using namespace std;
7
8  vector<vector<double>> multiple_matrix(const vector<vector<double>>& matrix1, const
    vector<vector<double>>& matrix2) {
9      int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
10     vector<vector<double>> res(n1);
11     for (int i=0; i<n1; ++i)
12         for (int j=0; j<m2; ++j)
13             res[i].push_back(0);
14
15     for (int i=0; i<n1; ++i) {
16         for (int j=0; j<m2; ++j) {
17             double cntr = 0;
18             for (int k=0; k<m1; k++)
19                 cntr += matrix1[i][k] * matrix2[k][j];
20             res[i][j] = cntr;
21         }
22     }
23     return res;
24 }
25
26 vector<vector<double>> transp(const vector<vector<double>>& matrix1){
27     int n = matrix1.size(), m = matrix1[0].size();
28     vector<vector<double>> res(m, vector<double>(n));
29     for (int i=0; i<n; ++i)
30         for (int j=0; j<m; ++j)
31             res[j][i] = matrix1[i][j];
32     return res;
33 }
34
35 vector<vector<double>> E(int n){
36     vector<vector<double>> E(n, vector<double>(n, 0));
37     for(int i=0; i<n; ++i)
38         E[i][i] = 1;
39     return E;
40 }
41
42 vector<vector<double>> U(const vector<vector<double>>& matrix1){
43     vector<vector<double>> u = E(matrix1.size());
44     int i_max = 0, j_max = 1;
45     double k = matrix1[0][1];
46     int n = matrix1.size(), m = matrix1[0].size();
```

```

47     for (int i = 0; i < n; ++i)
48         for (int j = i+1; j < m; ++j)
49             if (abs(matrix1[i][j]) > k) {
50                 k = abs(matrix1[i][j]);
51                 i_max = i;
52                 j_max = j;
53             }
54     double phi = (matrix1[i_max][i_max] == matrix1[j_max][j_max]) ? atan(2*matrix1[
        i_max][j_max]/(matrix1[i_max][i_max] - matrix1[j_max][j_max])) / 2 : 3.14/4;
55
56     u[i_max][j_max] = -sin(phi);
57     u[j_max][i_max] = sin(phi);
58     u[i_max][i_max] = cos(phi);
59     u[j_max][j_max] = cos(phi);
60     return u;
61 }
62
63 double eps(const vector<vector<double>>& matrix1){
64     int n=matrix1.size(), m=matrix1[0].size();
65     double error = 0;
66     for (int i=0; i<n; ++i)
67         for (int j=i+1; j<m; ++j)
68             error += matrix1[i][j]*matrix1[i][j];
69     return sqrt(error);
70 }
71
72 pair<vector<double>, vector<vector<double>>> jacobi(vector<vector<double>>&
    coeff_matrix, double EPS){
73     int n = coeff_matrix.size();
74     vector<vector<double>> eigenvectors = E(n);
75     while (eps(coeff_matrix) > EPS){
76         vector<vector<double>> u = U(coeff_matrix);
77         eigenvectors = multiple_matrix(eigenvectors, u);
78         vector<vector<double>> U_T = transp(u);
79         coeff_matrix = multiple_matrix(multiple_matrix(U_T, coeff_matrix), u);
80     }
81
82     vector<double> eigenvalues(n);
83     for (int i=0; i<n; ++i)
84         eigenvalues[i] = coeff_matrix[i][i];
85     return make_pair(eigenvalues, eigenvectors);
86 }
87
88 int main() {
89     ifstream in("input.txt");
90     int n; //
91     in >> n;
92     vector<vector<double>> X(n), Y(n);
93

```

```

94 //
95 for (int i=0; i<n; ++i)
96     for (int j=0; j<n; ++j) {
97         int number;
98         in >> number;
99         X[i].push_back(number);
100     }
101
102 in.close();
103
104 ofstream out("output.txt");
105
106 vector<double> eigenvalues;
107 vector<vector<double>> eigenvectors;
108 tie(eigenvalues, eigenvectors) = jacobi(X, 0.01);
109
110 out << " :";
111 for (int i=0; i<n; ++i)
112     out << " " << eigenvalues[i];
113 out << endl << endl;
114
115 out << " :";
116 for (int i=0; i<n; ++i){
117     out << endl << i+1 << ": ";
118     for(double element: eigenvectors[i])
119         out << element << " ";
120 }
121
122 out.close();
123 return 0;
124 }

```



## 1.5 QR – разложение матриц

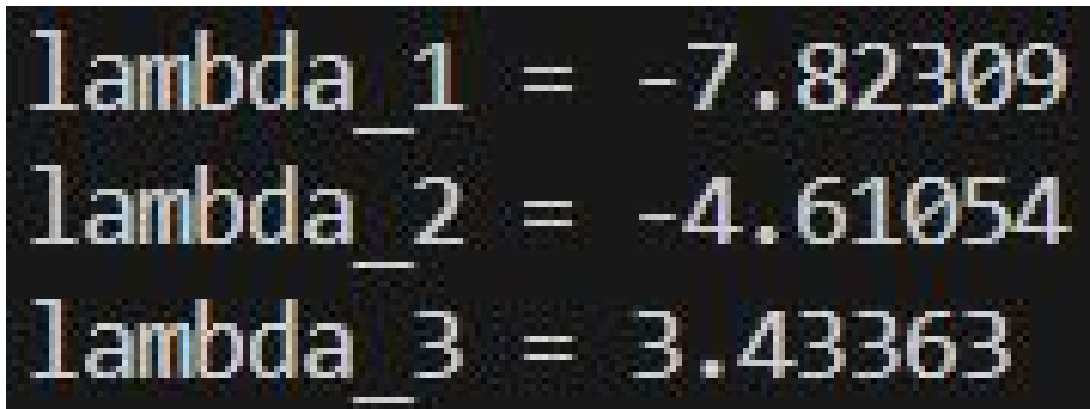
### 13 Постановка задачи

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: 27**

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

### 14 Результаты работы



```
lambda_1 = -7.82309
lambda_2 = -4.61054
lambda_3 = 3.43363
```

Рис. 5: Вывод программы в консоли

## 15 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using matrix = vector<vector<double> >;
5
6
7  void print_matrix(const matrix& matrix1) {
8      for(const auto& vect: matrix1) {
9          for (auto x: vect)
10             cout << x << " ";
11         cout << endl;
12     }
13 }
14
15
16 matrix plus_matrix(const matrix& matrix1, const matrix& matrix2) {
17     matrix res;
18     int n = matrix1.size();
19     for (int i = 0; i < n; i++) {
20         vector<double> row;
21         for (int j = 0; j < n; j++) {
22             row.push_back(matrix1[i][j] + matrix2[i][j]);
23         }
24         res.push_back(row);
25     }
26
27     return res;
28 }
29
30
31 matrix transposed(const matrix& matrix1){
32     int n = matrix1.size(), m = matrix1[0].size();
33     matrix res(m, vector<double>(n));
34     for (int i=0; i<n; i++)
35         for (int j=0; j<m; j++)
36             res[j][i] = matrix1[i][j];
37     return res;
38 }
39
40 matrix multiple_matrix(matrix& matrix1, matrix& matrix2) {
41     int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
42     matrix res(n1);
43     for (int i=0; i<n1; i++)
44         for (int j=0; j<m2; j++)
45             res[i].push_back(0);
46
47     for (int i=0; i<n1; i++) {
```

```

48     for (int j=0; j<m2; j++) {
49         double cntr = 0;
50         for (int k=0; k<m1; k++)
51             cntr += matrix1[i][k] * matrix2[k][j];
52         res[i][j] = cntr;
53     }
54 }
55 return res;
56 }
57
58
59 int sign(double a){
60     return (a >= 0) ? 1 : -1;
61 }
62
63
64 double get_eps(const matrix& matrix1){
65     double eps = 0;
66     int n = matrix1.size();
67     for(int i=0; i<n; i++)
68         for(int j=0; j<i-1; j++)
69             eps += matrix1[i][j]*matrix1[i][j];
70     return sqrt(eps);
71 }
72
73
74 matrix get_E_matrix(int n){
75     matrix E(n, vector<double>(n, 0));
76     for(int i=0; i<n; i++)
77         E[i][i] = 1;
78     return E;
79 }
80
81
82 matrix get_H_matrix(const matrix& coefficients, int ind){
83     int n = coefficients.size();
84     matrix v(n, vector<double>(1));
85
86     for(int i=0; i<n; i++){
87         if (i < ind)
88             v[i][0] = 0;
89         else if (i == ind){
90             double sum = 0;
91             for (int j=ind; j < n; j++)
92                 sum += coefficients[j][i]*coefficients[j][i];
93             v[i][0] = coefficients[i][i] + sign(coefficients[i][i]) * sqrt(sum);
94         }
95         else
96             v[i][0] = coefficients[i][ind];

```

```

97     }
98
99     matrix transposed_v = transposed(v);
100     double k = -multiple_matrix(transposed_v, v)[0][0]/2;
101     v = multiple_matrix(v, transposed_v);
102     for (int i=0; i<n; i++)
103         for (int j=0; j<n; j++)
104             v[i][j] /= k;
105
106     matrix E = get_E_matrix(n);
107     return plus_matrix(E, v);
108 }
109
110
111 pair<matrix, matrix> QR_decomposition(const matrix& coeff){
112     matrix coefficients = coeff;
113     matrix Q = get_H_matrix(coefficients, 0);
114     coefficients = multiple_matrix(Q, coefficients);
115     int n = coefficients.size();
116     for (int i=1; i<n-1; i++){
117         matrix H = get_H_matrix(coefficients, i);
118         Q = multiple_matrix(Q, H);
119         coefficients = multiple_matrix(H, coefficients);
120     }
121     return make_pair(Q, coefficients);
122 }
123
124
125 vector<double> get_eigenvalues(matrix& coefficients, double EPS){
126     while (get_eps(coefficients) > EPS){
127         pair<matrix, matrix> QR = QR_decomposition(coefficients);
128         coefficients = multiple_matrix(QR.second, QR.first);
129     }
130     int n = coefficients.size();
131     vector<double> result(n);
132     for (int i=0; i<n; i++)
133         result[i] = coefficients[i][i];
134     return result;
135 }
136
137
138 int main() {
139     matrix coefficient_matrix{
140         {-5, -8, 4},
141         {4, 2, 6},
142         {-2, 5, -6}
143     };
144
145

```

```

146 |     vector<double> eigenvalues = get_eigenvalues(coefficient_matrix, 0.01);
147 |     int n = eigenvalues.size();
148 |
149 |     for (int i=0; i<n; i++)
150 |         cout << "lambda_" << i+1 << " = " << eigenvalues[i] << endl;
151 |
152 |     return 0;
153 | }

```