

**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: М. Р. Жалялетдинов
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

3 Исходный код

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using double_matrix = vector<vector<double> >;
5
6
7 pair<double_matrix, double_matrix> decompose(double_matrix& coeffs, double_matrix&
  roots) {
8     int shape=coeffs.size(), shape2=coeffs[0].size(), shape3 = roots[0].size();
9     double_matrix L(shape), U = coeffs;
10    for (int i=0; i<shape; i++)
11        for (int j=0; j<shape2; j++)
12            L[i].push_back(0);
13
14    for (int k=0; k<shape; k++) {
15        if (U[k][k] == 0) {
16            for (int i=k+1; i<shape; i++) {
17                if (U[i][k] != 0) {
18                    swap(U[k], U[i]);
19                    swap(L[k], L[i]);
20                    swap(coeffs[k], coeffs[i]);
21                    swap(roots[k], roots[i]);
22                    break;
23                }
24            }
25        }
26        L[k][k] = 1;
27        for (int i=k+1; i<shape; i++) {
28            L[i][k] = U[i][k]/U[k][k];
29            if (U[i][k] == 0)
30                continue;
31            for(int j=k; j<shape2; j++)
32                U[i][j] -= L[i][k]*U[k][j];
33        }
34    }
35    }
36
37    return {L, U};
38 }
39
40
41 double operdelitel(const double_matrix& U) {
42     double det = 1;
43     for (int i=0; i<U.size(); i++)
44         det *= U[i][i];
45     return det;
46 }
```

```

47
48
49 double_matrix calculate_decisions(double_matrix& coeffs, double_matrix& roots) {
50     auto [L, U] = decompose(coeffs, roots);
51     double_matrix res = roots;
52
53     for (int k=0; k<res[0].size(); k++)
54         for (int i=0; i<res.size(); 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<res[0].size(); k++) {
58         for (int i=coeffs.size()-1; i>-1; i--) {
59             for (int j=i+1; j<roots.size(); 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 double_matrix get_inverse_matrix(double_matrix& matrix1) {
71     double_matrix E(matrix1.size());
72     for (int i=0; i<matrix1.size(); i++)
73         for (int j=0; j<matrix1.size(); j++)
74             E[i].push_back((i == j) ? 1 : 0);
75     return calculate_decisions(matrix1, E);
76 }
77
78 void print_matrix(const double_matrix& matrix1) {
79     for(const auto& vect: matrix1) {
80         for (auto val: vect)
81             cout << val << " ";
82         cout << endl;
83     }
84 }
85
86 int main() {
87     double_matrix coefficient_matrix{{-4, -9, 4, 3}, {2, 7, 9, 8}, {4, -4, 0, -2}, {-8,
88         5, 2, 9}};
89     double_matrix equation_roots = {{-51}, {76}, {26}, {-73}};
89
90     auto [l, u] = decompose(coefficient_matrix, equation_roots);
91
92     cout << endl << "L : " << endl;
93     print_matrix(l);
94

```

```

95     cout << endl << endl;
96     cout << "U : " << endl;
97     print_matrix(u);
98
99     cout << endl << endl;
100    cout << " : " << operdelitel(u) << endl;
101
102    cout << endl << endl << " : " << endl;
103    double_matrix inversed = get_inverse_matrix(coefficient_matrix);
104    print_matrix(inversed);
105
106    cout << endl << endl << " : " << endl;
107    double_matrix decisions = calculate_decisions(coefficient_matrix, equation_roots);
108    print_matrix(decisions);
109
110    return 0;
111 }

```

1.2 Метод прогонки

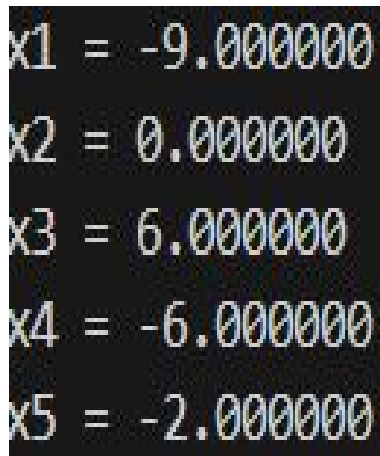
4 Постановка задачи

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: 8

$$\begin{cases} -11x_1 - 8x_2 = 99 \\ 9x_1 - 17x_2 + 1x_3 = -75 \\ -4x_2 + 20x_3 + 9x_4 = 66 \\ -4x_3 - 14x_4 + 3x_5 = 54 \\ -6x_4 + 14x_5 = 8 \end{cases}$$

5 Результаты работы



```
x1 = -9.000000
x2 = 0.000000
x3 = 6.000000
x4 = -6.000000
x5 = -2.000000
```

Рис. 2: Вывод программы в консоли

6 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using double_matrix = vector<vector<double>>>;
5
6
7  double_matrix tridiag(double_matrix& coeffs, double_matrix& res_matr) {
8      double a, b, c, d;
9      a = 0;
10     b = coeffs[0][0];
11     c = coeffs[0][1];
12     d = res_matr[0][0];
13     vector<double> P(coeffs[0].size(), 0), Q(coeffs[0].size(), 0);
14
15     P[0] = -c/b;
16     Q[0] = d/b;
17     for (int i=1; i < coeffs.size() - 1; i++){
18         a = coeffs[i][i-1];
19         b = coeffs[i][i];
20         c = coeffs[i][i+1];
21         d = res_matr[i][0];
22
23         P[i] = -c/(b + a*P[i-1]);
24         Q[i] = (d - a*Q[i-1])/(b + a*P[i-1]);
25     }
26
27     a = coeffs[coeffs.size()-1][coeffs[0].size()-2];
28     b = coeffs[coeffs.size()-1][coeffs[0].size()-1];
29     c = 0;
30     d = res_matr[res_matr.size()-1][0];
31
32     Q[Q.size()-1] = (d - a * Q[Q.size()-2]) / (b + a * P[P.size()-2]);
33
34     double_matrix decision(res_matr.size());
35     for(int i=0; i<decision.size(); i++)
36         decision[i].push_back(0);
37
38     decision[decision.size()-1][0] = Q[Q.size()-1];
39     for (int i = decision.size()-2; i > -1; i--)
40         decision[i][0] = P[i]*decision[i+1][0] + Q[i];
41
42     return decision;
43 }
44
45
46 void cout_matrix(double_matrix& matrix1) {
47     for(const auto& vect: matrix1) {
```

```

48     for (auto x: vect)
49         cout << x << " ";
50     cout << endl;
51 }
52 }
53
54
55 int main() {
56     double_matrix coeff_matrix{{-11, -8, 0, 0, 0}, {9, -17, 1, 0, 0}, {0, -4, 20, 9,
57         0}, {0, 0, -4, -14, 3}, {0, 0, 0, -6, 14}};
58     double_matrix decisions = {{99}, {-75}, {66}, {54}, {8}};
59     double_matrix res_matrix = tridiag(coeff_matrix, decisions);
60     cout << " : " << endl;
61     for (int i=0; i<res_matrix.size(); i++)
62         printf("x%d = %f \n", i+1, res_matrix[i][0]);
63
64     return 0;
65 }

```


1.3 Метод простых итераций. Метод Зейделя

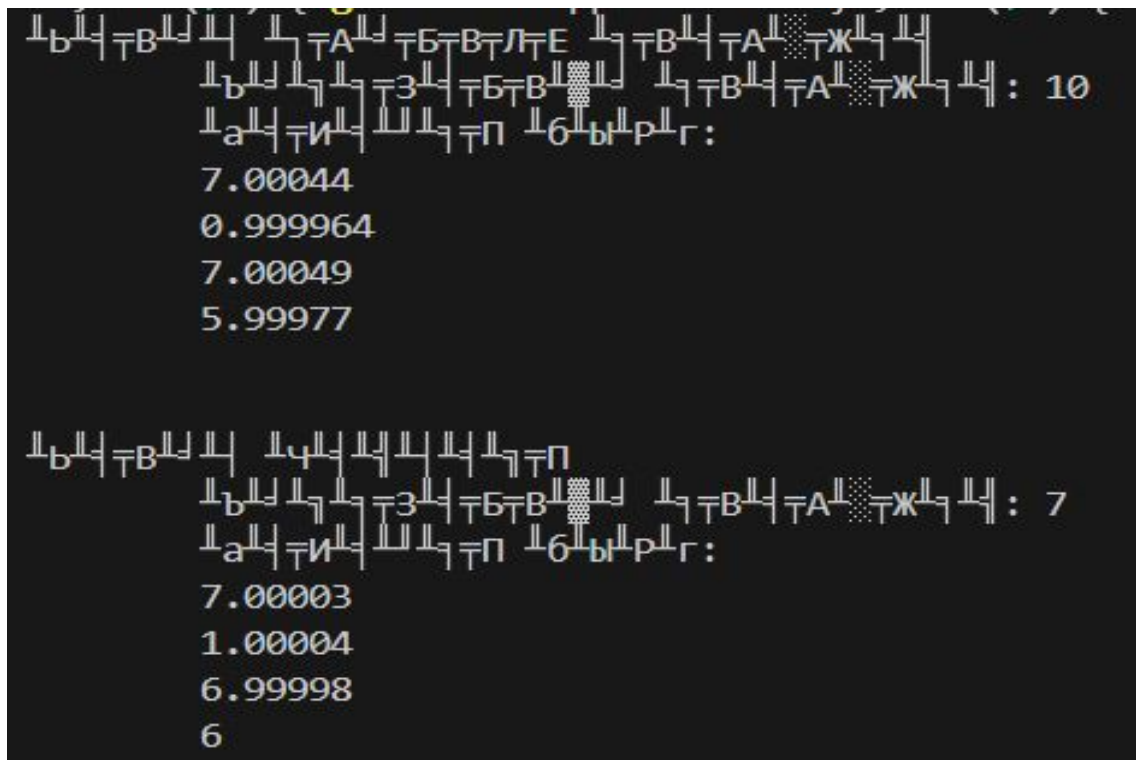
7 Постановка задачи

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: 8

$$\begin{cases} -7x_1 - 1x_2 + 2x_3 + 2x_4 = -24 \\ 3x_1 - 20x_2 - 8x_4 = -47 \\ -9x_1 + x_2 + 18x_3 - 6x_4 = 28 \\ -x_1 - 1x_3 - 6x_4 = -50 \end{cases}$$

8 Результаты работы



```
Матрица A:
7.00044
0.999964
7.00049
5.99977

Вектор b:
-24
-47
28
-50

Точность: 1e-06

Метод Зейделя:
Итерации: 10
Решение:
7.00003
1.00004
6.99998
6
```

Рис. 3: Вывод программы в консоли

9 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using double_matrix = vector<vector<double>>>;
5
6
7  double_matrix matr_plus(const double_matrix& matrix1, const double_matrix& matrix2) {
8      int n = matrix1.size();
9      int m = matrix1[0].size();
10
11      double_matrix 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          }
16      }
17
18      return res;
19 }
20
21
22 double_matrix mult(const double_matrix& matrix1, const double_matrix& matrix2) {
23     int n1 = matrix1.size(), m1 = matrix1[0].size(), n2 = matrix2.size(), m2 = matrix2
24     [0].size();
25     double_matrix res(n1, vector<double>(m2, 0));
26
27     for (int i = 0; i < n1; ++i)
28         for (int j = 0; j < m2; ++j) {
29             double cntr = 0;
30             for (int k = 0; k < m1; ++k)
31                 cntr += matrix1[i][k] * matrix2[k][j];
32             res[i][j] = cntr;
33         }
34
35     return res;
36 }
37
38 void matrix_preprocess(double_matrix& coeffs, double_matrix& r) {
39     double n = coeffs.size(), m = coeffs[0].size();
40     for(int i=0; i < n; i++) {
41         double a = coeffs[i][i];
42         if (a == 0)
43             continue;
44         r[i][0] /= a;
45         for (int j=0; j < m; j++)
46             if (i == j)
```

```

47         coeffs[i][j] = 0;
48     else
49         coeffs[i][j] /= -a;
50 }
51 }
52
53
54 double eps_curr(const double_matrix& vect1, const double_matrix& vect2) {
55     double eps = 0;
56     for(int i=0; i<vect1.size(); i++)
57         eps += pow(vect1[i][0] - vect2[i][0], 2);
58     return sqrt(eps);
59 }
60
61
62 void cout_matrix(const double_matrix& matrix1) {
63     for(const auto& vect: matrix1) {
64         cout << '\t';
65         for (auto x: vect)
66             cout << x << " ";
67         cout << endl;
68     }
69 }
70
71
72 pair<int, double_matrix> iterations(double_matrix& a, double_matrix& b, double EPS0) {
73     double_matrix p_x = b, c_x;
74     int k = 0;
75     while (eps_curr(c_x = matr_plus(b, mult(a, p_x)), p_x) > EPS0) {
76         k += 1;
77         p_x = c_x;
78     }
79     return {k, p_x};
80 }
81
82
83 pair<int, double_matrix> seidel(const double_matrix& a, const double_matrix& b, double
    EPS0) {
84     double_matrix p_x = b, c_x = p_x;
85     bool flag = true;
86     int k = 0;
87     double eps = 0;
88     while (flag or eps > EPS0) {
89         k += 1;
90         flag = false;
91         for(int i = 0; i < b.size(); i++) {
92             c_x[i][0] = 0;
93             for(int j=0; j < b.size(); j++) {
94                 if (i == j)

```

```

95         c_x[i][0] += b[i][0];
96     else if (i < j)
97         c_x[i][0] += a[i][j]*p_x[j][0];
98     else
99         c_x[i][0] += a[i][j]*c_x[j][0];
100     }
101 }
102 eps = eps_curr(c_x, p_x);
103 p_x = c_x;
104 }
105 return {k, p_x};
106 }
107
108
109 int main() {
110     double_matrix coeff_matrix = {
111         {-7, -1, 2, 2},
112         {3, -20, 0, -8},
113         {-9, 1, 18, -6},
114         {-1, 0, -1, -6}
115     };
116
117     double_matrix roots = {
118         {-24},
119         {-47},
120         {28},
121         {-50}
122     };
123
124     matrix_preprocess(coeff_matrix, roots);
125
126     int last_iter;
127     double_matrix res_matr;
128
129     tie(last_iter, res_matr) = iterations(coeff_matrix, roots, 0.001);
130     cout << " " << endl << "\t : " << last_iter << endl << "\t : " << endl;
131     cout_matrix(res_matr);
132
133     cout << endl << endl;
134
135     tie(last_iter, res_matr) = seidel(coeff_matrix, roots, 0.001);
136     cout << " " << endl << "\t : " << last_iter << endl << "\t : " << endl;
137     cout_matrix(res_matr);
138 }

```

1.4 Метод вращений

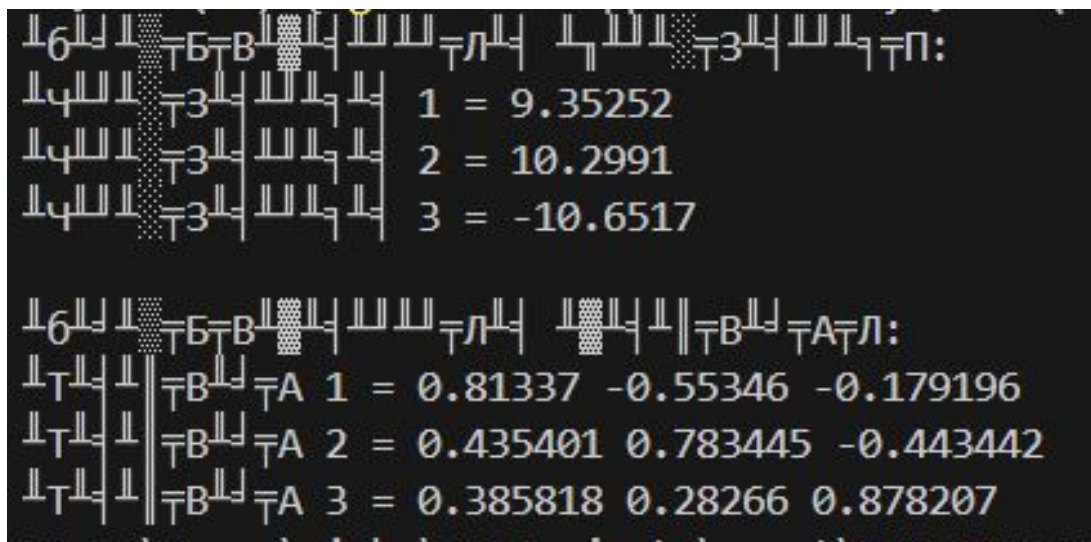
10 Постановка задачи

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: 8

$$\begin{pmatrix} 9 & -2 & 3 \\ -2 & 6 & 8 \\ 3 & 8 & -6 \end{pmatrix}$$

11 Результаты работы



```
Входная матрица A:
9.000000 -2.000000 3.000000
-2.000000 6.000000 8.000000
3.000000 8.000000 -6.000000

Собственные значения:
1 = 9.35252
2 = 10.2991
3 = -10.6517

Собственные векторы:
1 = 0.81337 -0.55346 -0.179196
2 = 0.435401 0.783445 -0.443442
3 = 0.385818 0.28266 0.878207
```

Рис. 4: Вывод программы в консоли

12 Исходный код

```
1 | #include <bits/stdc++.h>
2 |
3 | using namespace std;
4 | using double_matrix = vector<vector<double> >;
5 |
6 |
7 | pair<int, int> get_indexes(const double_matrix& matrix1){
8 |     double k = matrix1[0][1];
9 |     pair<int, int> indexes = make_pair(0, 1);
10 |    int n = matrix1.size(), m = matrix1[0].size();
11 |    for (int i = 0; i < n; i++)
12 |        for (int j = i+1; j < m; j++)
13 |            if (abs(matrix1[i][j]) > k) {
14 |                k = abs(matrix1[i][j]);
15 |                indexes = make_pair(i, j);
16 |            }
17 |    return indexes;
18 | }
19 |
20 |
21 | double ugol(double aji, double aii, double ajj){
22 |     if (aii == ajj)
23 |         return 3.14/4;
24 |     return atan(2*aji/(aii-ajj)) / 2;
25 | }
26 |
27 |
28 | double_matrix mult(const double_matrix& matrix1, const double_matrix& matrix2) {
29 |     int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
30 |     double_matrix res(n1);
31 |     for (int i=0; i<n1; i++)
32 |         for (int j=0; j<m2; j++)
33 |             res[i].push_back(0);
34 |
35 |     for (int i=0; i<n1; i++) {
36 |         for (int j=0; j<m2; j++) {
37 |             double cntr = 0;
38 |             for (int k=0; k<m1; k++)
39 |                 cntr += matrix1[i][k] * matrix2[k][j];
40 |             res[i][j] = cntr;
41 |         }
42 |     }
43 |     return res;
44 | }
45 |
46 |
47 | double_matrix ones_matr(int n){
```

```

48     double_matrix E(n, vector<double>(n, 0));
49     for(int i=0; i<n; i++)
50         E[i][i] = 1;
51     return E;
52 }
53
54
55 double_matrix u_matr(const double_matrix& matrix1){
56     double_matrix res = ones_matr(matrix1.size());
57     int i_max, j_max;
58     tie(i_max, j_max) = get_indexes(matrix1);
59     double phi = ugol(matrix1[i_max][j_max], matrix1[i_max][i_max], matrix1[j_max][
        j_max]);
60     res[i_max][i_max] = cos(phi);
61     res[j_max][j_max] = cos(phi);
62     res[i_max][j_max] = -sin(phi);
63     res[j_max][i_max] = sin(phi);
64     return res;
65 }
66
67
68 double deviation(const double_matrix& matrix1){
69     double eps = 0;
70     int n=matrix1.size(), m=matrix1[0].size();
71     for (int i=0; i<n; i++)
72         for (int j=i+1; j<m; j++)
73             eps += matrix1[i][j]*matrix1[i][j];
74     return sqrt(eps);
75 }
76
77
78 double_matrix trans(const double_matrix& matrix1){
79     int n = matrix1.size(), m = matrix1[0].size();
80     double_matrix res(m, vector<double>(n));
81     for (int i=0; i<n; i++)
82         for (int j=0; j<m; j++)
83             res[j][i] = matrix1[i][j];
84     return res;
85 }
86
87
88 pair<vector<double>, double_matrix> jacobi(double_matrix& coeff_matrix, double EPS){
89     int n = coeff_matrix.size();
90     double_matrix evec = ones_matr(n);
91     while (EPS <= deviation(coeff_matrix)){
92         double_matrix u = u_matr(coeff_matrix);
93         evec = mult(evec, u);
94         double_matrix trans_u = trans(u);
95         coeff_matrix = mult(mult(trans_u, coeff_matrix), u);

```

```

96     }
97
98     vector<double> eval(n);
99     for (int i=0; i<n; i++)
100         eval[i] = coeff_matrix[i][i];
101     return make_pair(eval, evec);
102 }
103
104
105 int main() {
106     double_matrix coeff_matrix{{9, -2, 3}, {-2, 6, 8}, {3, 8, -6}};
107     vector<double> eval;
108     double_matrix evec;
109
110     tie(eval, evec) = jacobi(coeff_matrix, 0.01);
111     int n = eval.size();
112
113     cout << " : " << endl;
114     for (int i=0; i<n; i++)
115         cout << " " << i+1 << " = " << eval[i] << endl;
116     cout << endl;
117
118     cout << " : " << endl;
119     for (int i=0; i<n; i++){
120         cout << " " << i+1 << " = ";
121         for(double element: evec[i])
122             cout << element << " ";
123         cout << endl;
124     }
125
126     return 0;
127 }

```


1.5 QR – разложение матриц

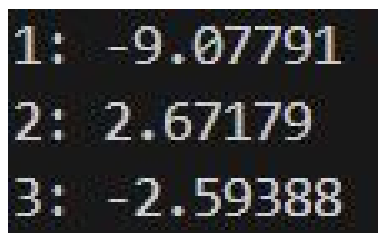
13 Постановка задачи

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: 8

$$\begin{pmatrix} -9 & 2 & 2 \\ -2 & 0 & 7 \\ 8 & 2 & 0 \end{pmatrix}$$

14 Результаты работы



```
1: -9.07791
2: 2.67179
3: -2.59388
```

Рис. 5: Вывод программы в консоли

15 Исходный код

```
1 | #include <bits/stdc++.h>
2 |
3 | using namespace std;
4 | using double_matrix = vector<vector<double>>>;
5 | using dvect = vector<double>;
6 |
7 | double_matrix trans(const double_matrix& matrix1){
8 |     int n = matrix1.size(), m = matrix1[0].size();
9 |     double_matrix res(m, dvect(n));
10 |     for (int i=0; i<n; i++)
11 |         for (int j=0; j<m; j++)
12 |             res[j][i] = matrix1[i][j];
13 |     return res;
14 | }
15 |
16 | double_matrix matr_plus(const double_matrix& matrix1, const double_matrix& matrix2) {
17 |     double_matrix res;
18 |     int n = matrix1.size();
19 |     for (int i = 0; i < n; i++) {
20 |         dvect 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 | double_matrix multiplone_matr(double_matrix& matrix1, double_matrix& matrix2) {
31 |     int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
32 |     double_matrix res(n1);
33 |     for (int i=0; i<n1; i++)
34 |         for (int j=0; j<m2; j++)
35 |             res[i].push_back(0);
36 |
37 |     for (int i=0; i<n1; i++) {
38 |         for (int j=0; j<m2; j++) {
39 |             double cntr = 0;
40 |             for (int k=0; k<m1; k++)
41 |                 cntr += matrix1[i][k] * matrix2[k][j];
42 |             res[i][j] = cntr;
43 |         }
44 |     }
45 |     return res;
46 | }
47 |
```

```

48 | int sign(double a){
49 |     return (a >= 0) ? 1 : -1;
50 | }
51 |
52 | double deviation(const double_matrix& matrix1){
53 |     double eps = 0;
54 |     int n = matrix1.size();
55 |     for(int i=0; i<n; i++)
56 |         for(int j=0; j<i-1; j++)
57 |             eps += matrix1[i][j]*matrix1[i][j];
58 |     return sqrt(eps);
59 | }
60 |
61 | double_matrix get_one_matr(int n){
62 |     double_matrix E(n, dvect(n, 0));
63 |     for(int i=0; i<n; i++)
64 |         E[i][i] = 1;
65 |     return E;
66 | }
67 |
68 | double_matrix get_H_matrix(const double_matrix& coeffts, int ind){
69 |     int n = coeffts.size();
70 |     double_matrix v(n, dvect(1));
71 |
72 |     for(int i=0; i<n; i++){
73 |         if (i < ind)
74 |             v[i][0] = 0;
75 |         else if (i == ind){
76 |             double sum = 0;
77 |             for (int j=ind; j < n; j++)
78 |                 sum += coeffts[j][i]*coeffts[j][i];
79 |             v[i][0] = coeffts[i][i] + sign(coeffts[i][i]) * sqrt(sum);
80 |         }
81 |         else
82 |             v[i][0] = coeffts[i][ind];
83 |     }
84 |
85 |     double_matrix trans_v = trans(v);
86 |     double k = -multiplone_matr(trans_v, v)[0][0]/2;
87 |     v = multiplone_matr(v, trans_v);
88 |     for (int i=0; i<n; i++)
89 |         for (int j=0; j<n; j++)
90 |             v[i][j] /= k;
91 |
92 |     double_matrix E = get_one_matr(n);
93 |     return matr_plus(E, v);
94 | }
95 |
96 | pair<double_matrix, double_matrix> decompose(const double_matrix& coeff){

```

```

97     double_matrix coeffts = coeff;
98     double_matrix Q = get_H_matrix(coeffts, 0);
99     coeffts = multiplone_matr(Q, coeffts);
100     int n = coeffts.size();
101     for (int i=1; i<n-1; i++){
102         double_matrix H = get_H_matrix(coeffts, i);
103         Q = multiplone_matr(Q, H);
104         coeffts = multiplone_matr(H, coeffts);
105     }
106     return make_pair(Q, coeffts);
107 }
108
109 dvect evals_calc(double_matrix& coeffts, double EPS){
110     while (deviation(coeffts) > EPS){
111         pair<double_matrix, double_matrix> QR = decompose(coeffts);
112         coeffts = multiplone_matr(QR.second, QR.first);
113     }
114     int n = coeffts.size();
115     dvect r(n);
116     for (int i=0; i<n; i++)
117         r[i] = coeffts[i][i];
118     return r;
119 }
120
121 int main() {
122     double_matrix coefft_matrix{{-9, 2, 2}, {-2, 0, 7}, {8, 2, 0}};
123     dvect evals = evals_calc(coefft_matrix, 0.01);
124     for (int i=0; i<evals.size(); i++)
125         cout << " " << i+1 << ": " << evals[i] << endl;
126 }

```