

**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: Т. В. Мохнач
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

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: 15

$$\begin{cases} -9x_1 + 8x_2 + 8x_3 + 6x_4 = -81 \\ -7x_1 - 9x_2 + 5x_3 + 4x_4 = -50 \\ -3x_1 - x_2 + 8x_3 = -69 \\ 3x_1 - x_2 - 4x_3 - 5x_4 = -5 \end{cases}$$

2 Результаты работы

```
≡ answer.txt X
Lab1 > lab 1_1 > ≡ answer.txt
1 LU matrices:
2 L:
3 1.00 0.00 0.00 0.00
4 0.78 1.00 0.00 0.00
5 0.33 0.24 1.00 0.00
6 -0.33 -0.11 -0.26 1.00
7 U:
8 -9.00 8.00 8.00 6.00
9 0.00 -15.22 -1.22 -0.67
10 0.00 0.00 5.63 -1.84
11 0.00 0.00 0.00 -3.55
12
13 Решение системы:
14 -1.00
15 0.00
16 -9.00
17 -3.00
18
19 Определитель: -2739.00
20
21 Обратная матрица:
22 -0.14 -0.11 0.08 -0.25
23 0.06 -0.06 -0.01 0.02
24 -0.05 -0.05 0.15 -0.09
25 -0.06 -0.01 -0.07 -0.28
26
```

Рис. 1: Вывод программы

3 Исходный код

```
1  #include<iostream>
2  #include<conio.h>
3  #include <fstream>
4  #include <vector>
5
6  using namespace std;
7  const int n = 4;
8
9  istream& operator>>(istream& stream, float a[n][n])
10 {
11     for(int i= 0;i<n;i++)
12     {
13         for(int j=0;j<n;j++)
14             stream >> a[i][j];
15     }
16     return stream;
17 }
18 ostream& operator<<(ostream& stream, float a[n][n])
19 {
20     for (int i = 0; i < n; i++)
21     {
22         for (int j = 0; j < n; j++)
23             stream << a[i][j] << " ";
24         stream << endl;
25     }
26     return stream;
27 }
28 istream& operator>>(istream& stream, float a[n])
29 {
30     for(int i=0;i<n;i++)
31         stream >> a[i];
32     return stream;
33 }
34 ostream& operator<<(ostream& stream, float a[n])
35 {
36     for (int i = 0; i < n; i++)
37     {
38         stream << a[i] << endl;
39     }
40     return stream;
41 }
42
43 float det_from_U(float u[n][n])
44 {
45     double det = 1;
46     for (int i = 0; i < n; i++)
47         det *= u[i][i];
48     return det;
49 }
50
51 void solve_eq_for_L(float l[n][n], const float b[n], float y[n]) {
52     for (int i = 0; i < n; ++i) {
53         y[i] = b[i];
54         for (int j = 0; j < i; ++j)
55             y[i] -= l[i][j] * y[j];
56     }
57 }
58
59 void solve_eq_for_U(float u[n][n], float z[n], float x[n]){
60     float E[n];
```

```

61     for (int i = 0; i < n; ++i) {
62         E[i] = 1;
63         for (int i = n - 1; i >= 0; --i) {
64             x[i] = z[i];
65             for (int j = i + 1; j < n; ++j)
66                 x[i] -= u[i][j] * x[j];
67             x[i] /= u[i][i];
68         }
69     }
70 }
71
72 void inverse(float const a[n][n], float l[n][n], float u[n][n], float inversed[n][n])
73 {
74     float E[n] = {0};
75     for (int i = 0; i < n; ++i) {
76         E[i] = 1;
77         float Y_E[n] = {0}, X_E[n] = {0};
78         solve_eq_for_L(l, E, Y_E);
79         solve_eq_for_U(u, Y_E, X_E);
80         for (int j = 0; j < n; ++j)
81             inversed[j][i] = X_E[j];
82         E[i] = 0;
83     }
84 }
85
86 void LU_Decomposite(float const a[n][n], float u[n][n], float l[n][n]) {
87
88     for (int i = 0; i < n; ++i) {
89         for (int j = 0; j < n; ++j) {
90             u[i][j] = a[i][j];
91         }
92     }
93
94     for(int i = 0; i < n; i++)
95         for(int j = i; j < n; j++)
96             l[j][i] = l[j][i] / u[i][i];
97
98     for(int k = 1; k < n; k++)
99     {
100         for(int i = k-1; i < n; i++)
101             for(int j = i; j < n; j++)
102                 l[j][i] = u[j][i] / u[i][i];
103
104         for(int i = k; i < n; i++)
105             for(int j = k-1; j < n; j++)
106                 u[i][j] = u[i][j] - l[i][k-1] * u[k-1][j];
107     }
108 }
109
110 int main()
111 {
112     int i,k,j,p;
113     float a[n][n], l[n][n] = {0}, u[n][n] = {0}, sum, b[n], z[n] = {0}, x[n] = {0};
114
115     ofstream file_ans("answer.txt");
116     file_ans.precision(2);
117     file_ans << fixed;
118
119     ifstream file_A("A_matrix.txt"), file_b("b_vector.txt");
120     file_A >> a;
121     file_b >> b;
122

```

```

123
124 // LU decomposition
125 LU_Decomposite(a, u ,l);
126 // Displaying LU matrix
127 file_ans<<"LU matrices: "<< endl;
128 file_ans << "L:" << endl << l;
129 file_ans<<"U:" << endl << u << endl;
130
131 //Finding Z; Lz=b
132 solve_eq_for_L(l,b, z);
133 //Finding X; Ux=Z
134 solve_eq_for_U(u,z,x);
135 //Finding inversed matrix
136 float inversed[n][n];
137 inverse(a, l, u, inversed);
138 //Solution
139 file_ans << "Решение системы:\n" << x;
140 file_ans << "\Определительn:" << det_from_U(u) << endl;
141 file_ans << "\Обратная матрица:\n" << inversed;
142 return 0;
143 }

```

2 Метод прогонки

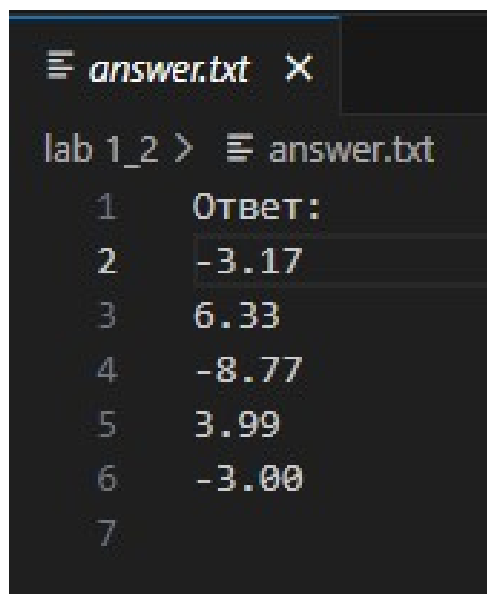
1 Постановка задачи

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: 15

$$\begin{cases} 16x_1 - 8x_2 = 0 \\ -7x_1 - 16x_2 + 5x_3 = -123 \\ 4x_2 + 12x_3 + 3x_4 = -68 \\ -4x_3 + 12x_4 - 7x_5 = 104 \\ -x_4 + 7x_5 = 20 \end{cases}$$

2 Результаты работы



```
lab 1_2 > ≡ answer.txt
1      Ответ:
2      -3.17
3      6.33
4      -8.77
5      3.99
6      -3.00
7
```

Рис. 2: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4
5  using namespace std;
6
7  int n = 5;
8  class matrix
9  {
10     private:
11         vector <vector <double>> obj;
12     public:
13         int cols = 0, rows = 0;
14
15         matrix() {}
16         matrix(int _rows, int _cols)
17         {
18             rows = _rows;
19             cols = _cols;
20             obj = vector <vector <double>>(rows, vector <double>(cols));
21         }
22
23         vector <double>& operator[](int i)
24         {
25             return obj[i];
26         }
27
28         operator double()
29         {
30             return obj[0][0];
31         }
32 };
33
34 istream& operator>>(istream& stream, matrix& m)
35 {
36     for (int i = 0; i < m.rows; i++)
37     {
38         for (int j = 0; j < m.cols; j++)
39             stream >> m[i][j];
40     }
41     return stream;
42 }
43
44 ostream& operator<<(ostream& stream, matrix m)
45 {
46     for (int i = 0; i < m.rows; i++)
47     {
48         for (int j = 0; j < m.cols; j++)
49             stream << m[i][j] << ' ';
50         stream << '\n';
51     }
52     return stream;
53 }
54
55
56 matrix solve_SLE (matrix& A, matrix& b)
57 {
58     vector <double> p(n), q(n);
59     matrix ans(n, 1);
60     p[0] = -A[0][1] / A[0][0];
```

```

61 | q[0] = b[0][0] / A[0][0];
62 | for (int i = 1; i < n; i++)
63 | {
64 |     if (i != n - 1)
65 |         p[i] = -A[i][i + 1] / (A[i][i] + A[i][i - 1] * p[i - 1]);
66 |     else
67 |         p[i] = 0;
68 |     q[i] = (b[i][0] - A[i][i - 1] * q[i - 1]) / (A[i][i] + A[i][i - 1] * p[i - 1]);
69 | }
70 | ans[n - 1][0] = q[n - 1];
71 | for (int i = n - 2; i >= 0; i--)
72 |     ans[i][0] = p[i] * ans[i + 1][0] + q[i];
73 | return ans;
74 | }
75 |
76 | int main()
77 | {
78 |     matrix A(n, n), b(n, 1);
79 |     ofstream fout("answer.txt");
80 |     fout.precision(2);
81 |     fout << fixed;
82 |     ifstream fin_A("A_matrix.txt"), fin_b("b_column.txt");
83 |     fin_A >> A;
84 |     fin_b >> b;
85 |     //Finding an answer
86 |     matrix ans;
87 |     ans = solve_SLE(A, b);
88 |     fout << "Ответ:\n" << ans;
89 | }

```


3 Метод простых итераций. Метод Зейделя

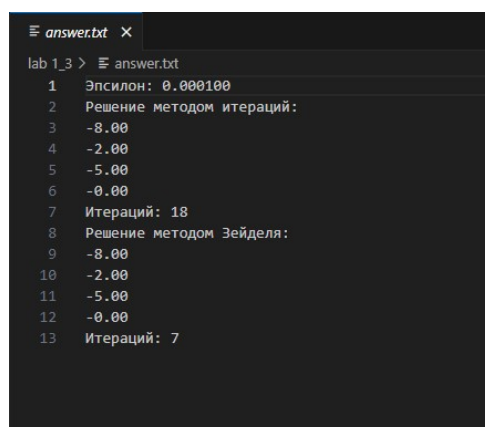
1 Постановка задачи

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: 15

$$\begin{cases} -14x_1 + 6x_2 + x_3 - 5x_4 = 95 \\ -6x_1 + 27x_2 + 7x_3 - 6x_4 = -41 \\ 7x_1 - 5x_2 - 23x_3 - 8x_4 = 69 \\ 7x_1 - 5x_2 - 23x_3 - 8x_4 = 27 \end{cases}$$

2 Результаты работы



```
lab 1_3 > answer.txt
1 Эпсилон: 0.000100
2 Решение методом итераций:
3 -8.00
4 -2.00
5 -5.00
6 -0.00
7 Итераций: 18
8 Решение методом Зейделя:
9 -8.00
10 -2.00
11 -5.00
12 -0.00
13 Итераций: 7
```

Рис. 3: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <string>
5
6  using namespace std;
7
8  int n = 4;
9  class matrix
10 {
11     private:
12         vector <vector <double>> obj;
13     public:
14         int cols = 0, rows = 0;
15
16         matrix() {}
17         matrix(int _rows, int _cols)
18         {
19             rows = _rows;
20             cols = _cols;
21             obj = vector <vector <double>>(rows, vector <double>(cols));
22         }
23
24         operator double()
25         {
26             return obj[0][0];
27         }
28
29         vector <double>& operator[](int i)
30         {
31             return obj[i];
32         }
33         double get_abs()
34         {
35             double mx = 0;
36             for (int i = 0; i < rows; i++)
37             {
38                 double s = 0;
39                 for (int j = 0; j < cols; j++)
40                     s += std::abs(obj[i][j]);
41                 mx = max(mx, s);
42             }
43             return mx;
44         }
45 };
46
47
48 //Определение**** операторов****//
49 istream& operator>>(istream& stream, matrix& m)
50 {
51     for (int i = 0; i < m.rows; i++)
52     {
53         for (int j = 0; j < m.cols; j++)
54             stream >> m[i][j];
55     }
56     return stream;
57 }
58
59 ostream& operator<<(ostream& stream, matrix m)
60 {
```

```

61     for (int i = 0; i < m.rows; i++)
62     {
63         for (int j = 0; j < m.cols; j++)
64             stream << m[i][j] << ' ';
65         stream << '\n';
66     }
67     return stream;
68 }
69
70 matrix operator*(double a, matrix b)
71 {
72     for (int i = 0; i < b.rows; i++)
73     {
74         for (int j = 0; j < b.cols; j++)
75             b[i][j] *= a;
76     }
77     return b;
78 }
79
80 matrix operator+(matrix a, matrix b)
81 {
82     if (a.rows != b.rows || b.cols != a.cols)
83         return matrix(0, 0);
84     matrix res(a.rows, a.cols);
85     for (int i = 0; i < b.rows; i++)
86     {
87         for (int j = 0; j < res.cols; j++)
88             res[i][j] = a[i][j] + b[i][j];
89     }
90     return res;
91 }
92
93 matrix operator-(matrix a, matrix b)
94 {
95     if (a.rows != b.rows || b.cols != a.cols)
96         return matrix(0, 0);
97     matrix res(a.rows, a.cols);
98     for (int i = 0; i < b.rows; i++)
99     {
100         for (int j = 0; j < res.cols; j++)
101             res[i][j] = a[i][j] - b[i][j];
102     }
103     return res;
104 }
105
106 matrix operator*(matrix a, matrix b)
107 {
108     if (a.cols != b.rows)
109         return matrix(0, 0);
110     matrix res(a.rows, b.cols);
111     for (int i = 0; i < res.rows; i++)
112     {
113         for (int j = 0; j < res.cols; j++)
114         {
115             res[i][j] = 0;
116             for (int k = 0; k < a.cols; k++)
117                 res[i][j] += a[i][k] * b[k][j];
118         }
119     }
120     return res;
121 }
122

```

```

123 //Решение*****//
124 void set_alpha_beta(matrix& alpha, matrix& beta, matrix a, matrix b) {
125     for (int i = 0; i < n; i++)
126     {
127         for (int j = 0; j < n; j++)
128             alpha[i][j] = -a[i][j] / a[i][i];
129         alpha[i][i] = 0;
130     }
131     for (int i = 0; i < n; i++)
132         beta[i][0] = b[i][0] / a[i][i];
133 }
134 matrix solve_SLE_iterations(matrix a, matrix b, double eps, int& iterations_number)
135 {
136     matrix alpha(n, n), beta(n, 1);
137     set_alpha_beta(alpha, beta, a, b);
138
139     matrix ans = beta, diff;
140     double m = abs(a);
141     double cur = m;
142     double eps_k = 2 * eps;
143     iterations_number = 0;
144
145     while (eps_k > eps)
146     {
147         matrix prev = ans;
148         ans = beta + alpha * ans;
149         diff = ans - prev;
150         if (m < 1)
151             eps_k = cur / (1 - m) * diff.get_abs();
152         else
153             eps_k = diff.get_abs();
154         cur = cur * m;
155         iterations_number++;
156     }
157     return ans;
158 }
159
160 matrix solve_SLE_seidel(matrix a, matrix b, double eps, int& iterations_number)
161 {
162     matrix alpha(n, n), beta(n, 1);
163     set_alpha_beta(alpha, beta, a, b);
164
165     matrix ans = beta, diff;
166     double m = abs(a);
167     double cur = m;
168     double eps_k = 2 * eps;
169     iterations_number = 0;
170     while (eps_k > eps)
171     {
172         matrix prev = ans;
173         for (int i = 0; i < n; i++)
174         {
175             double cur = beta[i][0];
176             for (int j = 0; j < n; j++)
177                 cur += alpha[i][j] * ans[j][0];
178             ans[i][0] = cur;
179         }
180         diff = ans - prev;
181         if (m < 1)
182             eps_k = cur / (1 - m) * diff.get_abs();
183         else
184             eps_k = diff.get_abs();

```

```

185         cur = cur * m;
186         iterations_number++;
187     }
188     return ans;
189 }
190
191 int main()
192 {
193     matrix A(n, n), b(n, 1);
194     float eps;
195     ofstream fout("answer.txt");
196     fout.precision(2);
197     fout << fixed;
198     ifstream fin_A("A_matrix.txt"), fin_b("b_column.txt"), fin_eps("accuracy.txt");
199     fin_A >> A;
200     fin_b >> b;
201     fin_eps >> eps;
202
203     int iterations_number = 0;
204     fout << "Эпсилон: " << to_string(eps) << endl;
205     fout << "Решение методом итераций:\n" << solve_SLE_iterations(A, b, eps, iterations_number) << "
        Итераций: ";
206     fout << iterations_number;
207     fout << "\n" << "Решение методом Зейделя:\n" << solve_SLE_seidel(A, b, eps, iterations_number) << "
        Итераций: ";
208     fout << iterations_number;
209 }

```

4 Метод вращений

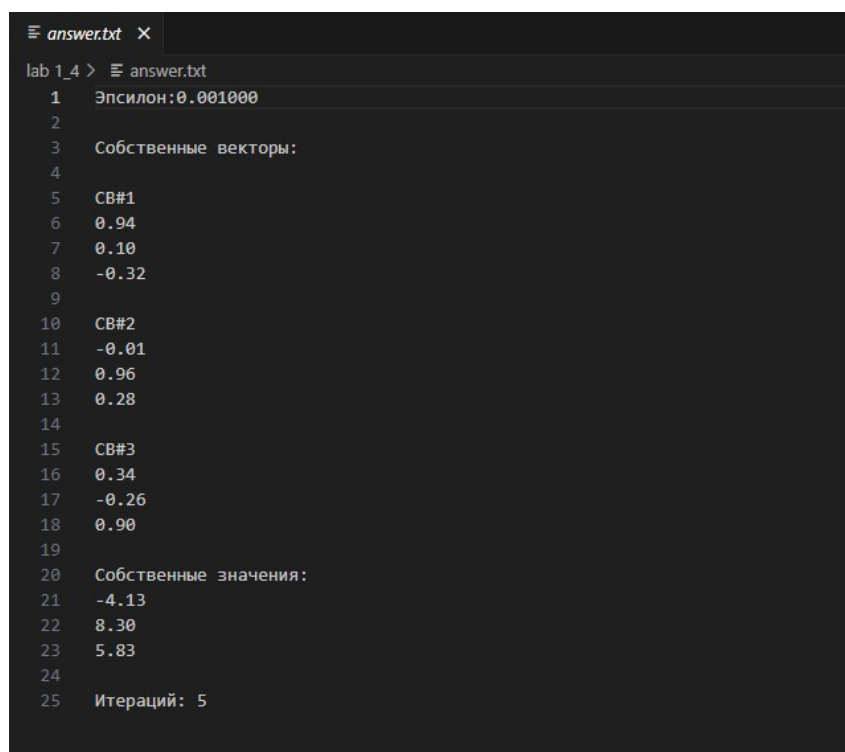
1 Постановка задачи

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: 15

$$\begin{pmatrix} -3 & -1 & 3 \\ -1 & 8 & 1 \\ 3 & 1 & 5 \end{pmatrix}$$

2 Результаты работы



```
answer.txt x
lab 1_4 > answer.txt
1  Эпсилон:0.001000
2
3  Собственные векторы:
4
5  СВ#1
6  0.94
7  0.10
8  -0.32
9
10 СВ#2
11 -0.01
12 0.96
13 0.28
14
15 СВ#3
16 0.34
17 -0.26
18 0.90
19
20 Собственные значения:
21 -4.13
22 8.30
23 5.83
24
25 Итераций: 5
```

Рис. 4: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <string>
5
6  using namespace std;
7
8  double pi = acos(-1);
9  int n = 3;
10
11 class matrix
12 {
13     private:
14         vector <vector <double>> obj;
15     public:
16         int cols = 0, rows = 0;
17
18         matrix() {}
19         matrix(int _rows, int _cols)
20         {
21             rows = _rows;
22             cols = _cols;
23             obj = vector <vector <double>>(rows, vector <double>(cols));
24         }
25         matrix get_transposed()
26         {
27             matrix result(cols, rows);
28             for (int i = 0; i < rows; i++)
29             {
30                 for (int j = 0; j < cols; j++)
31                     result[j][i] = obj[i][j];
32             }
33             return result;
34         }
35
36         vector <double>& operator[](int i)
37         {
38             return obj[i];
39         }
40
41         operator double()
42         {
43             return obj[0][0];
44         }
45     };
46
47 istream& operator>>(istream& stream, matrix& m)
48 {
49     for (int i = 0; i < m.rows; i++)
50     {
51         for (int j = 0; j < m.cols; j++)
52             stream >> m[i][j];
53     }
54     return stream;
55 }
56
57 ostream& operator<<(ostream& stream, matrix m)
58 {
59     for (int i = 0; i < m.rows; i++)
60     {
```

```

61         for (int j = 0; j < m.cols; j++)
62             stream << m[i][j] << ' ';
63         stream << '\n';
64     }
65     return stream;
66 }
67
68
69 matrix operator*(matrix a, matrix b)
70 {
71     matrix result(a.rows, b.cols);
72     for (int i = 0; i < result.rows; i++)
73     {
74         for (int j = 0; j < result.cols; j++)
75         {
76             result[i][j] = 0;
77             for (int k = 0; k < a.cols; k++)
78                 result[i][j] += a[i][k] * b[k][j];
79         }
80     }
81     return result;
82 }
83
84 matrix get_rotation_matrix(matrix A, int i, int j) {
85
86     matrix rotation(n, n);
87     double phi = pi / 4;
88     if (abs(A[i][i] - A[j][j]) > 0.0000001)
89         phi = 0.5 * atan((2 * A[i][j]) / (A[i][i] - A[j][j]));
90
91     for (int i = 0; i < n; i++)
92         rotation[i][i] = 1;
93
94     rotation[i][i] = cos(phi);
95     rotation[i][j] = -sin(phi);
96     rotation[j][j] = cos(phi);
97     rotation[j][i] = sin(phi);
98     return rotation;
99 }
100
101 int turns_method(matrix A, matrix& self_vec, matrix& self_value, double eps)
102 {
103     int iterations_number = 0;
104     double eps_k = 2 * eps;
105     for (int i = 0; i < n; i++)
106         self_vec[i][i] = 1;
107
108     while (eps_k > eps)
109     {
110         int i_max = 1, j_max = 0;
111         for (int i = 0; i < n; i++)
112         {
113             for (int j = 0; j < i; j++)
114             {
115                 if (abs(A[i_max][j_max]) < abs(A[i][j]))
116                 {
117                     i_max = i;
118                     j_max = j;
119                 }
120             }
121         }
122         matrix rotation = get_rotation_matrix(A, i_max, j_max);

```



```

123
124     self_vec = self_vec * rotation;
125     A = rotation.get_transposed() * A * rotation;
126     eps_k = 0;
127
128     for (int i = 0; i < n; i++)
129     {
130         for (int j = 0; j < i; j++)
131             eps_k += A[i][j] * A[i][j];
132     }
133     eps_k = sqrt(eps_k);
134     iterations_number++;
135 }
136 for (int i = 0; i < n; i++)
137     self_value[i][0] = A[i][i];
138 return iterations_number;
139 }
140
141 int main()
142 {
143     matrix A(n, n), self_vec(n,n), self_value(n,1);
144     double eps;
145     ofstream fout("answer.txt");
146     fout.precision(2);
147     fout << fixed;
148     ifstream fin("input.txt");
149     fin >> eps;
150     fin >> A;
151     int iterations_number = turns_method(A, self_vec, self_value, eps);
152     fout << "Эпсилон:" << to_string(eps) << endl << endl;
153     fout << "Собственные векторы:\n" << endl;
154     for (int i = 0; i < n; i++) {
155         fout << "CB#" << i+1 << endl;
156         for (int j = 0; j < n; j++) {
157             fout << self_vec[j][i] << endl ;
158         }
159         fout << endl;
160     }
161     fout << "Собственные значения:\n" << self_value << "\Итерацийn: ";
162     fout << iterations_number;
163 }

```

5 QR – разложение матриц

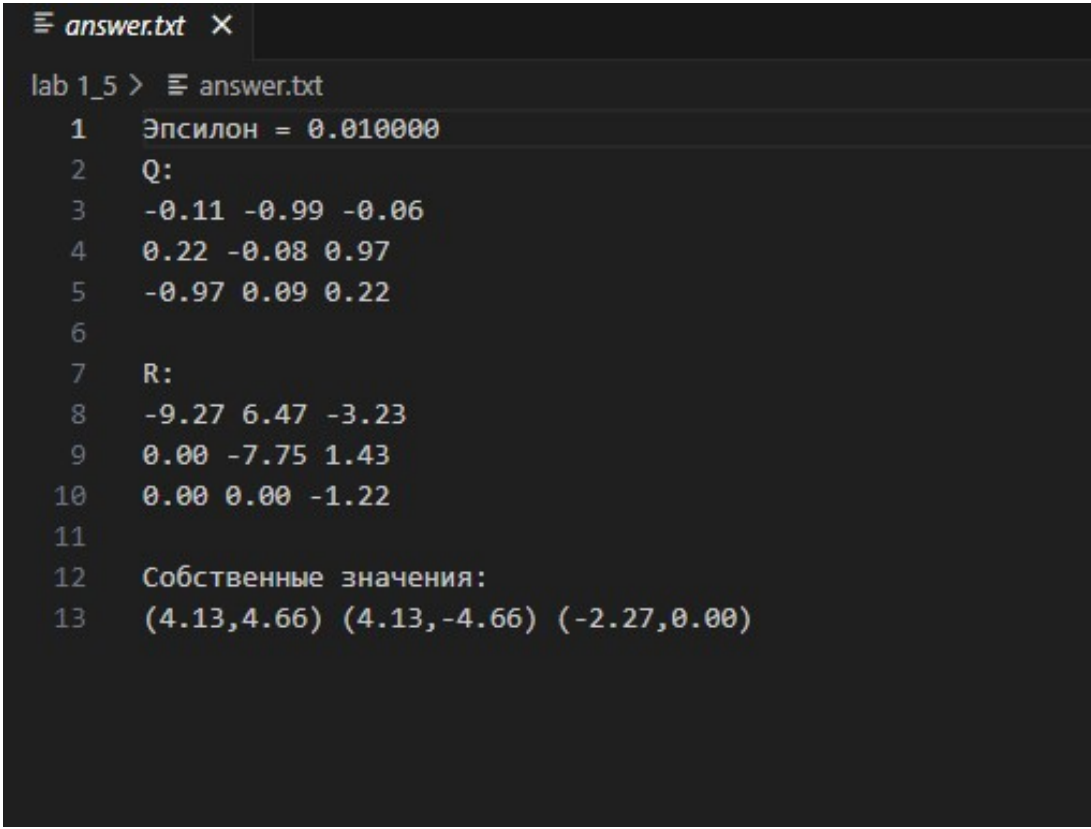
1 Постановка задачи

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: 15

$$\begin{pmatrix} 1 & 7 & -1 \\ -2 & 2 & -2 \\ 9 & -7 & 3 \end{pmatrix}$$

2 Результаты работы



```
lab 1_5 > ≡ answer.txt
1  Эпсилон = 0.010000
2  Q:
3  -0.11 -0.99 -0.06
4  0.22 -0.08 0.97
5  -0.97 0.09 0.22
6
7  R:
8  -9.27 6.47 -3.23
9  0.00 -7.75 1.43
10 0.00 0.00 -1.22
11
12 Собственные значения:
13 (4.13,4.66) (4.13,-4.66) (-2.27,0.00)
```

Рис. 5: Вывод программы в консоли

3 Исходный код

```
1  #include <ccomplex>
2  #include <cmath>
3  #include <fstream>
4  #include <iostream>
5  #include <vector>
6
7  using namespace std;
8
9  #define EPS 1e-5
10
11 int const n = 3;
12
13 class Matrix {
14 private:
15     int rows_, cols_;
16     vector<vector<double>> matrix_;
17     vector<int> swp_;
18
19     void SwapMatrix(Matrix &other) {
20         swap(rows_, other.rows_);
21         swap(cols_, other.cols_);
22         swap(matrix_, other.matrix_);
23     }
24
25 public:
26     Matrix(int rows, int cols) {
27         rows_ = rows;
28         cols_ = cols;
29         matrix_.resize(rows_);
30         for (int i = 0; i < rows_; ++i) {
31             matrix_[i].resize(cols_);
32         }
33     }
34     Matrix() : Matrix(1, 1) {}
35     Matrix(const Matrix &other) : Matrix(other.rows_, other.cols_) {
36         for (int i = 0; i < rows_; ++i) {
37             for (int j = 0; j < cols_; ++j) {
38                 matrix_[i][j] = other.matrix_[i][j];
39             }
40         }
41     }
42     Matrix(Matrix &&other) {
43         this->SwapMatrix(other);
44         other.rows_ = 0;
45         other.cols_ = 0;
46     }
47
48     int GetRows() const { return rows_; }
49     int GetCols() const { return cols_; }
50     const vector<int> &GetSwp() const { return swp_; }
51     void SumMatrix(const Matrix &other) {
52         if (rows_ != other.rows_ || cols_ != other.cols_)
53             throw runtime_error("Нельзя сложить матрицы разной размерности\n");
54         for (int i = 0; i < rows_; ++i) {
55             for (int j = 0; j < cols_; ++j) {
56                 matrix_[i][j] += other.matrix_[i][j];
57             }
58         }
59     }
60     void SubMatrix(const Matrix &other) {
```

```

61         if (rows_ != other.rows_ || cols_ != other.cols_)
62             throw runtime_error("Нельзя вычитать матрицы разной размерности\n");
63         for (int i = 0; i < rows_; ++i) {
64             for (int j = 0; j < cols_; ++j) {
65                 matrix_[i][j] -= other.matrix_[i][j];
66             }
67         }
68     }
69     void MulNumber(const double num) {
70         for (int i = 0; i < rows_; ++i) {
71             for (int j = 0; j < cols_; ++j) {
72                 matrix_[i][j] *= num;
73             }
74         }
75     }
76
77     Matrix MulMatrixReturn(const double num) {
78         Matrix tmp = *this;
79         tmp.MulNumber(num);
80         return tmp;
81     }
82
83     void MulMatrix(const Matrix &other) {
84         Matrix tmp(rows_, other.cols_);
85         for (int i = 0; i < rows_; ++i) {
86             for (int j = 0; j < other.cols_; ++j) {
87                 for (int k = 0; k < cols_; ++k)
88                     tmp.matrix_[i][j] += matrix_[i][k] * other.matrix_[k][j];
89             }
90         }
91         this->SwapMatrix(tmp);
92     }
93
94     Matrix MulMatrixReturn(const Matrix &other) {
95         Matrix tmp = *this;
96         tmp.MulMatrix(other);
97         return tmp;
98     }
99
100     Matrix Transpose() const {
101         Matrix result(cols_, rows_);
102         for (int i = 0; i < result.rows_; ++i) {
103             for (int j = 0; j < result.cols_; ++j) {
104                 result.matrix_[i][j] = matrix_[j][i];
105             }
106         }
107         return result;
108     }
109     int sign(double x) {
110         if (x > 0)
111             return 1;
112         if (x < 0)
113             return -1;
114         return 0;
115     }
116
117     pair<Matrix, Matrix> qr_decomposition() {
118         int n = this->rows_;
119         Matrix E(n, n);
120         for (int i = 0; i < n; ++i) {
121             E(i, i) = 1;
122         }

```

```

123     Matrix Q = E;
124     Matrix A = *this;
125     for (int i = 0; i < n - 1; ++i) {
126         Matrix H(n, n);
127         Matrix V(n, 1);
128         double norm = 0;
129         for (int j = i; j < n; ++j) {
130             norm += A(j, i) * A(j, i);
131         }
132         norm = sqrt(norm);
133         for (int j = i; j < V.GetRows(); ++j) {
134             if (j == i) {
135                 V(j, 0) = A(i, i) + sign(A(i, i)) * norm;
136             } else {
137                 V(j, 0) = A(j, i);
138             }
139         }
140         Matrix V_T = V.Transpose();
141         H = E - V.MulMatrixReturn(V_T).MulMatrixReturn(2 / (V_T.MulMatrixReturn(V))(0, 0));
142         A = H.MulMatrixReturn(A);
143         Q = Q.MulMatrixReturn(H);
144     }
145     return {Q, A};
146 }
147
148 vector<complex<double>> qr_method(double eps) {
149     int n = this->rows_;
150     Matrix A = *this;
151     vector<complex<double>> lambda;
152     vector<complex<double>> lambda_prev;
153     int counter = 0;
154     int iter = 50;
155     while (true) {
156         pair<Matrix, Matrix> QR = A.qr_decomposition();
157         Matrix Q = QR.first;
158         Matrix R = QR.second;
159         A = R.MulMatrixReturn(Q);
160         // cout << "A\n";
161         // A.ShowMatrix();
162         if (counter != iter) {
163             ++counter;
164             continue;
165         }
166         for (int i = 0; i < n; i += 1) {
167             double sum = 0;
168             for (int j = i + 1; j < n; ++j) {
169                 sum += abs(A(j, i));
170             }
171             if (sum < 0.001) {
172                 lambda.push_back(A(i, i));
173             } else {
174                 // (a_jj - Lambda)(a_j+1,j+1 - Lambda) = a_j,j+1 * a_j+1, j
175                 double a = 1;
176                 double b = (-1) * (A(i, i) + A(i + 1, i + 1));
177                 double c = A(i, i) * A(i + 1, i + 1) - A(i, i + 1) * A(i + 1, i);
178                 double d = b * b - 4 * c;
179                 complex<double> x1, x2;
180                 if (d < 0) {
181                     x1 = (-b + sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
182                     x2 = (-b - sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
183                 } else {
184                     x1 = (-b + sqrt(d)) / (2 * a);

```

```

185         x2 = (-b - sqrt(d)) / (2 * a);
186     }
187     lambda.push_back(x1);
188     lambda.push_back(x2);
189     ++i;
190 }
191 }
192 bool exit = true;
193 // исключаем первую итерацию
194 if (lambda_prev.size() != 0) {
195     for (int i = 0; i < lambda.size(); i++) {
196         if (abs(lambda[i] - lambda_prev[i]) > eps) {
197             exit = false;
198             break;
199         }
200     }
201     if (exit == true)
202         break;
203 }
204 lambda_prev = lambda;
205 lambda.clear();
206 counter = 0;
207 }
208 return lambda;
209 }
210
211 Matrix operator+(const Matrix &other) {
212     Matrix result(*this);
213     result.SumMatrix(other);
214     return result;
215 }
216 Matrix operator-(const Matrix &other) {
217     Matrix result(*this);
218     result.SubMatrix(other);
219     return result;
220 }
221 Matrix operator=(const Matrix &other) {
222     if (this != &other) { // b = b
223         Matrix tmp(other);
224         this->SwapMatrix(tmp);
225     }
226     return *this;
227 }
228 double &operator()(int i, int j) {
229     if (i < 0 || i >= rows_ || j < 0 || j >= cols_)
230         throw runtime_error("Индекс за пределами матрицы\n");
231     return matrix_[i][j];
232 }
233 vector<double> &operator()(int row) { return matrix_[row]; }
234 };
235
236 ostream &operator<<(ostream &stream, Matrix A) {
237     for (int i = 0; i < A.GetRows(); i++) {
238         for (int j = 0; j < A.GetCols(); j++)
239             stream << A(i, j) << ' ';
240         stream << '\n';
241     }
242     return stream;
243 }
244
245 istream &operator>>(istream &stream, Matrix &A) {
246     for (int i = 0; i < A.GetRows(); i++) {

```

```

247         for (int j = 0; j < A.GetCols(); j++)
248             stream >> A(i, j);
249     }
250     return stream;
251 }
252 int main()
253 {
254     Matrix A(n, n), self_vec(n,n), self_value(n,1);
255     double eps;
256     ofstream fout("answer.txt");
257     fout.precision(2);
258     fout << fixed;
259     ifstream fin("input.txt");
260     fin >> eps;
261     fin >> A;
262     fout << "Эпсилон = " << to_string(eps) << endl;
263     pair<Matrix, Matrix> QR = A.qr_decomposition();
264     fout << "Q:\n";
265     fout << QR.first;
266     fout << "\nR:\n";
267     fout << QR.second;
268
269     vector<complex<double>> labmda = A.qr_method(eps);
270     fout << "\Собственные значения:\n";
271     for (int i = 0; i < labmda.size(); ++i) {
272         fout << labmda[i] << " ";
273     }
274 }

```