

**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: Наумов Г.К.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

1.1 LU - разложение матриц

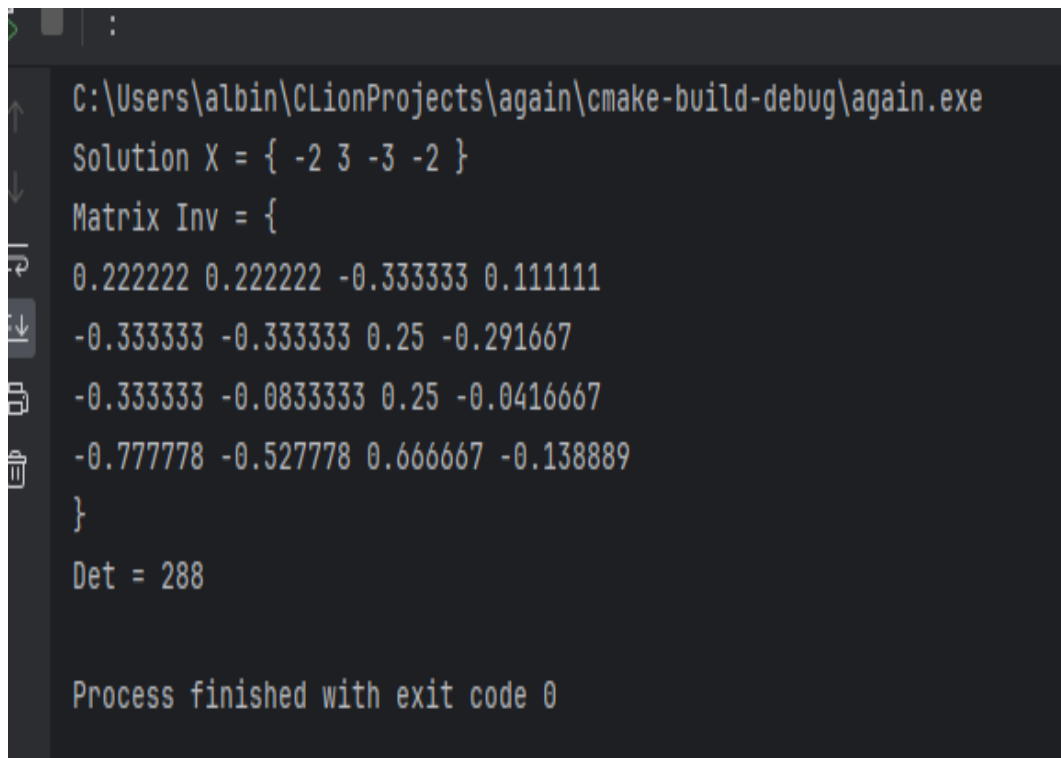
1 Постановка задачи

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: 16

$$\begin{cases} -5x_1 - x_2 - 3x_3 - x_4 = 18 \\ -2x_1 + 9x_3 - 4x_4 = -12 \\ -7x_1 - 2x_2 + 2x_3 - 2x_4 = 6 \\ 2x_1 - 4x_2 - 4x_3 + 4x_4 = -12 \end{cases}$$

2 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Solution X = { -2 3 -3 -2 }
Matrix Inv = {
0.222222 0.222222 -0.333333 0.111111
-0.333333 -0.333333 0.25 -0.291667
-0.333333 -0.0833333 0.25 -0.0416667
-0.777778 -0.527778 0.666667 -0.138889
}
Det = 288

Process finished with exit code 0
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <tuple>
4
5  std::tuple<std::vector<std::vector<double>>, std::vector<std::vector<double>>, std::
   vector<std::vector<double>>> lu_decomposition_with_pivoting(const std::vector<std
   ::vector<double>>& matrix, int n) {
6      std::vector<std::vector<double>> L(n, std::vector<double>(n, 0.0));
7      std::vector<std::vector<double>> U(n, std::vector<double>(n, 0.0));
8      std::vector<std::vector<double>> P(n, std::vector<double>(n, 0.0));
9
10     for (int i = 0; i < n; i++)
11         P[i][i] = 1.0;
12
13     for (int i = 0; i < n; i++) {
14         double max_val = 0.0;
15         for (int j = 0; j < n; j++)
16             max_val = std::max(max_val, std::abs(matrix[i][j]));
17
18         if (max_val == 0.0)
19             return std::make_tuple(std::vector<std::vector<double>>(), std::vector<std
   ::vector<double>>(), std::vector<std::vector<double>>()); //
20
21         for (int j = 0; j < n; j++) {
22             if (i <= j) {
23                 U[i][j] = matrix[i][j];
24                 for (int k = 0; k < i; ++k)
25                     U[i][j] -= L[i][k] * U[k][j];
26             }
27             if (i > j) {
28                 L[i][j] = matrix[i][j];
29                 for (int k = 0; k < j; ++k)
30                     L[i][j] -= L[i][k] * U[k][j];
31                 L[i][j] /= U[j][j];
32             }
33         }
34     }
35
36     return std::make_tuple(L, U, P);
37 }
38
39 std::vector<double> solve_lu_decomposition(const std::vector<std::vector<double>>& L,
40                                           const std::vector<std::vector<double>>& U,
41                                           const std::vector<std::vector<double>>& P,
42                                           const std::vector<double>& b, int n) {
43     std::vector<double> y(n, 0.0);
44     std::vector<double> x(n, 0.0);
```

```

45
46 // Ly = Pb
47 for (int i = 0; i < n; i++) {
48     y[i] = 0.0;
49     for (int j = 0; j < n; j++)
50         y[i] += P[i][j] * b[j];
51     for (int j = 0; j < i; j++)
52         y[i] -= L[i][j] * y[j];
53 }
54
55 // Ux = y
56 for (int i = n - 1; i >= 0; --i) {
57     x[i] = y[i];
58     for (int j = i + 1; j < n; j++)
59         x[i] -= U[i][j] * x[j];
60     x[i] /= U[i][i];
61 }
62
63 return x;
64 }
65
66 double determinant_from_lu(const std::vector<std::vector<double>>& U, const std:::
    vector<std::vector<double>>& P, int n) {
67     double det_U = 1.0;
68     for (int i = 0; i < n; i++)
69         det_U *= U[i][i];
70     double det_P = 1.0;
71     for (int i = 0; i < n; i++)
72         det_P *= P[i][i];
73     return det_U * det_P;
74 }
75
76 std::vector<std::vector<double>> inverse_from_lu(const std::vector<std::vector<double
    >>& L,
77
78                                     const std::vector<std::vector<double>>& U,
79                                     const std::vector<std::vector<double>>& P,
80                                     int n) {
81     std::vector<std::vector<double>> inv(n, std::vector<double>(n, 0.0));
82     for (int i = 0; i < n; i++) {
83         std::vector<double> b(n, 0.0);
84         b[i] = 1.0;
85         std::vector<double> x = solve_lu_decomposition(L, U, P, b, n);
86         for (int j = 0; j < n; j++)
87             inv[j][i] = x[j];
88     }
89     return inv;
90 }
91 void print_results(std::vector<double> x, double det, std::vector<std::vector<double>>

```

```

    inv, int n) {
92
93     std::cout << "Solution X = { ";
94     for (int i = 0; i < n; i++) {
95         std::cout << x[i] << " ";
96     }
97     std::cout << "}" << std::endl;
98
99     std::cout << "Matrix Inv = {" << std::endl;
100    for (int i = 0; i < n; i++) {
101        for (int j = 0; j < n; j++) {
102            std::cout << inv[i][j] << " ";
103        }
104        std::cout << std::endl;
105    }
106    std::cout << "}" << std::endl;
107
108    std::cout << "Det = " << det << std::endl;
109 }
110
111 int main() {
112     std::vector<std::vector<double>> A = {{-5.0, -1.0, -3.0, -1.0},
113                                           {-2.0, 0.0, 8.0, -4.0},
114                                           {-7.0, -2.0, 2.0, -2.0},
115                                           {2.0, -4.0, -4.0, 4.0}};
116     std::vector<double> b = {18.0, -12.0, 6.0, -12.0};
117
118     int n = 4;
119
120     std::vector<std::vector<double>> L, U, P;
121     std::tie(L, U, P) = lu_decomposition_with_pivoting(A, n);
122     if (L.empty()) {
123         std::cout << "Solution does not exist." << std::endl;
124         return 1;
125     }
126
127     std::vector<double> x = solve_lu_decomposition(L, U, P, b, n);
128     double det = determinant_from_lu(U, P, n);
129     std::vector<std::vector<double>> inv = inverse_from_lu(L, U, P, n);
130     print_results(x, det, inv, n);
131     return 0;
132 }

```

1.2 Метод прогонки

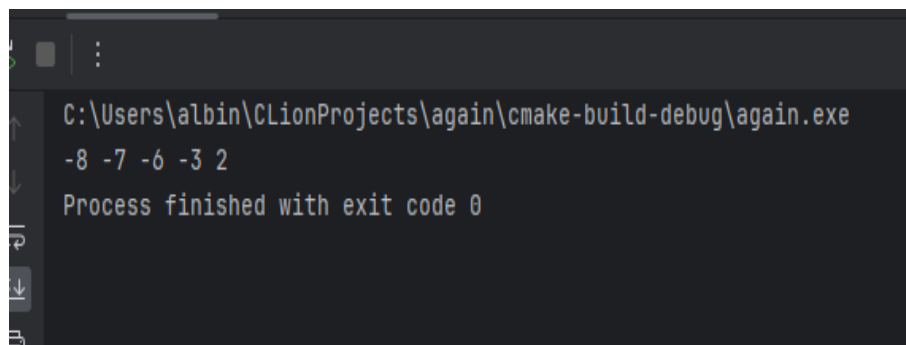
4 Постановка задачи

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: 16

$$\begin{cases} 18x_1 - 9x_2 = -81 \\ 2x_1 - 9x_2 - 4x_3 = 71 \\ -9x_2 + 21x_3 - 8x_4 = -39 \\ -4x_3 - 10x_4 + 5x_5 = 64 \\ 7x_4 + 12x_5 = 3 \end{cases}$$

5 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
-8 -7 -6 -3 2
Process finished with exit code 0
```

Рис. 2: Вывод программы

6 Исходный код

```
1 #include <iostream>
2 #include <vector>
3
4 std::vector<double> run_through_method(std::vector<std::vector<double>> matrix, std::
    vector<double> b, int n) {
5     std::vector<double> P(n, 0.0);
6     std::vector<double> Q(n, 0.0);
7
8     P[0] = -matrix[0][1] / matrix[0][0];
9     Q[0] = b[0] / matrix[0][0];
10
11     for (int i = 1; i < n - 1; ++i) {
12         double denominator = matrix[i][i] + matrix[i][i - 1] * P[i - 1];
13         P[i] = -matrix[i][i + 1] / denominator;
14         Q[i] = (b[i] - matrix[i][i - 1] * Q[i - 1]) / denominator;
15     }
16
17     std::vector<double> x(n, 0.0);
18     double denominator = matrix[n - 1][n - 1] + matrix[n - 1][n - 2] * P[n - 2];
19     Q[n - 1] = (b[n - 1] - matrix[n - 1][n - 2] * Q[n - 2]) / denominator;
20     x[n - 1] = Q[n - 1];
21
22     for (int i = n - 2; i >= 0; --i) {
23         x[i] = P[i] * x[i + 1] + Q[i];
24     }
25
26     return x;
27 }
28
29 void print_result(std::vector<double> res, int n) {
30     for (int i = 0; i < n; i++) {
31         std::cout << res[i] << " ";
32     }
33 }
34
35 int main() {
36     std::vector<std::vector<double>> matrix = {{18.0, -9.0, 0.0, 0.0, 0.0},
37                                                {2.0, -9.0, -4.0, 0.0, 0.0},
38                                                {0.0, -9.0, 21.0, -8.0, 0.0},
39                                                {0.0, 0.0, -4.0, -10.0, 5.0},
40                                                {0.0, 0.0, 0.0, 7.0, 12.0}};
41     std::vector<double> b = {-81.0, 71.0, -39.0, 64.0, 3.0};
42
43     int n = 5;
44     std::vector<double> result = run_through_method(matrix, b, n);
45     print_result(result, n);
46     return 0;
```


1.3 Метод простых итераций. Метод Зейделя

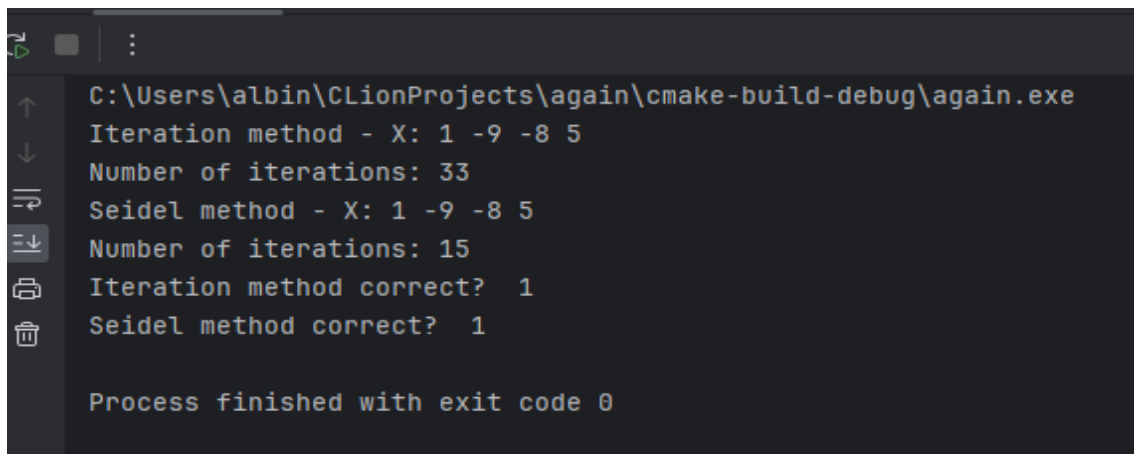
7 Постановка задачи

Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

Вариант: 16

$$\begin{cases} 21x_1 - 6x_2 - 9x_3 - 4x_4 = 127 \\ -6x_1 + 20x_2 - 4x_3 + 2x_4 = -144 \\ -2x_1 - 7x_2 - 20x_3 + 3x_4 = 236 \\ 4x_1 + 9x_2 + 6x_3 + 24x_4 = -5 \end{cases}$$

8 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Iteration method - X: 1 -9 -8 5
Number of iterations: 33
Seidel method - X: 1 -9 -8 5
Number of iterations: 15
Iteration method correct? 1
Seidel method correct? 1

Process finished with exit code 0
```

Рис. 3: Вывод программы

9 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 | //using namespace std;
6 |
```

```

7 | std::vector<double> solve_simple_iteration(std::vector<std::vector<double>>& matrix,
  | std::vector<double>& vector_b, double precision, int &iterations, int
  | max_iterations=1000) {
8 |     int n = matrix.size();
9 |     std::vector<double> x(n, 0.0);
10 |    while (true) {
11 |        iterations++;
12 |        std::vector<double> x_new(n, 0.0);
13 |        for (int i = 0; i < n; i++) {
14 |            double sum = 0.0;
15 |            for (int j = 0; j < n; j++) {
16 |                if (j != i) {
17 |                    sum += matrix[i][j] * x[j];
18 |                }
19 |            }
20 |            x_new[i] = (vector_b[i] - sum) / matrix[i][i];
21 |        }
22 |        double max_diff = 0.0;
23 |        for (int i = 0; i < n; i++) {
24 |            max_diff = std::max(max_diff, std::fabs(x[i] - x_new[i]));
25 |        }
26 |        if (max_diff < precision) {
27 |            break;
28 |        }
29 |        x = x_new;
30 |        if (iterations > max_iterations) {
31 |            std::cout << "Warning: The method of simple iterations did not converge."
  | << std::endl;
32 |            break;
33 |        }
34 |    }
35 |    return x;
36 | }
37 |
38 | std::vector<double> SolveUsingGaussSeidel(std::vector<std::vector<double>>& matrix,
  | std::vector<double>& vector_b, double precision, int& iterations, int
  | max_iterations = 1000) {
39 |     int n = matrix.size();
40 |     std::vector<double> x(n, 0.0);
41 |     std::vector<double> x_new(n, 0.0);
42 |     while (true) {
43 |         iterations++;
44 |         for (int i = 0; i < n; i++) {
45 |             double sum1 = 0.0;
46 |             double sum2 = 0.0;
47 |             for (int j = 0; j < i; j++) {
48 |                 sum1 += matrix[i][j] * x_new[j];
49 |             }
50 |             for (int j = i + 1; j < n; j++) {

```

```

51         sum2 += matrix[i][j] * x[j];
52     }
53     x_new[i] = (vector_b[i] - sum1 - sum2) / matrix[i][i];
54 }
55 double max_diff = 0.0;
56 for (int i = 0; i < n; i++) {
57     max_diff = std::max(max_diff, std::fabs(x[i] - x_new[i]));
58 }
59 if (max_diff < precision) {
60     break;
61 }
62 x = x_new;
63 if (iterations > max_iterations) {
64     break;
65 }
66 }
67 return x;
68 }
69
70 bool check(std::vector<std::vector<double>> a, std::vector<double> x, std::vector<
71     double> b) {
72     double eps = 0.00001;
73     for (int i = 0; i < a.size(); i++) {
74         double row = 0.0;
75         for (int j = 0; j < a[i].size(); j++){
76             row += a[i][j] * x[j];
77         }
78         if (std::fabs(row - b[i]) > eps) {
79             return false;
80         }
81     }
82     return true;
83 }
84
85 int main() {
86     std::vector<std::vector<double>> A = {{21, -6, -9, -4}, {-6, 20, -4, 2}, {-2, -7,
87         -20, 3}, {4, 9, 6, 24}};
88     std::vector<double> b = {127, -144, 236, -5};
89     double eps = 0.000000001;
90     int iteration_first = 0;
91     int iteration_second = 0;
92
93     std::vector<double> x_iter = solve_simple_iteration(A, b, eps, iteration_first);
94     std::cout << "Iteration method - X: ";
95     for (double xi : x_iter) {
96         std::cout << xi << " ";
97     }
98     std::cout << std::endl;
99     std::cout << "Number of iterations: " << iteration_first << std::endl;

```

```

98 |
99 |     std::vector<double> x_seidel = SolveUsingGaussSeidel(A, b, eps, iteration_second);
100 |     std::cout << "Seidel method - X: ";
101 |     for (double xi : x_seidel) {
102 |         std::cout << xi << " ";
103 |     }
104 |     std::cout << std::endl;
105 |     std::cout << "Number of iterations: " << iteration_second << std::endl;
106 |
107 |     std::cout << "Iteration method correct? - " << check(A, x_iter, b) << std::endl;
108 |     std::cout << "Seidel method correct? - " << check(A, x_seidel, b) << std::endl;
109 |     return 0;
110 | }

```

1.4 Метод вращений

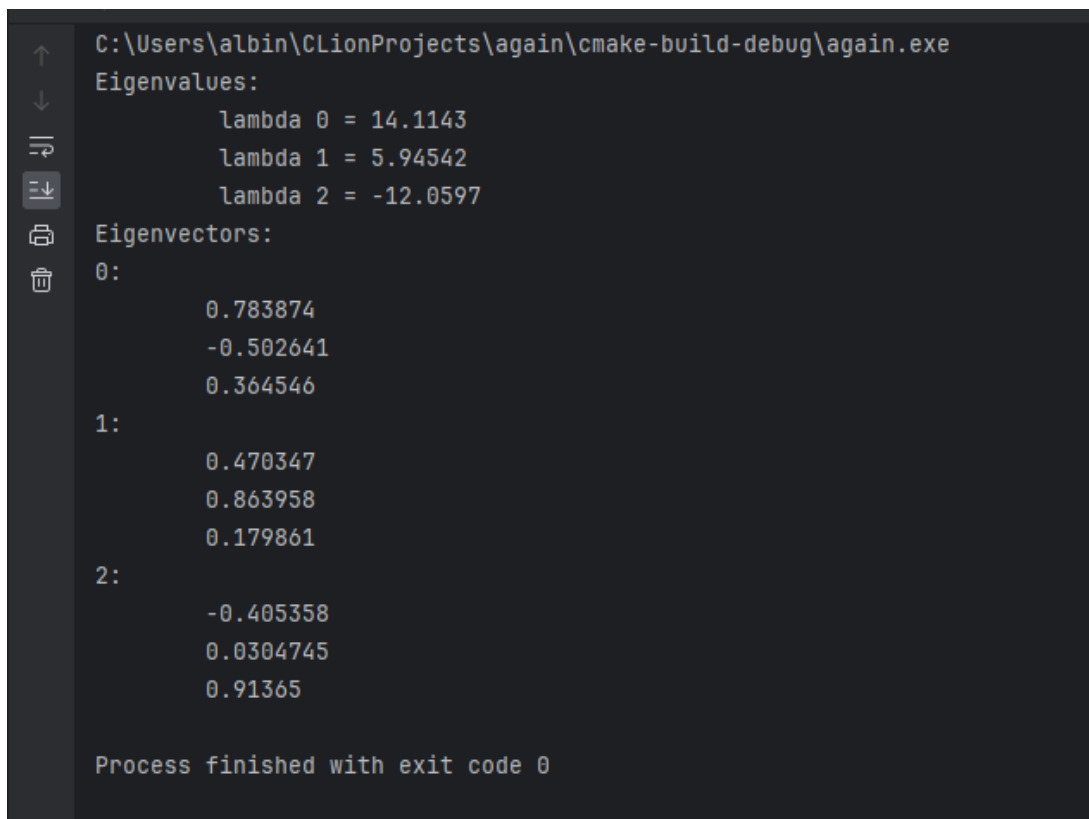
10 Постановка задачи

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: 16

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

11 Результаты работы



```
C:\Users\albin\CLionProjects\again\cmake-build-debug\again.exe
Eigenvalues:
    lambda 0 = 14.1143
    lambda 1 = 5.94542
    lambda 2 = -12.0597
Eigenvectors:
0:
    0.783874
    -0.502641
    0.364546
1:
    0.470347
    0.863958
    0.179861
2:
    -0.405358
    0.0304745
    0.91365
Process finished with exit code 0
```

Рис. 4: Вывод программы

12 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5  constexpr double EPS = 0.01;
6
7
8  double rmsNonDiagonal(const std::vector<std::vector<double>>& matrix) {
9      double sum = 0.0;
10     int n = matrix.size();
11     for (int i = 0; i < n; ++i)
12         for (int j = i + 1; j < n; ++j)
13             sum += matrix[i][j] * matrix[i][j];
14     return std::sqrt(sum);
15 }
16
17 std::pair<int, int> maxUpDiagonalElement(const std::vector<std::vector<double>>& A) {
18     double maxElement = -std::numeric_limits<double>::max();
19     std::pair<int, int> maxIndex = std::make_pair(0, 0);
20     int size = A.size();
21     for (int i = 0; i < size - 1; ++i)
22         for (int j = i + 1; j < size; ++j)
23             if (std::abs(A[i][j]) > maxElement) {
24                 maxElement = std::abs(A[i][j]);
25                 maxIndex = std::make_pair(i, j);
26             }
27     return maxIndex;
28 }
29
30 double Get_Phi(int max_i, int max_j, const std::vector<std::vector<double>>& A) {
31     if(A[max_i][max_i] == A[max_j][max_j])
32         return M_PI / 4;
33     else
34         return 0.5 * std::atan(2 * A[max_i][max_j] / (A[max_i][max_i] - A[max_j][max_j]));
35 }
36
37 std::vector<std::vector<double>> Transpose_Matrix(const std::vector<std::vector<double>>& A) {
38     int rows = A.size();
39     int cols = A[0].size();
40     std::vector<std::vector<double>> result(cols, std::vector<double>(rows, 0));
41     for (int i = 0; i < rows; ++i)
42         for (int j = 0; j < cols; ++j)
43             result[j][i] = A[i][j];
44     return result;
45 }
```

```

46
47 std::vector<std::vector<double>> Matrix_Multiplication(const std::vector<std::vector<
    double>>& A, const std::vector<std::vector<double>>& B) {
48     int n = A.size();
49     int m = B[0].size();
50     int p = B.size();
51     std::vector<std::vector<double>> C(n, std::vector<double>(m, 0.0));
52     for (int i = 0; i < n; ++i)
53         for (int j = 0; j < m; ++j)
54             for (int k = 0; k < p; ++k)
55                 C[i][j] += A[i][k] * B[k][j];
56     return C;
57 }
58
59 std::vector<std::vector<double>> Initialize_U(const std::vector<std::vector<double>>&
    A) {
60     int im, jm;
61     std::tie(im, jm) = maxUpDiagonalElement(A);
62     double phi = Get_Phi(im, jm, A);
63
64     int size = A.size();
65     std::vector<std::vector<double>> U(size, std::vector<double>(size, 0));
66     for (int i = 0; i < size; ++i)
67         U[i][i] = 1;
68     U[im][jm] = -std::sin(phi);
69     U[jm][im] = std::sin(phi);
70     U[im][im] = U[jm][jm] = std::cos(phi);
71     return U;
72 }
73
74 void Jacobi_Eigenvalue(std::vector<std::vector<double>> A) {
75     std::vector<std::vector<double>> V(A.size(), std::vector<double>(A.size(), 0));
76     for(int i = 0; i < A.size(); ++i)
77         V[i][i] = 1;
78
79     while(rmsNonDiagonal(A) > EPS) {
80         std::vector<std::vector<double>> U = Initialize_U(A);
81         std::vector<std::vector<double>> U_t = Transpose_Matrix(U);
82         A = Matrix_Multiplication(Matrix_Multiplication(U_t, A), U);
83         V = Matrix_Multiplication(V, U);
84     }
85
86     std::vector<double> lambda(A.size());
87     for(int i = 0; i < A.size(); ++i)
88         lambda[i] = A[i][i];
89
90     std::cout << "Eigenvalues:" << std::endl;
91     for(int i = 0; i < lambda.size(); ++i)
92         std::cout << "\t lambda " << i << " = " << lambda[i] << std::endl;

```

```

93     std::cout << "Eigenvectors:" << std::endl;
94     for(int j = 0; j < V.size(); ++j){
95         std::cout << j << ":" << std::endl;
96         for(int i = 0; i < V.size(); ++i)
97             std::cout << "\t" << V[i][j] << std::endl;
98     }
99 }
100
101 int main() {
102     std::vector<std::vector<double>> A = {
103         {8, -3, 9},
104         {-3, 8, -2},
105         {9, -2, -8}
106     };
107     Jacobi_Eigenvalue(A);
108     return 0;
109 }

```


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: 16

$$\begin{pmatrix} 1 & 2 & 5 \\ -8 & 0 & -6 \\ 7 & -9 & -7 \end{pmatrix}$$

14 Результаты работы

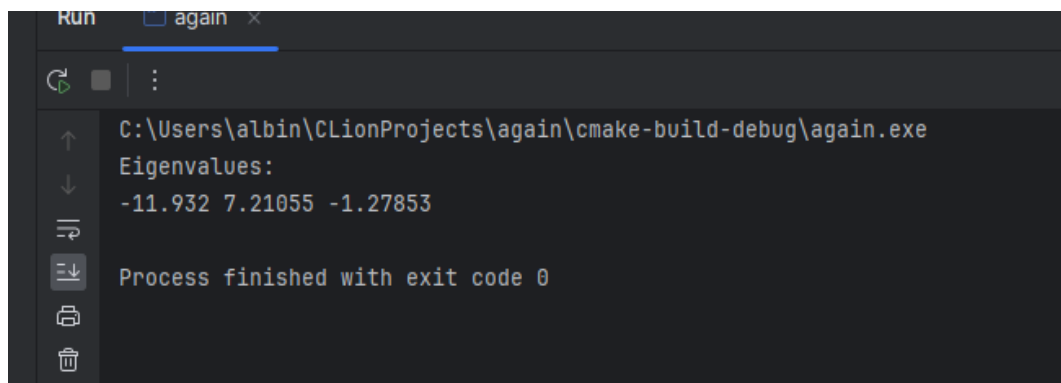


Рис. 5: Вывод программы

15 Исходный код

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4
5 void qr_decomposition(const std::vector<std::vector<double>>& A, std::vector<std::
    vector<double>>& Q, std::vector<std::vector<double>>& R, double eps) {
6     int n = A.size();
7     int m = A[0].size();
8 }
```

```

9      Q = A;
10     R = std::vector<std::vector<double>>(m, std::vector<double>(m, 0.0));
11
12     for (int j = 0; j < m; ++j) {
13         // Compute the j-th column of Q and R
14         for (int k = 0; k < j; ++k) {
15             double dot_product = 0.0;
16             for (int i = 0; i < n; ++i) {
17                 dot_product += Q[i][j] * Q[i][k];
18             }
19             for (int i = 0; i < n; ++i) {
20                 Q[i][j] -= dot_product * Q[i][k];
21             }
22         }
23
24         double norm = 0.0;
25         for (int i = 0; i < n; ++i) {
26             norm += Q[i][j] * Q[i][j];
27         }
28         norm = sqrt(norm);
29
30         for (int i = 0; i < n; ++i) {
31             Q[i][j] /= norm;
32             R[j][j] = norm;
33         }
34
35         for (int k = j + 1; k < m; ++k) {
36             double dot_product = 0.0;
37             for (int i = 0; i < n; ++i) {
38                 dot_product += Q[i][j] * A[i][k];
39             }
40             R[j][k] = dot_product;
41         }
42     }
43 }
44
45 std::vector<std::vector<double>> matrix_multiply(const std::vector<std::vector<double
46 >>& A, const std::vector<std::vector<double>>& B) {
47     int n = A.size();
48     int m = B[0].size();
49     int p = B.size();
50
51     std::vector<std::vector<double>> result(n, std::vector<double>(m, 0.0));
52
53     for (int i = 0; i < n; ++i) {
54         for (int j = 0; j < m; ++j) {
55             for (int k = 0; k < p; ++k) {
56                 result[i][j] += A[i][k] * B[k][j];

```

```

57     }
58 }
59
60     return result;
61 }
62
63 std::vector<double> compute_eigenvalues(const std::vector<std::vector<double>>& A, int
        iterations, double eps) {
64     int n = A.size();
65
66     std::vector<std::vector<double>> Ak = A;
67
68     for (int iter = 0; iter < iterations; ++iter) {
69         std::vector<std::vector<double>> Q, R;
70         qr_decomposition(Ak, Q, R, eps);
71         Ak = matrix_multiply(R, Q);
72     }
73
74     std::vector<double> eigenvalues(n);
75
76     for (int i = 0; i < n; ++i) {
77         eigenvalues[i] = Ak[i][i];
78     }
79
80     return eigenvalues;
81 }
82
83
84 int main() {
85     std::vector<std::vector<double>> A = {{1, 2, 5}, {-8, 0, -6}, {7, -9, -7}};
86
87     double epsilon = 1e-6;
88
89     std::vector<std::vector<double>> Q;
90     std::vector<std::vector<double>> R;
91
92     qr_decomposition(A, Q, R, epsilon);
93
94     std::vector<double> eigenvalues = compute_eigenvalues(A, 50, epsilon);
95
96     std::cout << "Eigenvalues:" << std::endl;
97     for (double eigenvalue : eigenvalues) {
98         std::cout << eigenvalue << " ";
99     }
100     std::cout << std::endl;
101
102     return 0;
103 }

```