

**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: Ерофеева Е.С.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

1.1 LU - разложение матриц

1 Постановка задачи

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

Вариант: 6

$$\begin{cases} x_1 + 2x_2 - x_3 - 7x_4 = -23 \\ 8x_1 - 9x_3 - 3x_4 = 39 \\ 2x_1 - 3x_2 + 7x_3 + x_4 = -7 \\ x_1 - 5x_2 - 6x_3 + 8x_4 = 30 \end{cases}$$

2 Результаты работы

```
Матрица L:
1      0      0      0
8      1      0      0
2      0.44    1      0
1      0.44   -0.48    1

Матрица U:
1      2      -1      -7
0     -16     -1      53
0      0      9.44    -8.19
0      0      0     -12.15

Решение системы:
6      6     -1      6

Детерминант: 1834

Обратная матрица:
-0.18  0.15  0.07  -0.11
-0.38  0.11 -0.14  -0.27
-0.07  0     0.07  -0.07
-0.27  0.05 -0.04  -0.08
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1  #include<iostream>
2  #include<fstream>
3  #include<vector>
4  #include<cmath>
5
6  using Matrix = std::vector<std::vector<double>>>;
7
8
9  class LUMatrix{
10 public:
11     std::vector<double> x;
12     Matrix L;
13     Matrix U;
14     Matrix inv;
15     LUMatrix(std::string filename);
16     void print_matrix(Matrix &matrix);
17     void print_vector(std::vector<double> &b);
18     double get_determinant(Matrix &matrix);
19 private:
20     int n;
21     Matrix data;
22     std::vector<double> b;
23     void decompose();
24     std::vector<double> solve_system(std::vector<double> &b);
25     void transpose(Matrix &matrix);
26     void get_inv();
27 };
28
29
30 LUMatrix::LUMatrix(std::string filename){
31     std::ifstream idescr(filename);
32     //
33     double d; int cnt = 0;
34     while(idescr >> d){
35         cnt++;
36     }
37     n = (-1 + sqrt(1 + 4*cnt)) / 2;
38     idescr.close();
39     idescr.open(filename);
40     //
41     data.resize(n, std::vector<double>(n));
42     for(int i = 0; i < n; i++){
43         for(int j = 0; j < n; j++){
44             idescr >> data[i][j];
45         }
46     }
47     b.resize(n);
```

```

48         for(int i = 0; i < n; i++){
49             idescr >> b[i];
50         }
51         idescr.close();
52         // LU
53         L.resize(n, std::vector<double>(n));
54         U.resize(n, std::vector<double>(n));
55         U = data;
56         decompose();
57         //
58         x = solve_system(b);
59         //
60         get_inv();
61     }
62
63
64     void LUMatrix::print_matrix(Matrix &matrix){
65         for(int i = 0; i < n; i++){
66             for(int j = 0; j < n; j++){
67                 std::cout << std::round(matrix[i][j]*100)/100.0 << '\t';
68             }
69             std::cout << std::endl;
70         }
71         std::cout << std::endl;
72     }
73
74
75     void LUMatrix::print_vector(std::vector<double> &b){
76         for(int i = 0; i < n; i++){
77             std::cout << std::round(b[i]*100)/100.0 << '\t';
78         }
79         std::cout << std::endl << std::endl;
80     }
81
82
83     void LUMatrix::decompose(){
84         for(int k = 0; k < n; k++){
85             L[k][k] = 1;
86             for(int i = k + 1; i < n; i++){
87                 double mu = U[i][k]/U[k][k];
88                 L[i][k] = mu;
89                 for(int j = k; j < n; j++){
90                     U[i][j] -= mu*U[k][j];
91                 }
92             }
93         }
94     }
95
96

```

```

97 | std::vector<double> LUMatrix::solve_system(std::vector<double> &rs){
98 |     // z Lz = b;
99 |     std::vector<double> z(n, 0);
100 |     for(int i = 0; i < n; i++){
101 |         z[i] = rs[i];
102 |         for(int j = 0; j < i; j++){
103 |             z[i] -= L[i][j]*z[j];
104 |         }
105 |     }
106 |     // x Ux = z;
107 |     std::vector<double> x(n,0);
108 |     for(int i = n - 1; i >= 0; i--){
109 |         x[i] = z[i];
110 |         for(int j = i + 1; j < n; j++){
111 |             x[i] -= U[i][j]*x[j];
112 |         }
113 |         x[i] /= U[i][i];
114 |     }
115 |     return x;
116 | }
117 |
118 |
119 | void LUMatrix::transpose(Matrix &matrix){
120 |     Matrix res(n, std::vector<double>(n, 0));
121 |     for(int i = 0; i < n; i++){
122 |         for(int j = 0; j < n; j++){
123 |             res[i][j] = matrix[j][i];
124 |         }
125 |     }
126 |     matrix = res;
127 | }
128 |
129 |
130 | void LUMatrix::get_inv(){
131 |     Matrix E(n, std::vector<double>(n));
132 |     for(int i = 0; i < n; i++){
133 |         E[i][i] = 1;
134 |     }
135 |     std::vector<double> res;
136 |     for(std::vector<double> e: E){
137 |         inv.push_back(solve_system(e));
138 |     }
139 |     transpose(inv);
140 | }
141 |
142 |
143 | double LUMatrix::get_determinant(Matrix &matrix){
144 |     double provide = matrix[0][0];
145 |     for(int i = 1; i < n; i++){

```

```

146         provide *= matrix[i][i];
147     }
148     return provide;
149 }
150
151
152 int main(int argc, char** argv){
153     LUMatrix matrix = LUMatrix(argv[1]);
154
155     // L U
156     std::cout << std::endl;
157     std::cout << "\t L:" << std::endl;
158     matrix.print_matrix(matrix.L);
159     std::cout << "\t U:" << std::endl;
160     matrix.print_matrix(matrix.U);
161
162     //
163     std::cout << "\t :" << std::endl;
164     matrix.print_vector(matrix.x);
165
166     //
167     std::cout << ": ";
168     std::cout << matrix.get_determinant(matrix.U) << std::endl;
169     std::cout << std::endl;
170
171     //
172     std::cout << "\t :" << std::endl;
173     matrix.print_matrix(matrix.inv);
174     return 0;
175 }

```

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: 6

$$\begin{cases} 6x_1 - 5x_2 = -58 \\ -6x_1 + 16x_2 + 9x_3 = 161 \\ 9x_2 - 17x_3 - 3x_4 = -114 \\ 8x_3 + 22x_4 - 8x_5 = -90 \\ 6x_4 - 13x_5 = -55 \end{cases}$$

5 Результаты работы

Решение системы				
-8	2	9	-7	1

Рис. 2: Вывод программы в консоли

6 Исходный код

```
1  #include<iostream>
2  #include<fstream>
3  #include<vector>
4  #include<cmath>
5
6
7  int input_coeff( std::vector<double> &a,
8                  std::vector<double> &b,
9                  std::vector<double> &c,
10                 std::vector<double> &d,
11                 std::string filename ) {
12     std::ifstream idescr(filename);
13     int n;
14     idescr >> n;
15     a.resize(n);
16     for(int i = 1; i < n; i++){
17         idescr >> a[i];
18     }
19     b.resize(n);
20     for(int i = 0; i < n; i++){
21         idescr >> b[i];
22     }
23     c.resize(n);
24     for(int i = 0; i < n - 1; i++){
25         idescr >> c[i];
26     }
27     d.resize(n);
28     for(int i = 0; i < n; i++){
29         idescr >> d[i];
30     }
31     idescr.close();
32     return n;
33 }
34
35
36 int main(int argc, char **argv){
37     std::vector<double> a;
38     std::vector<double> b;
39     std::vector<double> c;
40     std::vector<double> d;
41     int n = input_coeff(a, b, c, d, argv[1]);
42
43     //
44     std::vector<double> p(n);
45     std::vector<double> q(n);
46     p[0] = -c[0] / b[0];
47     q[0] = d[0] / b[0];
```



```

48     for(int i = 1; i < n; i++){
49         p[i] = -c[i] / (b[i] + a[i] * p[i-1]);
50         q[i] = (d[i] - a[i] * q[i-1]) / (b[i] + a[i] * p[i-1]);
51     }
52     //
53     std::vector<double> x(n);
54     x[n-1] = q[n-1];
55     for(int i = n - 2; i >= 0; i--){
56         x[i] = p[i] * x[i+1] + q[i];
57     }
58
59     //
60     std::cout << "\t " << std::endl;
61     for(int i = 0; i < n; i++){
62         std::cout << std::round(x[i]*100)/100.0 << '\t';
63     }
64     std::cout << std::endl;
65     return 0;
66 }

```

1.3 Метод простых итераций. Метод Зейделя

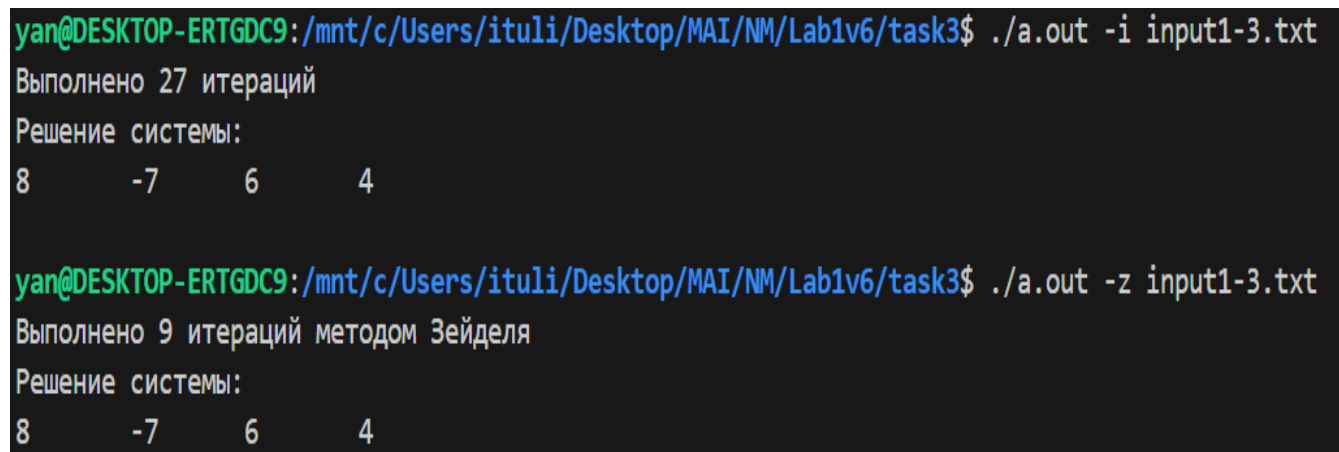
7 Постановка задачи

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: 6

$$\begin{cases} 23x_1 - 6x_2 - 5x_3 + x_4 = 232 \\ 8x_1 + 22x_2 - 2x_3 + 5x_4 = -82 \\ 7x_1 - 6x_2 + 18x_3 - x_4 = 202 \\ 3x_1 + 5x_2 + 5x_3 - 19x_4 = -57 \end{cases}$$

8 Результаты работы



```
yan@DESKTOP-ERTGDC9:/mnt/c/Users/ituli/Desktop/MAI/NM/Lab1v6/task3$ ./a.out -i input1-3.txt
Выполнено 27 итераций
Решение системы:
8      -7      6      4

yan@DESKTOP-ERTGDC9:/mnt/c/Users/ituli/Desktop/MAI/NM/Lab1v6/task3$ ./a.out -z input1-3.txt
Выполнено 9 итераций методом Зейделя
Решение системы:
8      -7      6      4
```

Рис. 3: Вывод программы в консоли

9 Исходный код

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <cmath>
5
6  using Matrix = std::vector<std::vector<double>>>;
7
8
9  class IterationMatrix{
10 public:
11     IterationMatrix(std::string filename, bool useZeidel);
12     std::vector<double> solve;
13     void print_vector(const std::vector<double> &x);
14 private:
15     int n;
16     double epsilon;
17     Matrix data;
18     std::vector<double> b;
19     void toYakobi(Matrix &a, std::vector<double> &d);
20     double getNorm(const Matrix &a);
21     double getDiffNorm(const std::vector<double> &xk, const std::vector<double> &
        xk_inc);
22     std::vector<double> simpleIteration(double norm);
23     std::vector<double> zeidel(double norm);
24 };
25
26
27 IterationMatrix::IterationMatrix(std::string filename, bool useZeidel){
28     std::ifstream idescr(filename);
29     //
30     int cnt = 0;
31     while(idescr >> epsilon){
32         cnt++;
33     }
34     n = (-1 + sqrt(1 + 4*(cnt-1))) / 2;
35     idescr.close();
36     idescr.open(filename);
37     //
38     data.resize(n, std::vector<double>(n));
39     for(int i = 0; i < n; i++){
40         for(int j = 0; j < n; j++){
41             idescr >> data[i][j];
42         }
43     }
44     b.resize(n);
45     for(int i = 0; i < n; i++){
46         idescr >> b[i];
```

```

47     }
48     idescr.close();
49
50     //
51     toYakobi(data, b);
52
53     //
54     double g = getNorm(data);
55     if(g >= 1){
56         std::cout << "\t    !" << std::endl;
57     } else {
58         //      epsilon
59         solve = useZeidel? zeidel(g) : simpleIteration(g);
60     }
61 }
62
63
64 void IterationMatrix::toYakobi(Matrix &a, std::vector<double> &d){
65     for(int i = 0; i < n; i++){
66         if(!a[i][i]){
67             break;
68         }
69         for(int j = 0; j < n; j++){
70             a[i][j] = (i == j)? a[i][i]: -a[i][j]/a[i][i];
71         }
72         d[i] /= a[i][i];
73         a[i][i] = 0;
74     }
75 }
76
77
78 double IterationMatrix::getNorm(const Matrix &a){
79     double g = 0;
80     for(int i = 0; i < n; i++){
81         double sum = 0;
82         for(int j = 0; j < n; j++){
83             sum += fabs(a[i][j]);
84         }
85         g = (sum > g)? sum: g;
86     }
87     return g;
88 }
89
90
91 double IterationMatrix::getDiffNorm(const std::vector<double> &xk, const std::vector<
92     double> &xk_inc){
93     double f = 0;
94     for(int i = 0; i < n; i++){
95         if(fabs(xk[i] - xk_inc[i]) > f){

```

```

95         f = fabs(xk[i] - xk_inc[i]);
96     }
97 }
98 return f;
99 }
100
101
102 std::vector<double> IterationMatrix::simpleIteration(double norm){
103     std::vector<double> xk(n);
104     std::vector<double> xk_inc(b);
105     double precision = getDiffNorm(xk, xk_inc) * norm / (1 - norm);
106     int k = 0;
107     while(precision > epsilon) {
108         xk = xk_inc;
109         for(int i = 0; i < n; i++){
110             xk_inc[i] = b[i];
111             for(int j = 0; j < n; j++){
112                 xk_inc[i] += data[i][j]*xk[j];
113             }
114         }
115         precision = getDiffNorm(xk, xk_inc) * norm / (1 - norm);
116         k++;
117     };
118     std::cout << " " << k << " " << std::endl;
119     return xk_inc;
120 }
121
122
123 std::vector<double> IterationMatrix::zeidel(double norm){
124     std::vector<double> xk(b);
125     std::vector<double> xk_inc(n);
126     double precision = getDiffNorm(xk, xk_inc) * norm / (1 - norm);
127     int k = 0;
128     while(precision > epsilon) {
129         xk = xk_inc;
130         for(int i = 0; i < n; i++){
131             xk_inc[i] = b[i];
132             for(int j = 0; j < n; j++){
133                 xk_inc[i] += data[i][j]*xk_inc[j];
134             }
135         }
136         precision = getDiffNorm(xk, xk_inc) * norm / (1 - norm);
137         k++;
138     };
139     std::cout << " " << k << " " << std::endl;
140     return xk_inc;
141 }
142
143

```

```

144 void IterationMatrix::print_vector(const std::vector<double> &x){
145     for(int i = 0; i < n; i++){
146         std::cout << std::round(x[i]*100)/100.0 << '\t';
147     }
148     std::cout << std::endl << std::endl;
149 }
150
151
152
153
154 int main(int argc, char **argv){
155     std::string method = argv[1];
156     IterationMatrix matrix = IterationMatrix(argv[2], method == "-z");
157     if(matrix.solve.empty()){
158         std::cout << " ." << std::endl;
159     } else {
160         std::cout << " :" << std::endl;
161         matrix.print_vector(matrix.solve);
162     }
163     return 0;
164 }

```

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: 6

$$\begin{pmatrix} 9 & 2 & -7 \\ 2 & -4 & -1 \\ -7 & -1 & 1 \end{pmatrix}$$

11 Результаты работы

```
Выполнено 5 итераций

Собственные значения:
13.35    -4.3    -3.05

Собственные векторы:
0.86    -0.17    0.49
0.13     0.99    0.11
-0.5    -0.03    0.87
```

Рис. 4: Вывод программы в консоли

12 Исходный код

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <cmath>
5
6  using Matrix = std::vector<std::vector<double>>>;
7
8  Matrix operator * (const Matrix &a, const Matrix &b){
9      Matrix res;
10     int n = a.size();
11     int p = b.size();
12     int m = (*b.begin()).size();
13     if(a.empty() || b.empty()){
14         return res;
15     }
16     res.resize(n, std::vector<double>(m));
17     for(int i = 0; i < n; i++){
18         for(int j = 0; j < m; j++){
19             for(int k = 0; k < p; k++){
20                 res[i][j] += a[i][k]*b[k][j];
21             }
22         }
23     }
24     return res;
25 }
26
27
28 class SymmetricalMatrix{
29     public:
30         void print_vector(const std::vector<double> &b);
31         void print_matrix(const Matrix &matrix);
32         SymmetricalMatrix(std::string filename);
33         std::vector<double> eigen_val;
34         Matrix eigen_vec;
35     private:
36         int n;
37         double epsilon;
38         Matrix data;
39         bool checkSymmetric(Matrix &matrix);
40         bool checkPresicion(Matrix &matrix);
41         std::pair<int, int> maxElem(Matrix &matrix);
42         Matrix E();
43         void transpose(Matrix &matrix);
44 };
45
46
47
```



```

48 void SymmetricalMatrix::print_matrix(const Matrix &matrix){
49     if(matrix.empty()){
50         std::cout << "Error: couldn't print empty matrix" << std::endl;
51         return;
52     }
53     for(int i = 0; i < n; i++){
54         for(int j = 0; j < n; j++){
55             std::cout << std::round(matrix[i][j]*100)/100.0 << '\t';
56         }
57         std::cout << std::endl;
58     }
59     std::cout << std::endl;
60 }
61
62
63 void SymmetricalMatrix::print_vector(const std::vector<double> &b){
64     if(b.empty()){
65         std::cout << "Error: couldn't print empty vector" << std::endl;
66         return;
67     }
68     for(int i = 0; i < n; i++){
69         std::cout << std::round(b[i]*100)/100.0 << '\t';
70     }
71     std::cout << std::endl << std::endl;
72 }
73
74
75
76 bool SymmetricalMatrix::checkSymmetric(Matrix &matrix){
77     for(int i = 0; i < n; i++){
78         for(int j = 0; j < n; j++){
79             if(matrix[i][j] != matrix[j][i]){
80                 return false;
81             }
82         }
83     }
84     return true;
85 }
86
87
88 SymmetricalMatrix::SymmetricalMatrix(std::string filename){
89     std::ifstream idescr(filename);
90     //
91     int cnt = 0;
92     while(idescr >> epsilon){
93         cnt++;
94     }
95     n = sqrt(cnt - 1);
96     idescr.close();

```

```

97     idescr.open(filename);
98     //
99     data.resize(n, std::vector<double>(n));
100    for(int i = 0; i < n; i++){
101        for(int j = 0; j < n; j++){
102            idescr >> data[i][j];
103        }
104    }
105    idescr.close();
106
107    //
108    if(!checkSymmetric(data)){
109        std::cout << "Error: the matrix is not symmetric" << std::endl;
110    } else {
111        //       $V = E$ 
112        eigen_vec = E();
113        int k = 0;
114        while(checkPresicion(data)){
115            //
116            std::pair<int, int> indexes = maxElem(data);
117            int l = indexes.first;
118            int m = indexes.second;
119            //       $(U)$ 
120            Matrix U = E();
121            //       $(\phi)$ 
122            double phi;
123            if(data[m][m] == data[l][l]){
124                phi = atan(1);
125            } else {
126                phi = atan(2*data[l][m]/(data[l][l]-data[m][m]))/2;
127            }
128            //
129            U[l][l] = cos(phi);
130            U[l][m] = -sin(phi);
131            U[m][l] = sin(phi);
132            U[m][m] = cos(phi);
133            //       $(UT)$ 
134            Matrix UT = U;
135            transpose(UT);
136            //       $(\lambda = UT \times data \times U)$ 
137            data = UT*data*U;
138            //
139            eigen_vec = eigen_vec*U;
140            k++;
141        }
142        //
143        eigen_val.resize(n);
144        for(int i = 0; i < n; i++){
145            eigen_val[i] = data[i][i];

```

```

146     }
147     std::cout << " " << k << " \n" << std::endl;
148 }
149 }
150
151
152 bool SymmetricalMatrix::checkPresicion(Matrix &matrix){
153     double f = 0;
154     for(int i = 0; i < n; i++){
155         for(int j = i+1; j < n; j++){
156             f += matrix[i][j]*matrix[i][j];
157         }
158     }
159     return sqrt(f) > epsilon;
160 }
161
162
163 std::pair<int, int> SymmetricalMatrix::maxElem(Matrix &matrix){
164     double maxElem = 0;
165     int l = 0;
166     int m = 1;
167     for(int i = 0; i < n; i++){
168         for(int j = i + 1; j < n; j++){
169             if(fabs(matrix[i][j]) > maxElem){
170                 maxElem = fabs(matrix[i][j]);
171                 l = i;
172                 m = j;
173             }
174         }
175     }
176     return std::make_pair(l, m);
177 }
178
179
180 Matrix SymmetricalMatrix::E(){
181     Matrix matrix(n, std::vector<double>(n));
182     for(int i = 0; i < n; i++){
183         matrix[i][i] = 1;
184     }
185     return matrix;
186 }
187
188
189 void SymmetricalMatrix::transpose(Matrix &matrix){
190     Matrix res(n, std::vector<double>(n, 0));
191     for(int i = 0; i < n; i++){
192         for(int j = 0; j < n; j++){
193             res[i][j] = matrix[j][i];
194         }

```

```

195     }
196     matrix = res;
197 }
198
199
200 int main(int argc, char** argv){
201     SymmetricalMatrix matrix = SymmetricalMatrix(argv[1]);
202     std::cout << " : " << std::endl;
203     matrix.print_vector(matrix.eigen_val);
204     std::cout << " : " << std::endl;
205     matrix.print_matrix(matrix.eigen_vec);
206
207     return 0;
208 }

```

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: 6

$$\begin{pmatrix} 8 & -1 & -3 \\ -5 & 9 & -8 \\ 4 & -5 & 7 \end{pmatrix}$$

14 Результаты работы

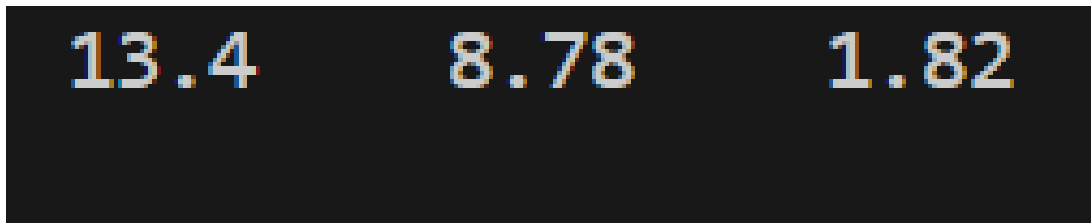


Рис. 5: Вывод программы в консоли

15 Исходный код

```
1  #include<iostream>
2  #include<fstream>
3  #include<vector>
4  #include<cmath>
5
6
7  using Matrix = std::vector<std::vector<double>>>;
8
9  Matrix operator * (const Matrix &a, const Matrix &b){
10     Matrix res;
11     int n = a.size();
12     int p = b.size();
13     int m = (*b.begin()).size();
14     if(a.empty() || b.empty()){
15         return res;
16     }
17     res.resize(n, std::vector<double>(m));
18     for(int i = 0; i < n; i++){
19         for(int j = 0; j < m; j++){
20             for(int k = 0; k < p; k++){
21                 res[i][j] += a[i][k]*b[k][j];
22             }
23         }
24     }
25     return res;
26 }
27
28 Matrix operator * (double alpha, const Matrix &A){
29     Matrix res;
30     int n = A.size();
31     int m = (*A.begin()).size();
32     if(A.empty()){
33         return res;
34     }
35     res.resize(n, std::vector<double>(m));
36     for(int i = 0; i < n; i++){
37         for(int j = 0; j < m; j++){
38             res[i][j] = alpha * A[i][j];
39         }
40     }
41     return res;
42 }
43
44
45 Matrix operator - (const Matrix &a, const Matrix &b){
46     Matrix res;
47     int n = a.size();
```

```

48     int m = (*a.begin()).size();
49     if(a.empty() || b.empty()){
50         return res;
51     }
52     res.resize(n, std::vector<double>(m));
53     for(int i = 0; i < n; i++){
54         for(int j = 0; j < m; j++){
55             res[i][j] = a[i][j] - b[i][j];
56         }
57     }
58     return res;
59 }
60
61
62 std::pair<double, double> discriminant(double a, double b, double c){
63     double D = b*b - 4 * a * c;
64     if(D < 0){
65         std::cout << "Error: discriminant less than zero" << std::endl;
66         return std::make_pair(0, 0);
67     }
68     double x1 = (-b + sqrt(D)) / (2 * a);
69     double x2 = (-b - sqrt(D)) / (2 * a);
70     return std::make_pair(x1, x2);
71 }
72
73
74
75 class QRMatrix{
76     public:
77         void print_vector(const std::vector<double> &b);
78         void print_matrix(const Matrix &matrix);
79         QRMatrix(std::string filename);
80         std::vector<double> eigenVal;
81     private:
82         int n;
83         double epsilon;
84         Matrix data;
85         Matrix E();
86         void transpose(Matrix &matrix);
87
88         Matrix getH(const Matrix &A, int col);
89         double getNorm(const std::vector<double> &b);
90         Matrix Q;
91         Matrix R;
92         void decompose(const Matrix &A);
93         Matrix prevIter;
94         std::pair<double, double> getRoots(const Matrix &A, int i);
95         void getEigenVal(const Matrix &A);
96 };

```

```

97
98
99 QRMatrix::QRMatrix(std::string filename){
100     std::ifstream idescr(filename);
101     //
102     int cnt = 0;
103     while(idescr >> epsilon){
104         cnt++;
105     }
106     n = sqrt(cnt - 1);
107     idescr.close();
108     idescr.open(filename);
109     //
110     data.resize(n, std::vector<double>(n));
111     for(int i = 0; i < n; i++){
112         for(int j = 0; j < n; j++){
113             idescr >> data[i][j];
114         }
115     }
116     idescr.close();
117     eigenVal.resize(n);
118     getEigenVal(data);
119 }
120
121
122 Matrix QRMatrix::E(){
123     Matrix matrix(n, std::vector<double>(n));
124     for(int i = 0; i < n; i++){
125         matrix[i][i] = 1;
126     }
127     return matrix;
128 }
129
130
131 void QRMatrix::transpose(Matrix &matrix){
132     Matrix res((*matrix.begin()).size(), std::vector<double>(matrix.size()));
133     for(int i = 0; i < (*matrix.begin()).size(); i++){
134         for(int j = 0; j < matrix.size(); j++){
135             res[i][j] = matrix[j][i];
136         }
137     }
138     matrix = res;
139 }
140
141
142 void QRMatrix::print_matrix(const Matrix &matrix){
143     if(matrix.empty()){
144         std::cout << "Error: couldn't print empty matrix" << std::endl;
145         return;

```



```

146     }
147     for(int i = 0; i < matrix.size(); i++){
148         for(int j = 0; j < (*matrix.begin()).size(); j++){
149             std::cout << std::round(matrix[i][j]*10000)/10000.0 << '\t';
150         }
151         std::cout << std::endl;
152     }
153     std::cout << std::endl;
154 }
155
156
157 void QRMatrix::print_vector(const std::vector<double> &b){
158     if(b.empty()){
159         std::cout << "Error: couldn't print empty vector" << std::endl;
160         return;
161     }
162     for(int i = 0; i < n; i++){
163         std::cout << std::round(b[i]*100)/100.0 << '\t';
164     }
165     std::cout << std::endl << std::endl;
166 }
167
168
169 Matrix QRMatrix::getH(const Matrix &A, int col){
170     std::vector<double> a(n);
171     for(int i = 0; i < n; i++){
172         a[i] = A[i][col];
173     }
174     Matrix v(1, std::vector<double>(n));
175     v[0][col] = (a[col] < 0)? a[col] - getNorm(a): a[col] + getNorm(a);
176     for(int i = col + 1; i < n; i++){
177         v[0][i] = a[i];
178     }
179     Matrix vT(v);
180     transpose(v);
181     Matrix vTv = vT*v;
182     Matrix vvT = v*vT;
183     Matrix temp = (2 / vTv[0][0])*vvT;
184     Matrix identity = E();
185     return identity - temp;
186 }
187
188
189 double QRMatrix::getNorm(const std::vector<double> &b){
190     double sum = 0;
191     for(double elem: b){
192         sum += elem*elem;
193     }
194     return sqrt(sum);

```

```

195 }
196
197
198 void QRMatrix::decompose(const Matrix &A){
199     Q = E();
200     R = A;
201     for(int i = 0; i < n - 1; i++){
202         Matrix H = getH(R, i);
203         Q = Q * H;
204         R = H * R;
205     }
206 }
207
208
209 std::pair<double, double> QRMatrix::getRoots(const Matrix &A, int i){
210     double a, b, c;
211     a = 1;
212     b = -A[i][i] - A[i+1][i+1];
213     c = A[i][i]*A[i+1][i+1] - A[i][i+1]*A[i+1][i];
214     std::pair<double, double> lambdas = discriminant(a, b, c);
215     return lambdas;
216 }
217
218
219 void QRMatrix::getEigenVal(const Matrix &A){
220     Matrix Ak = A;
221     int i = 0;
222     int complexCol = -1;
223     int cnt = 0;
224     while(i < n){
225         decompose(Ak);
226         Ak = R * Q;
227         std::vector<double> a(n - i - 1);
228         for(int j = i + 1; j < n; j++){
229             a[j - i - 1] = Ak[j][i];
230         }
231         if(getNorm(a) < epsilon){
232             eigenVal[i] = Ak[i][i];
233             i++;
234         } else {
235             std::vector<double> b(n - i - 1);
236             for(int j = i + 2; j < n; j++){
237                 b[j - i - 2] = Ak[j][i];
238             }
239             bool cond0 = getNorm(b) < epsilon;
240             if(complexCol == i){
241                 std::pair<double, double> lambdas = getRoots(A, i);
242                 std::pair<double, double> prevLambdas = getRoots(prevIter, i);
243                 bool cond1 = fabs(lambdas.first - prevLambdas.first) < epsilon;

```

```

244         bool cond2 = fabs(lambdas.second - prevLambdas.second) < epsilon;
245         if(cond0 && cond1 && cond2){
246             eigenVal[i] = lambdas.first;
247             eigenVal[i+1] = lambdas.second;
248             i += 2;
249         }
250     }
251     if(cond0){
252         complexCol = i;
253     }
254 }
255 prevIter = Ak; cnt++;
256 }
257 }
258
259
260 int main(int argc, char **argv){
261     QRMatrix matrix = QRMatrix(argv[1]);
262     matrix.print_vector(matrix.eigenVal);
263     return 0;
264 }

```