

**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: Батулин Е.А.
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: 1

$$\begin{cases} x_1 + 2x_2 - 2x_3 + 6x_4 = 24 \\ -3x_1 - 5x_2 + 14x_3 + 13x_4 = 41 \\ x_1 + 2x_2 - 2x_3 - 2x_4 = 0 \\ -2x_1 - 4x_2 + 5x_3 + 10x_4 = 20 \end{cases}$$

2 Результаты работы

```
1 2 -2 6 -3 -5 14 13 1 2 -2 -2 -2 -4 5 10
24 41 0 20
1766 -725 128 -6

-8

62 -2 18 42.5
-25.25 1 -8 -18.125
4.5 0 1 2.75
-0.25 -0 -0 -0.125
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cmath>
5
6  using namespace std;
7
8  class Matrix {
9  public:
10     int rows, cols;
11     double **a;
12
13     Matrix (){
14         rows = 0;
15         cols = 0;
16         a = new double*[rows];
17         for(int i = 0; i < rows; i++)
18             a[i] = new double[cols];
19     }
20
21     Matrix(int n, int m, bool identity = 0)
22     {
23         rows = n;
24         cols = m;
25         a = new double *[rows];
26         for (int i = 0; i < rows; i++){
27             a[i] = new double[cols];
28             for (int j = 0; j < cols; j++){
29                 a[i][j] = identity * (i == j);
30             }
31         }
32     }
33
34     void swapRows(int row1, int row2)
35     {
36         swap(a[row1], a[row2]);
37     }
38
39     void scan()
40     {
41         for(int i = 0; i < rows; i++)
42         {
43             for(int j = 0; j < cols; j++)
44             {
45                 scanf ("%lf", &a[i][j]);
46             }
47         }
```

```

48     }
49
50 void print()
51 {
52     for(int i = 0; i < rows; i++)
53     {
54         for(int j = 0; j < cols; j++)
55         {
56             cout << "\t" << a[i][j] << "\t";
57         }
58         cout << endl;
59     }
60 }
61
62 void inverse(Matrix &L, Matrix &U, double Z[])
63 {
64     Matrix E = Matrix(rows, cols, 1);
65     for (int k = 0; k < rows; ++k){
66         for (int i = 0; i < cols; ++i){
67             double s = 0;
68             for (int j = 0; j < i; ++j){
69                 s += L[i][j] * Z[j];
70             }
71             Z[i] = E[i][k] - s;
72         }
73
74         for (int i = rows - 1; i > -1; --i){
75             double s = 0;
76             for (int j = i + 1; j < rows; ++j){
77                 s += U[i][j] * a[j][k];
78             }
79             a[i][k] = (Z[i] - s) / U[i][i];
80         }
81     }
82 }
83
84 double det()
85 {
86     double d = 0;
87     if (rows == 1){
88         return a[0][0];
89     }
90     else if (rows == 2){
91         return a[0][0] * a[1][1] - a[0][1] * a[1][0];
92     }
93     for (int k = 0; k < rows; ++k) {
94         Matrix M = Matrix(rows - 1, cols - 1);
95         for (int i = 1; i < rows; ++i) {
96             int t = 0;

```

```

97         for (int j = 0; j < rows; ++j) {
98             if (j == k)
99                 continue;
100             M[i-1][t] = a[i][j];
101             t += 1;
102         }
103     }
104     d += pow(-1, k + 2) * a[0][k] * M.det();
105 }
106 return d;
107 }
108
109 double* operator[](int i)
110 {
111     return a[i];
112 }
113
114 Matrix& operator*=(Matrix& m)
115 {
116     if (cols != m.rows)
117     {
118         throw "Wait. That's illegal.";
119     }
120     Matrix temp(m.rows, m.cols);
121     for (int i = 0; i < temp.rows; ++i)
122     {
123         for (int j = 0; j < temp.cols; ++j)
124         {
125             for (int k = 0; k < cols; ++k)
126             {
127                 temp.a[i][j] += (a[i][k] * m.a[k][j]);
128             }
129         }
130     }
131     return (*this = temp);
132 }
133 };
134
135 void LU(Matrix &A, Matrix &B, Matrix &L, Matrix &U, Matrix M[])
136 {
137     int n = A.rows;
138     L = Matrix(n, n, 1);
139
140     for(int i = 0; i < n; i++)
141         for(int j = 0; j < n; j++)
142             U[i][j] = A[i][j];
143
144     for (int k = 0; k < n - 1; k++){
145         M[k] = Matrix(n, n, 1);

```

```

146     for (int i = k + 1; i < n; i++){
147         if (U[k][k] == 0){
148             int j = k + 1;
149             while (U[j][j] == 0 and j < n){
150                 j += 1;
151             }
152             if (j == n){
153                 break;
154             }
155             U.swapRows(k, j);
156             B.swapRows(k, j);
157         }
158         M[k][i][k] = U[i][k] / U[k][k];
159         for (int j = k; j < n; ++j){
160             U[i][j] -= M[k][i][k] * U[k][j];
161         }
162     }
163     L *= M[k];
164 }
165 }
166
167 /*
168 1 2 -2 6 -3 -5 14 13 1 2 -2 -2 -2 -4 5 10
169 24 41 0 20
170 */
171
172 int main()
173 {
174     int n = 4;
175     Matrix A = Matrix(n, n);
176     Matrix U = Matrix(n, n);
177     Matrix B = Matrix(n, 1);
178     A.scan();
179     B.scan();
180
181     Matrix M[n - 1];
182     Matrix L = Matrix(n, n, 1);
183
184     LU(A,B,L,U,M);
185
186     double Z[n];
187     for (int i = 0; i < n; ++i){
188         double s = 0;
189         for (int j = 0; j < i; ++j){
190             double tmp = L[i][j] * Z[j];
191             s += tmp;
192         }
193         Z[i] = B[i][0] - s;
194     }

```

```

195
196 double X[n];
197 for (int i = n - 1; i > -1; --i){
198     double s = 0;
199     for (int j = i + 1; j < n; ++j){
200         s += U[i][j] * X[j];
201     }
202     X[i] = (Z[i] - s) / U[i][i];
203 }
204
205 for (int i = 0; i < n; ++i){
206     cout << X[i] << " ";
207 }
208 cout << "\n\n";
209
210 cout << U.det() << "\n\n";
211
212 Matrix Ai = Matrix(n, n);
213 Ai.inverse(L,U,Z);
214 Ai.print();
215 }

```

1.2 Метод прогонки

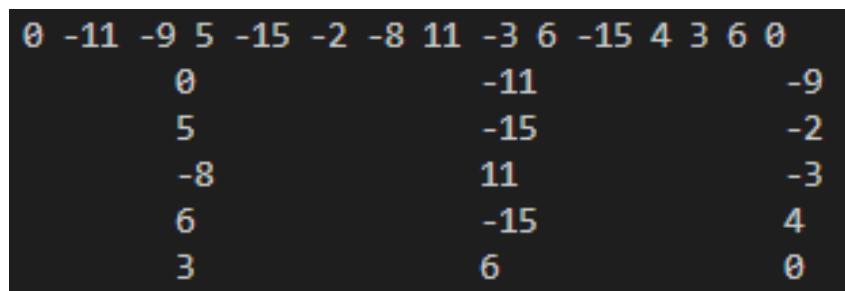
4 Постановка задачи

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: 1

$$\begin{cases} -11x_1 - 9x_2 = -122 \\ 5x_1 - 15x_2 - 2x_3 = -48 \\ -8x_2 + 11x_3 - 3x_4 = -14 \\ 6x_3 - 15x_4 + 4x_5 = -50 \\ 3x_4 + 6x_5 = 42 \end{cases}$$

5 Результаты работы



0	-11	-9	5	-15	-2	-8	11	-3	6	-15	4	3	6	0
		0						-11						-9
		5						-15						-2
		-8						11						-3
		6						-15						4
		3						6						0

Рис. 2: Вывод программы в консоли

6 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <string>
4 | #include <cmath>
5 |
6 | using namespace std;
7 |
8 | class Matrix {
9 | public:
10 |     int rows, cols;
11 |     double **a;
12 |
13 |     Matrix () {
14 |         rows = 0;
15 |         cols = 0;
16 |         a = new double*[rows];
17 |         for(int i = 0; i < rows; i++)
18 |             a[i] = new double[cols];
19 |     }
20 |
21 |     Matrix(int n, int m, bool identity = 0)
22 |     {
23 |         rows = n;
24 |         cols = m;
25 |         a = new double *[rows];
26 |         for (int i = 0; i < rows; i++){
27 |             a[i] = new double[cols];
28 |             for (int j = 0; j < cols; j++){
29 |                 a[i][j] = identity * (i == j);
30 |             }
31 |         }
32 |     }
33 |
34 |     void swapRows(int row1, int row2)
35 |     {
36 |         swap(a[row1], a[row2]);
37 |     }
38 |
39 |     void scan()
40 |     {
41 |         for(int i = 0; i < rows; i++)
42 |         {
43 |             for(int j = 0; j < cols; j++)
44 |             {
45 |                 scanf ("%lf", &a[i][j]);
46 |             }
47 |         }
```

```

48     }
49
50 void print()
51 {
52     for(int i = 0; i < rows; i++)
53     {
54         for(int j = 0; j < cols; j++)
55         {
56             cout << "\t" << a[i][j] << "\t";
57         }
58         cout << endl;
59     }
60 }
61
62 double* operator[](int i)
63 {
64     return a[i];
65 }
66 };
67
68
69 double determinant(Matrix M)
70 {
71     double det = 1;
72     int p = 1;
73     for (int i=0; i < M.rows; i++)
74     {
75         det *= pow(-1, p) * M[i][i];
76         p++;
77     }
78     return det;
79 }
80
81 /*
82 0 -11 -9 5 -15 -2 -8 11 -3 6 -15 4 3 6 0
83 -122 -48 -14 -50 42
84 */
85
86 int main()
87 {
88
89     int n = 5;
90     Matrix A = Matrix(n, 3);
91     double B[n] = {-122, -48, -14, -50, -42};
92     A.scan();
93     A.print();
94     //B.scan();
95
96     //

```

```

97     double P[n], Q[n], X[n];
98     P[0] = -A[0][2] / A[0][1];
99     Q[0] = B[0] / A[0][1];
100    for (int i = 1; i < n; ++i){
101        P[i] = -A[i][2] / (A[i][1] + A[i][0] * P[i - 1]);
102        Q[i] = (B[i] - A[i][0] * Q[i - 1]) / (A[i][1] + A[i][0] * P[i - 1]);
103    }
104
105    //
106    X[-1] = Q[-1];
107    for (int i = n - 2; i > -1; --i){
108        X[i] = P[i] * X[i + 1] + Q[i];
109    }
110    for (int i = 0; i < n; ++i){
111        cout << X[i] << " ";
112    }
113 }
114
115     for(int i = 0; i < n; ++i) {
116         if(i == 0) {
117             P[i] = -A[i][i+1] / A[i][i];
118             Q[i] = d[i] / A[i][i];
119         } else if(i == n - 1) {
120             P[i] = 0;
121             Q[i] = (d[i] - A[i][i - 1] * Q[i - 1]) / (A[i][i] + A[i][i - 1] * P[i - 1])
122             ;
123         } else {
124             P[i] = -A[i][i+1] / (A[i][i] + A[i][i - 1] * P[i - 1]);
125             Q[i] = (d[i] - A[i][i - 1] * Q[i - 1]) / (A[i][i] + A[i][i - 1] * P[i - 1])
126             ;
127         }
128     }
129
130     for(int i = n - 1; i >= 0; --i) {
131         if(i == n - 1) x[i] = Q[i];
132         else {
133             x[i] = P[i] * x[i + 1] + Q[i];
134         }
135     }

```

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: 1

$$\begin{cases} 19x_1 - 4x_2 + 9x_3 - x_4 = 100 \\ -2x_1 + 20x_2 - 2x_3 - 7x_4 = -5 \\ 6x_1 - 5x_2 - 25x_3 + 9x_4 = 34 \\ -3x_2 - 9x_3 + 12x_4 = 69 \end{cases}$$

8 Результаты работы

```
19 -4 -9 -1 -2 20 -2 -7 6 5 -25 9 0 -3 -9 12
100 -5 34 69
      10.195
      5.65194
      6.56334
      12.0817

26
      12.4579
      7.85907
      10.008
      15.2542

27
```

Рис. 3: Вывод программы в консоли

9 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cmath>
5
6  using namespace std;
7
8  class Matrix {
9  public:
10     int rows, cols;
11     double **a;
12
13     Matrix (){
14         rows = 0;
15         cols = 0;
16         a = new double*[rows];
17         for(int i = 0; i < rows; i++)
18             a[i] = new double[cols];
19     }
20
21     Matrix(int n, int m, bool identity = 0)
22     {
23         rows = n;
24         cols = m;
25         a = new double *[rows];
26         for (int i = 0; i < rows; i++){
27             a[i] = new double[cols];
28             for (int j = 0; j < cols; j++){
29                 a[i][j] = identity * (i == j);
30             }
31         }
32     }
33
34     void swapRows(int row1, int row2)
35     {
36         swap(a[row1], a[row2]);
37     }
38
39     void scan()
40     {
41         for(int i = 0; i < rows; i++)
42         {
43             for(int j = 0; j < cols; j++)
44             {
45                 scanf ("%lf", &a[i][j]);
46             }
47         }
```

```

48     }
49
50 void print()
51 {
52     for(int i = 0; i < rows; i++)
53     {
54         for(int j = 0; j < cols; j++)
55         {
56             cout << "\t" << a[i][j] << "\t";
57         }
58         cout << endl;
59     }
60 }
61
62 double* operator[] (int i)
63 {
64     return a[i];
65 }
66
67 Matrix transpose()
68 {
69     Matrix C = Matrix (cols, rows);
70     for (int i = 0; i < rows; ++ i){
71         for (int j = 0; j < cols; ++ j){
72             C[j][i] = a[i][j];
73         }
74     }
75     return C;
76 }
77
78 Matrix minor(int y, int z)
79 {
80     Matrix C = Matrix (rows - 1, cols - 1);
81     int k = 0;
82     for (int i = 0; i < rows; ++i){
83         if (i > y){
84             k = 1;
85         }
86         for (int j = 0; j < cols; ++j){
87             if (i != y){
88                 if (j < z){
89                     C[i - k][j] = a[i][j];
90                 }
91                 else if (j > z){
92                     C[i - k][j - 1] = a[i][j];
93                 }
94             }
95         }
96     }

```

```

97     return C;
98 }
99
100 Matrix inverse()
101 {
102     double d = this->determinant();
103     Matrix inv_A = Matrix(rows, cols);
104     for(int i = 0; i < rows; ++i){
105         for(int j = 0; j < rows; ++j){
106             Matrix M = Matrix(rows - 1, rows - 1);
107             M = this->minor(i, j);
108             inv_A[i][j] = pow(-1.0, i + j + 2) * M.determinant() / d;
109         }
110     }
111     inv_A = inv_A.transpose();
112     return inv_A;
113 }
114
115 double determinant()
116 {
117     double d = 0;
118     if (rows == 1){
119         return a[0][0];
120     }
121     else if (rows == 2){
122         return a[0][0] * a[1][1] - a[0][1] * a[1][0];
123     }
124     for (int k = 0; k < rows; ++k) {
125         Matrix M = Matrix(rows - 1, cols - 1);
126         for (int i = 1; i < rows; ++i) {
127             int t = 0;
128             for (int j = 0; j < rows; ++j) {
129                 if (j == k)
130                     continue;
131                 M[i-1][t] = a[i][j];
132                 t += 1;
133             }
134         }
135         d += pow(-1, k + 2) * a[0][k] * M.determinant();
136     }
137     return d;
138 }
139 double Norm(){
140     double norm = 0;
141     for (int i = 0; i < rows; ++i){
142         for (int j = 0; j < cols; ++j){
143             norm += pow(a[i][j], 2);
144         }
145     }

```

```

146         return pow(norm, 0.5);
147     }
148 };
149
150 Matrix operator* (Matrix & A, Matrix & B){
151     Matrix C = Matrix (A.rows, B.cols);
152     for (int i = 0; i < A.rows; ++ i){
153         for (int j = 0; j < B.cols; ++ j){
154             for (int k = 0; k < A.cols; ++ k){
155                 C[i][j] += A[i][k] * B[k][j];
156             }
157         }
158     }
159     return C;
160 }
161
162 Matrix operator+ (Matrix A, Matrix B){
163     Matrix C = Matrix(A.rows, A.cols);
164     for (int i = 0; i < A.rows; ++i){
165         for (int j = 0; j < A.cols; ++j){
166             C[i][j] = A[i][j] + B[i][j];
167         }
168     }
169     return C;
170 }
171
172 Matrix operator- (Matrix A, Matrix B){
173     Matrix C = Matrix(A.rows, A.cols);
174     for (int i = 0; i < A.rows; ++i){
175         for (int j = 0; j < A.cols; ++j){
176             C[i][j] = A[i][j] - B[i][j];
177         }
178     }
179     return C;
180 }
181
182 /*
183 19 -4 -9 -1 -2 20 -2 -7 6 5 -25 9 0 -3 -9 12
184 100 -5 34 69
185 */
186
187 int main()
188 {
189
190     int n = 4;
191     Matrix A = Matrix (n, n);
192     Matrix Alpha = Matrix (n, n);
193     Matrix Down = Matrix (n, n);
194     Matrix E = Matrix (n, n, 1);

```



```

195 Matrix B = Matrix (n, 1);
196 Matrix Beta = Matrix(n, 1);
197 A.scan();
198 B.scan();
199
200 //
201 for (int i = 0; i < n; ++i){
202     for (int j = 0; j < n; ++j){
203         double s = 0;
204         if (i != j){
205             Alpha[i][j] = -A[i][j] / A[i][i];
206             if (i > j){
207                 Down[i][j] = Alpha[i][j];
208             }
209             s += abs(A[i][j]);
210         }
211         else{
212             Alpha[i][j] = 0;
213         }
214     }
215     Beta[i][0] = B[i][0] / A[i][i];
216 }
217
218 //
219 Matrix X[2] = {Beta, Beta + Alpha * Beta};
220 int k = 1;
221 double eps = 0.01;
222 while ((X[1] - X[0]).Norm() > eps){
223     X[0] = X[1];
224     X[1] = Beta + Alpha * X[1];
225     k += 1;
226 }
227 X[1].print();
228 cout << "\n";
229 cout << k << "\n";
230
231 //
232 k = 1;
233 Matrix C = E - Down;
234 C = C.inverse();
235 C = C * Beta;
236 X[0] = Beta;
237 X[1] = Alpha * Beta + C;
238 while ((X[1] - X[0]).Norm() > eps){
239     X[0] = X[1];
240     X[1] = Alpha * X[1] + C;
241     k += 1;
242 }
243

```

```
244 || cout << "\n";  
245 || X[1].print();  
246 || cout << k << "\n";  
247 || }
```

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: 1

$$\begin{pmatrix} -7 & 4 & 5 \\ 4 & -6 & -9 \\ 5 & -9 & -8 \end{pmatrix}$$

11 Результаты работы



```
-7 4 5 4 -6 -9 5 -9 -8
-3.71065 2.07377 -19.3631
4
```

Рис. 4: Вывод программы в консоли

12 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cmath>
5
6  using namespace std;
7
8  class Matrix {
9  public:
10     int rows, cols;
11     double **a;
12
13     Matrix () {
14         rows = 0;
15         cols = 0;
16         a = new double*[rows];
17         for(int i = 0; i < rows; i++)
18             a[i] = new double[cols];
19     }
20
21     Matrix(int n, int m, bool identity = 0)
22     {
23         rows = n;
24         cols = m;
25         a = new double *[rows];
26         for (int i = 0; i < rows; i++){
27             a[i] = new double[cols];
28             for (int j = 0; j < cols; j++){
29                 a[i][j] = identity * (i == j);
30             }
31         }
32     }
33
34     void scan()
35     {
36         for(int i = 0; i < rows; i++)
37         {
38             for(int j = 0; j < cols; j++)
39             {
40                 scanf ("%lf", &a[i][j]);
41             }
42         }
43     }
44
45     void print()
46     {
47         for(int i = 0; i < rows; i++)
```

```

48 {
49     for(int j = 0; j < cols; j++)
50     {
51         cout << "\t" << a[i][j] << "\t";
52     }
53     cout << endl;
54 }
55 }
56
57 double* operator[] (int i)
58 {
59     return a[i];
60 }
61
62 Matrix transpose()
63 {
64     Matrix C = Matrix (cols, rows);
65     for (int i = 0; i < rows; ++ i){
66         for (int j = 0; j < cols; ++ j){
67             C[j][i] = a[i][j];
68         }
69     }
70     return C;
71 }
72
73 double crit()
74 {
75     double s = 0;
76     for (int i = 0; i < rows - 1; ++i){
77         for (int j = i + 1; j < cols; ++j){
78             s += pow(a[i][j], 2);
79         }
80     }
81     return pow(s, 0.5);
82 }
83
84 pair<int, int> find_max(){
85     int i_max = 0;
86     int j_max = 1;
87     for (int i = 0; i < rows - 1; ++i){
88         for (int j = i + 1; j < rows; ++j){
89             if (abs(a[i][j]) > abs(a[i_max][j_max])){
90                 i_max = i;
91                 j_max = j;
92             }
93         }
94     }
95     return pair<int, int>(i_max, j_max);
96 }

```

```

97 };
98
99 Matrix operator* (Matrix & A, Matrix & B){
100     Matrix C = Matrix (A.rows, B.cols);
101     for (int i = 0; i < A.rows; ++ i){
102         for (int j = 0; j < B.cols; ++ j){
103             for (int k = 0; k < A.cols; ++ k){
104                 C[i][j] += A[i][k] * B[k][j];
105             }
106         }
107     }
108     return C;
109 }
110
111 Matrix operator+ (Matrix A, Matrix B){
112     Matrix C = Matrix(A.rows, A.cols);
113     for (int i = 0; i < A.rows; ++i){
114         for (int j = 0; j < A.cols; ++j){
115             C[i][j] = A[i][j] + B[i][j];
116         }
117     }
118     return C;
119 }
120
121 Matrix operator- (Matrix A, Matrix B){
122     Matrix C = Matrix(A.rows, A.cols);
123     for (int i = 0; i < A.rows; ++i){
124         for (int j = 0; j < A.cols; ++j){
125             C[i][j] = A[i][j] - B[i][j];
126         }
127     }
128     return C;
129 }
130
131 /*
132 -7 4 5 4 -6 -9 5 -9 -8
133 */
134
135 int main()
136 {
137     int n = 3;
138     Matrix A = Matrix(n, n);
139     A.scan();
140     Matrix X = Matrix(n, n, 1);
141     double eps = 0.01;
142     int k = 0;
143
144     //
145     while (A.crit() > eps){

```

```

146     Matrix U = Matrix(n, n, 1);
147     Matrix U_T = Matrix(n, n, 1);
148     auto [i_max, j_max] = A.find_max();
149     double phi = atan(2 * A[i_max][j_max] / (A[i_max][i_max] - A[j_max][j_max])) / 2;
150     U[i_max][i_max] = cos(phi);
151     U[j_max][j_max] = cos(phi);
152     U[i_max][j_max] = -sin(phi);
153     U[j_max][i_max] = sin(phi);
154     U_T = U.transpose();
155     A = U_T * A;
156     A = A * U;
157     X = X * U;
158     k += 1;
159 }
160
161 Matrix Lambda = Matrix(1, n);
162 for (int i = 0; i < n; ++i){
163     Lambda[0][i] = A[i][i];
164 }
165
166 Lambda.print();
167 cout << k << "\n";
168 }

```

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: 1

$$\begin{pmatrix} 3 & -7 & -1 \\ -9 & -8 & 7 \\ 5 & 2 & 2 \end{pmatrix}$$

14 Результаты работы

```
3 -7 -1 -9 -8 7 5 2 2
      -13.5018          1.00397          4.26353
      -0.00802919      3.67831          -4.25969
      -0.00249093      2.50617          6.82348
10
-13.501796 -5.250898 + 2.863999i -5.250898 - 2.863999i
```

Рис. 5: Вывод программы в консоли

15 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cmath>
5
6  using namespace std;
7
8  class Matrix {
9  public:
10     int rows, cols;
11     double **a;
12
13     Matrix () {
14         rows = 0;
15         cols = 0;
16         a = new double*[rows];
17         for(int i = 0; i < rows; i++)
18             a[i] = new double[cols];
19     }
20
21     Matrix(int n, int m, bool identity = 0)
22     {
23         rows = n;
24         cols = m;
25         a = new double *[rows];
26         for (int i = 0; i < rows; i++){
27             a[i] = new double[cols];
28             for (int j = 0; j < cols; j++){
29                 a[i][j] = identity * (i == j);
30             }
31         }
32     }
33
34     void scan()
35     {
36         for(int i = 0; i < rows; i++)
37         {
38             for(int j = 0; j < cols; j++)
39             {
40                 scanf ("%lf", &a[i][j]);
41             }
42         }
43     }
44
45     void print()
46     {
47         for(int i = 0; i < rows; i++)
```

```

48     {
49         for(int j = 0; j < cols; j++)
50         {
51             cout << "\t" << a[i][j] << "\t";
52         }
53         cout << endl;
54     }
55 }
56
57 double* operator[] (int i)
58 {
59     return a[i];
60 }
61
62 void swapRows(int row1, int row2)
63 {
64     swap(a[row1], a[row2]);
65 }
66
67 Matrix transpose()
68 {
69     Matrix C = Matrix (cols, rows);
70     for (int i = 0; i < rows; ++ i){
71         for (int j = 0; j < cols; ++ j){
72             C[j][i] = a[i][j];
73         }
74     }
75     return C;
76 }
77
78 double crit()
79 {
80     double s = 0;
81     for (int i = 0; i < rows - 1; ++i){
82         for (int j = i + 1; j < cols; ++j){
83             s += pow(a[i][j], 2);
84         }
85     }
86     return pow(s, 0.5);
87 }
88
89 pair<int, int> find_max(){
90     int i_max = 0;
91     int j_max = 1;
92     for (int i = 0; i < rows - 1; ++i){
93         for (int j = i + 1; j < rows; ++j){
94             if (abs(a[i][j]) > abs(a[i_max][j_max])){
95                 i_max = i;
96                 j_max = j;

```

```

97         }
98     }
99 }
100 return pair<int, int>(i_max, j_max);
101 }
102 };
103
104 Matrix operator* (Matrix & A, double mult){
105     Matrix C = Matrix (A.rows, A.cols);
106     for (int i = 0; i < A.rows; ++i)
107     {
108         for (int j = 0; j < A.cols; ++j)
109         {
110             C[i][j] += A[i][j] * mult;
111         }
112     }
113     return C;
114 }
115
116 Matrix operator* (Matrix & A, Matrix & B){
117     Matrix C = Matrix (A.rows, B.cols);
118     for (int i = 0; i < A.rows; ++i){
119         for (int j = 0; j < B.cols; ++j){
120             for (int k = 0; k < A.cols; ++k){
121                 C[i][j] += A[i][k] * B[k][j];
122             }
123         }
124     }
125     return C;
126 }
127
128 Matrix operator+ (Matrix A, Matrix B){
129     Matrix C = Matrix(A.rows, A.cols);
130     for (int i = 0; i < A.rows; ++i){
131         for (int j = 0; j < A.cols; ++j){
132             C[i][j] = A[i][j] + B[i][j];
133         }
134     }
135     return C;
136 }
137
138 Matrix operator- (Matrix A, Matrix B){
139     Matrix C = Matrix(A.rows, A.cols);
140     for (int i = 0; i < A.rows; ++i){
141         for (int j = 0; j < A.cols; ++j){
142             C[i][j] = A[i][j] - B[i][j];
143         }
144     }
145     return C;

```

```

146 }
147
148 double scalar(Matrix &A, Matrix &B){
149     double s;
150     for (int i = 0; i < A.rows; ++i){
151         for (int j = 0; j < B.cols; ++j){
152             for (int k = 0; k < A.cols; ++k){
153                 s += A[i][k] * B[k][j];
154             }
155         }
156     }
157     return s;
158 }
159
160 int sign(double a){
161     return (a >= 0) ? 1 : -1;
162 }
163
164 pair<Matrix, Matrix> QR_decomposition(Matrix A){
165     int n = A.rows;
166     Matrix E = Matrix(n, n, 1);
167     Matrix Q = Matrix(n, n, 1);
168     Matrix H = Matrix(n, n);
169     for (int j = 0; j < n - 1; ++j){
170         Matrix v = Matrix(n, 1);
171         for (int i = j; i < n; ++i){
172             v[i][0] = A[i][j];
173             if (i == j){
174                 double s = 0;
175                 for (int k = j; k < n; ++k){
176                     s += pow(A[k][j], 2);
177                 }
178                 v[i][0] += sign(A[i][j]) * pow(s, 0.5);
179             }
180         }
181         Matrix v_T = v.transpose();
182         Matrix v1 = v * v_T;
183         H = v1 * 2;
184         H = E - H * (1 / scalar(v_T, v));
185         Q = Q * H;
186         A = H * A;
187     }
188     return pair<Matrix, Matrix> (Q, A);
189 }
190
191 pair<double, double> complex(double b, double c){
192     if (pow(b, 2) - 4 * c < 0){
193         return pair<double, double> (-b / 2, pow(- pow(b, 2) + 4 * c, 0.5) / 2);
194     }

```

```

195     else{
196         return pair<double, double> (-b / 2 + pow(pow(b, 2) - 4 * c, 0.5) / 2, 0);
197     }
198 }
199
200 bool endcheck(Matrix A[2], double eps){
201     int n = A[1].rows;
202     int j = 0;
203     while (j < n - 1){
204         pair<double, double> z0 = complex( - A[0][j][j] - A[0][j + 1][j + 1], A[0][j][j]
                * A[0][j + 1][j + 1] - A[0][j][j + 1] * A[0][j + 1][j]);
205         pair<double, double> z1 = complex( - A[1][j][j] - A[1][j + 1][j + 1], A[1][j][j]
                * A[1][j + 1][j + 1] - A[1][j][j + 1] * A[1][j + 1][j]);
206         if (z1.second == 0){
207             double s = 0;
208             for (int i = j + 1; i < n; ++i){
209                 s += pow(A[1][i][j], 2);
210             }
211             s = pow(s, 0.5);
212             if (s > eps){
213                 return true;
214             }
215         }
216         else{
217             if (pow(pow(z1.first - z0.first, 2) + pow(z1.second - z0.second, 2), 0.5) >
                eps){
218                 return true;
219             }
220             else{
221                 j += 1;
222             }
223         }
224         j += 1;
225     }
226     return false;
227 }
228
229 /*
230 3 -7 -1 -9 -8 7 5 2 2
231 */
232
233 int main()
234 {
235     int n = 3;
236     Matrix A = Matrix (n, n);
237     A.scan();
238
239     // QR-
240     double eps = 0.01;

```

```

241 Matrix AA[2] = {A, Matrix(n, n)};
242 auto [Q, R] = QR_decomposition(A);
243 AA[1] = R * Q;
244 int k = 1;
245
246 while (endcheck(AA, eps)){
247     auto [Q, R] = QR_decomposition(AA[1]);
248     AA[0] = AA[1];
249     AA[1] = R * Q;
250     k += 1;
251 }
252
253 //
254 string eigenvalue[n];
255 int i = 0;
256 while (i < n - 1){
257     pair<double, double> z = complex(AA[1][i][i] + AA[1][i + 1][i + 1], AA[1][i][i]
        * AA[1][i + 1][i + 1] - AA[1][i][i + 1] * AA[1][i + 1][i]);
258     if (z.second == 0){
259         eigenvalue[i] = to_string(AA[1][i][i]);
260     }
261     else{
262         eigenvalue[i] = to_string(z.first) + " + " + to_string(z.second) + "i";
263         i += 1;
264         eigenvalue[i] = to_string(z.first) + " - " + to_string(z.second) + "i";
265         i += 1;
266     }
267     i += 1;
268 }
269
270 AA[1].print();
271 cout << k << "\n";
272 for (int i = 0; i < n; ++i){
273     cout << eigenvalue[i] << " ";
274 }
275 }

```