

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

Москва, 2024

1.1 LU - разложение матриц

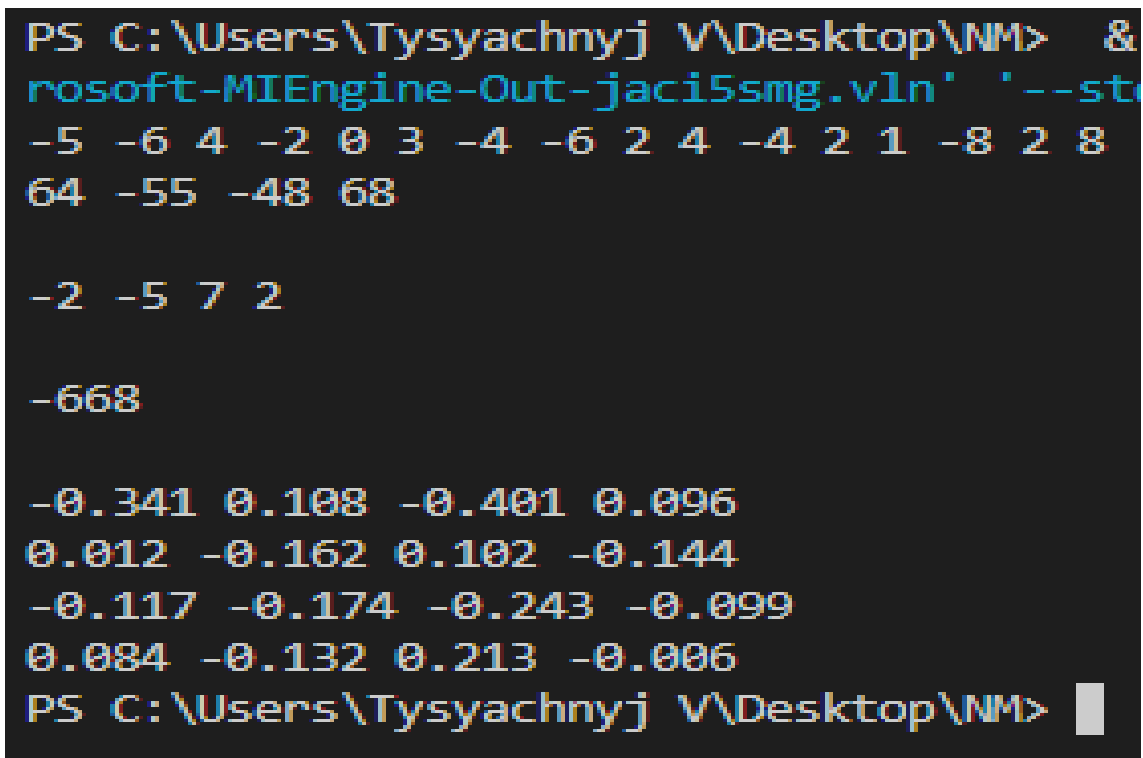
1 Постановка задачи

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: 25

$$\begin{cases} -5x_1 - 6x_2 + 4x_3 - 2x_4 = 64 \\ 3x_2 - 4x_3 - 6x_4 = -55 \\ 2x_1 + 4x_2 - 4x_3 + 2x_4 = -48 \\ x_1 - 8x_2 + 2x_3 + 8x_4 = 68 \end{cases}$$

2 Результаты работы



```
PS C:\Users\Tysyachnyj V\Desktop\NM> &
rossoft-MIEngine-Out-jaci5smg.vln' '--st
-5 -6 4 -2 0 3 -4 -6 2 4 -4 2 1 -8 2 8
64 -55 -48 68

-2 -5 7 2

-668

-0.341 0.108 -0.401 0.096
0.012 -0.162 0.102 -0.144
-0.117 -0.174 -0.243 -0.099
0.084 -0.132 0.213 -0.006
PS C:\Users\Tysyachnyj V\Desktop\NM>
```

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  {
10 private:
11     double **a;
12     int n, m;
13 public:
14     //
15     matrix () {
16         a = 0;
17         n = 0;
18         m = 0;
19     }
20
21     // NxM, E, ,
22     matrix (int N, int M, bool E = 0) {
23         n = N;
24         m = M;
25         a = new double *[n];
26         for (int i = 0; i < n; ++ i) {
27             a[i] = new double[m];
28             for (int j = 0; j < m; ++ j) {
29                 a[i][j] = (i == j) * E;
30             }
31         }
32     }
33
34     //
35     int get_n_rows() {
36         return n;
37     }
38     int get_n_cols() {
39         return m;
40     }
41
42     double* operator [] (int index) {
43         return getRow (index);
44     }
45
46
47     //
```

```

48 double* getRow(int index){
49     if (index >= 0 && index < n){
50         return a[index];
51     }
52     return 0;
53 }
54
55 //
56 double* getColumn(int index){
57     if (index < 0 || index >= m){
58         return 0;
59     }
60     double * c = new double [n];
61     for (int i = 0; i < n; ++ i){
62         c[i] = a[i][index];
63     }
64     return c;
65 }
66
67 //
68 void swapRows (int index1, int index2){
69     if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
70         return ;
71     }
72     for (int i = 0; i < m; ++ i){
73         swap (a[index1][i], a[index2][i]);
74     }
75 }
76 };
77
78 //
79 matrix scanMatrix(int n, int m){
80     matrix a = matrix (n, m);
81     for (int i = 0; i < n; ++ i){
82         for (int j = 0; j < m; ++ j){
83             scanf ("%lf", & a[i][j]);
84         }
85     }
86     return a;
87 }
88
89 //
90 void printMatrix (matrix & a){
91     for (int i = 0; i < a.get_n_rows (); ++ i){
92         for (int j = 0; j < a.get_n_cols (); ++ j){
93             printf ("%5.3lf ", a[i][j]);
94         }
95         puts ("");
96     }

```

```

97 }
98
99 //
100 matrix mul (matrix & a, double k){
101     matrix c = matrix (a.get_n_rows (), a.get_n_cols ());
102     for (int i = 0; i < a.get_n_rows (); ++ i){
103         for (int j = 0; j < a.get_n_cols (); ++ j){
104             c[i][j] = a[i][j] * k;
105         }
106     }
107     return c;
108 }
109
110 //
111 matrix mul (matrix & a, matrix & b){
112     if (a.get_n_cols () != b.get_n_rows ()){
113         throw "Error";
114     }
115     matrix c = matrix (a.get_n_rows (), b.get_n_cols ());
116     for (int i = 0; i < a.get_n_rows (); ++ i){
117         for (int j = 0; j < b.get_n_cols (); ++ j){
118             for (int k = 0; k < a.get_n_cols (); ++ k){
119                 c[i][j] += a[i][k] * b[k][j];
120             }
121         }
122     }
123     return c;
124 }
125
126
127
128 /*
129 -5 -6 4 -2 0 3 -4 -6 2 4 -4 2 1 -8 2 8
130 64 -55 -48 68
131 */
132
133 int main()
134 {
135     int n = 4;
136     matrix A = matrix (n, n);
137     matrix U = matrix (n, n);
138
139     matrix B = matrix (n, 1);
140     A, U = scanMatrix(n, n);
141     B = scanMatrix(n, 1);
142
143     matrix M[n - 1];
144     matrix L = matrix(n, n, 1);
145     int p = 0;

```

```

146     double det = 1;
147
148     // LU-
149     for (int k = 0; k < n - 1; ++k){
150         M[k] = matrix(n, n, 1);
151         for (int i = k + 1; i < n; ++i){
152             if (U[k][k] == 0){
153                 int j = k + 1;
154                 while (U[j][j] == 0 and j < n){
155                     j += 1;
156                 }
157                 if (j == n){
158                     break;
159                 }
160                 U.swapRows(k, j);
161                 B.swapRows(k, j);
162                 p += 1;
163             }
164             M[k][i][k] = U[i][k] / U[k][k];
165             for (int j = k; j < n; ++j){
166                 U[i][j] -= M[k][i][k] * U[k][j];
167             }
168         }
169         det *= U[k][k];
170         L = mul(L, M[k]);
171     }
172     det *= pow(-1, p) * U[n - 1][n - 1];
173     cout << "\n";
174
175     //
176     double Z[n];
177     for (int i = 0; i < n; ++i){
178         double s = 0;
179         for (int j = 0; j < i; ++j){
180             s += L[i][j] * Z[j];
181         }
182         Z[i] = B[i][0] - s;
183     }
184
185     //
186     double X[n];
187     for (int i = n - 1; i > -1; --i){
188         double s = 0;
189         for (int j = i + 1; j < n; ++j){
190             s += U[i][j] * X[j];
191         }
192         X[i] = (Z[i] - s) / U[i][i];
193     }
194

```

```

195     for (int i = 0; i < n; ++i){
196         cout << X[i] << " ";
197     }
198     cout << "\n\n";
199
200     //
201     cout << det << "\n\n";
202
203     //
204     matrix A_inv = matrix(n, n);
205     matrix E = matrix(n, n, 1);
206     for (int k = 0; k < n; ++k){
207         for (int i = 0; i < n; ++i){
208             double s = 0;
209             for (int j = 0; j < i; ++j){
210                 s += L[i][j] * Z[j];
211             }
212             Z[i] = E[i][k] - s;
213         }
214
215         for (int i = n - 1; i > -1; --i){
216             double s = 0;
217             for (int j = i + 1; j < n; ++j){
218                 s += U[i][j] * A_inv[j][k];
219             }
220             A_inv[i][k] = (Z[i] - s) / U[i][i];
221         }
222     }
223     printMatrix(A_inv);
224 }

```

1.2 Метод прогонки

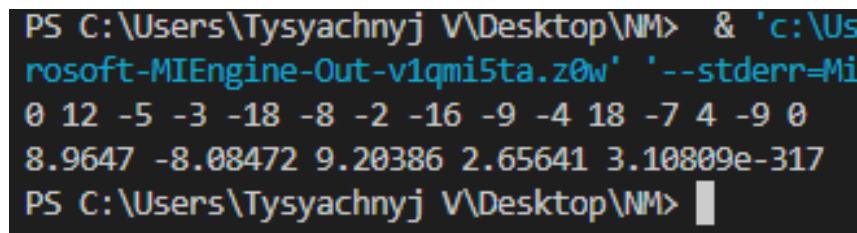
4 Постановка задачи

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: 25

$$\begin{cases} 12x_1 - 5x_2 = 148 \\ -3x_1 - 18x_2 - 8x_3 = 45 \\ -2x_2 - 16x_3 - 9x_4 = -155 \\ -4x_3 + 18x_4 - 7x_5 = 11 \\ 4x_4 - 9x_5 = 3 \end{cases}$$

5 Результаты работы



```
PS C:\Users\Tsyachnyj V\Desktop\NM> & 'c:\Usrosoft-MIEngine-Out-v1qmi5ta.z0w' '--stderr=Mi
0 12 -5 -3 -18 -8 -2 -16 -9 -4 18 -7 4 -9 0
8.9647 -8.08472 9.20386 2.65641 3.10809e-317
PS C:\Users\Tsyachnyj V\Desktop\NM>
```

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  {
10 private:
11     double **a;
12     int n, m;
13 public:
14     //
15     matrix (){
16         a = 0;
17         n = 0;
18         m = 0;
19     }
20
21     // NxM, E, ,
22     matrix (int N, int M, bool E = 0){
23         n = N;
24         m = M;
25         a = new double *[n];
26         for (int i = 0; i < n; ++ i){
27             a[i] = new double[m];
28             for (int j = 0; j < m; ++ j){
29                 a[i][j] = (i == j) * E;
30             }
31         }
32     }
33
34     //
35     int get_n_rows(){
36         return n;
37     }
38     int get_n_cols(){
39         return m;
40     }
41
42     double* operator [] (int index){
43         return getRow (index);
44     }
45
46
47     //
```

```

48 double* getRow(int index){
49     if (index >= 0 && index < n){
50         return a[index];
51     }
52     return 0;
53 }
54
55 //
56 double* getColumn(int index){
57     if (index < 0 || index >= m){
58         return 0;
59     }
60     double * c = new double [n];
61     for (int i = 0; i < n; ++ i){
62         c[i] = a[i][index];
63     }
64     return c;
65 }
66
67 //
68 void swapRows (int index1, int index2){
69     if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
70         return ;
71     }
72     for (int i = 0; i < m; ++ i){
73         swap (a[index1][i], a[index2][i]);
74     }
75 }
76 };
77
78 //
79 matrix scanMatrix(int n, int m){
80     matrix a = matrix (n, m);
81     for (int i = 0; i < n; ++ i){
82         for (int j = 0; j < m; ++ j){
83             scanf ("%lf", & a[i][j]);
84         }
85     }
86     return a;
87 }
88
89 //
90 void printMatrix (matrix & a){
91     for (int i = 0; i < a.get_n_rows (); ++ i){
92         for (int j = 0; j < a.get_n_cols (); ++ j){
93             printf ("%5.3lf ", a[i][j]);
94         }
95         puts ("");
96     }

```

```

97 | }
98 |
99 |
100 |
101 | /*
102 | 0 12 -5 -3 -18 -8 -2 -16 -9 -4 18 -7 4 -9 0
103 | */
104 |
105 | int main()
106 | {
107 |
108 |     int n = 5;
109 |     matrix A = matrix (n, 3);
110 |     double B[n] = {148, 45, -155, 11, 3};
111 |     A = scanMatrix(n, 3);
112 |
113 |     //
114 |     double P[n], Q[n], X[n];
115 |     P[0] = -A[0][2] / A[0][1];
116 |     Q[0] = B[0] / A[0][1];
117 |     for (int i = 1; i < n; ++i){
118 |         P[i] = -A[i][2] / (A[i][1] + A[i][0] * P[i - 1]);
119 |         Q[i] = (B[i] - A[i][0] * Q[i - 1]) / (A[i][1] + A[i][0] * P[i - 1]);
120 |     }
121 |
122 |     //
123 |     X[-1] = Q[-1];
124 |     for (int i = n - 2; i > -1; --i){
125 |         X[i] = P[i] * X[i + 1] + Q[i];
126 |     }
127 |     for (int i = 0; i < n; ++i){
128 |         cout << X[i] << " ";
129 |     }
130 | }

```

1.3 Метод простых итераций. Метод Зейделя

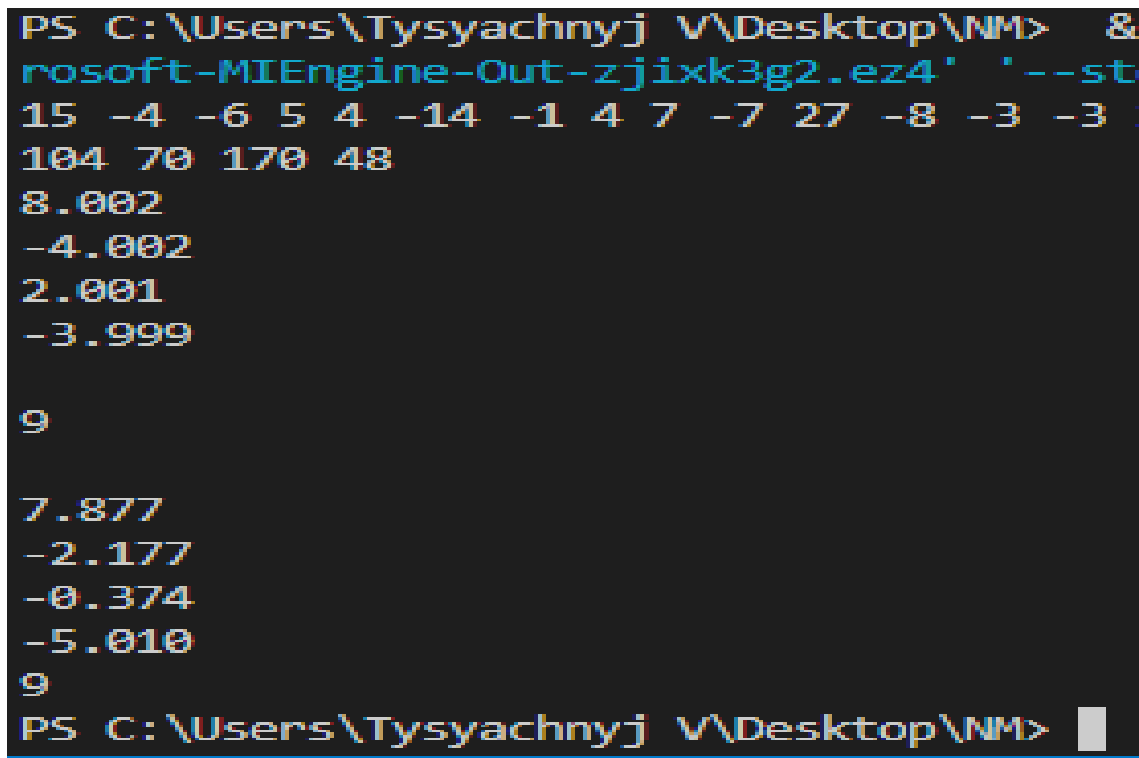
7 Постановка задачи

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: 25

$$\begin{cases} 15x_1 - 4x_2 - 6x_3 + 5x_4 = 104 \\ 4x_1 - 14x_2 - x_3 + 4x_4 = 70 \\ 7x_1 - 7x_2 + 27x_3 - 8x_4 = 170 \\ -3x_1 - 3x_2 + 2x_3 - 14x_4 = 48 \end{cases}$$

8 Результаты работы



```
PS C:\Users\Tsyachnyj V\Desktop\NM> &
rossoft-MIEngine-Out-zjixk3g2.ez4' '--st
15 -4 -6 5 4 -14 -1 4 7 -7 27 -8 -3 -3 :
104 70 170 48
8.002
-4.002
2.001
-3.999

9

7.877
-2.177
-0.374
-5.010
9
PS C:\Users\Tsyachnyj V\Desktop\NM>
```

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  {
10 private:
11     double **a;
12     int n, m;
13 public:
14     //
15     matrix () {
16         a = 0;
17         n = 0;
18         m = 0;
19     }
20
21     // NxM, E, ,
22     matrix (int N, int M, bool E = 0) {
23         n = N;
24         m = M;
25         a = new double *[n];
26         for (int i = 0; i < n; ++ i) {
27             a[i] = new double[m];
28             for (int j = 0; j < m; ++ j) {
29                 a[i][j] = (i == j) * E;
30             }
31         }
32     }
33
34     //
35     int get_n_rows() {
36         return n;
37     }
38     int get_n_cols() {
39         return m;
40     }
41
42     double* operator [] (int index) {
43         return getRow (index);
44     }
45
46
47     //
```

```

48 double* getRow(int index){
49     if (index >= 0 && index < n){
50         return a[index];
51     }
52     return 0;
53 }
54
55 //
56 double* getColumn(int index){
57     if (index < 0 || index >= m){
58         return 0;
59     }
60     double * c = new double [n];
61     for (int i = 0; i < n; ++ i){
62         c[i] = a[i][index];
63     }
64     return c;
65 }
66
67 //
68 void swapRows (int index1, int index2){
69     if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
70         return ;
71     }
72     for (int i = 0; i < m; ++ i){
73         swap (a[index1][i], a[index2][i]);
74     }
75 }
76 };
77
78 //
79 matrix scanMatrix(int n, int m){
80     matrix a = matrix (n, m);
81     for (int i = 0; i < n; ++ i){
82         for (int j = 0; j < m; ++ j){
83             scanf ("%lf", & a[i][j]);
84         }
85     }
86     return a;
87 }
88
89 //
90 void printMatrix (matrix & a){
91     for (int i = 0; i < a.get_n_rows (); ++ i){
92         for (int j = 0; j < a.get_n_cols (); ++ j){
93             printf ("%5.3lf ", a[i][j]);
94         }
95         puts ("");
96     }

```

```

97 }
98
99 //
100 matrix mul (matrix & a, double k){
101     matrix c = matrix (a.get_n_rows (), a.get_n_cols ());
102     for (int i = 0; i < a.get_n_rows (); ++ i){
103         for (int j = 0; j < a.get_n_cols (); ++ j){
104             c[i][j] = a[i][j] * k;
105         }
106     }
107     return c;
108 }
109
110 //
111 matrix mul (matrix & a, matrix & b){
112     if (a.get_n_cols () != b.get_n_rows ()){
113         throw "Error";
114     }
115     matrix c = matrix (a.get_n_rows (), b.get_n_cols ());
116     for (int i = 0; i < a.get_n_rows (); ++ i){
117         for (int j = 0; j < b.get_n_cols (); ++ j){
118             for (int k = 0; k < a.get_n_cols (); ++ k){
119                 c[i][j] += a[i][k] * b[k][j];
120             }
121         }
122     }
123     return c;
124 }
125
126 matrix operator+ (matrix A, matrix B){
127     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
128         throw "Matrix size not matched";
129     }
130     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
131     for (int i = 0; i < A.get_n_rows(); ++i){
132         for (int j = 0; j < A.get_n_cols(); ++j){
133             C[i][j] = A[i][j] + B[i][j];
134         }
135     }
136     return C;
137 }
138
139 matrix operator- (matrix A, matrix B){
140     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
141         throw "Matrix size not matched";
142     }
143     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
144     for (int i = 0; i < A.get_n_rows(); ++i){
145         for (int j = 0; j < A.get_n_cols(); ++j){

```

```

146         C[i][j] = A[i][j] - B[i][j];
147     }
148 }
149 return C;
150 }
151
152 //
153 matrix transpose(matrix A){
154     int n = A.get_n_rows();
155     int m = A.get_n_cols();
156     matrix C = matrix (m, n);
157     for (int i = 0; i < n; ++ i){
158         for (int j = 0; j < m; ++ j){
159             C[j][i] = A[i][j];
160         }
161     }
162     return C;
163 }
164
165 //
166 matrix minor(matrix A, int a, int b){
167     int n = A.get_n_rows();
168     int m = A.get_n_cols();
169     matrix C = matrix (n - 1, m - 1);
170     int k = 0;
171     for (int i = 0; i < n; ++i){
172         if (i > a){
173             k = 1;
174         }
175         for (int j = 0; j < m; ++j){
176             if (i != a){
177                 if (j < b){
178                     C[i - k][j] = A[i][j];
179                 }
180                 else if (j > b){
181                     C[i - k][j - 1] = A[i][j];
182                 }
183             }
184         }
185     }
186     return C;
187 }
188
189 //
190 double det(matrix A){
191     double d = 0;
192     int n = A.get_n_rows();
193     if (n == 1){
194         return A[0][0];

```



```

195     }
196     else if (n == 2){
197         return A[0][0] * A[1][1] - A[0][1] * A[1][0];
198     }
199     else {
200         for (int k = 0; k < n; ++k) {
201             matrix M = matrix(n - 1, n - 1);
202             for (int i = 1; i < n; ++i) {
203                 int t = 0;
204                 for (int j = 0; j < n; ++j) {
205                     if (j == k)
206                         continue;
207                     M[i-1][t] = A[i][j];
208                     t += 1;
209                 }
210             }
211             d += pow(-1, k + 2) * A[0][k] * det(M);
212         }
213     }
214     return d;
215 }
216
217 //
218 matrix inv(matrix A){
219     double d = det(A);
220     int n = A.get_n_rows();
221     matrix inv_A = matrix(n, n);
222     for(int i = 0; i < n; ++i){
223         for(int j = 0; j < n; ++j){
224             matrix M = matrix(n - 1, n - 1);
225             M = minor(A, i, j);
226             inv_A[i][j] = pow(-1.0, i + j + 2) * det(M) / d;
227         }
228     }
229     inv_A = transpose(inv_A);
230     return inv_A;
231 }
232
233 //
234 double Norm(matrix A){
235     double norm = 0;
236     for (int i = 0; i < A.get_n_rows(); ++i){
237         for (int j = 0; j < A.get_n_cols(); ++j){
238             norm += pow(A[i][j], 2);
239         }
240     }
241     return pow(norm, 0.5);
242 }
243

```

```

244
245
246 /*
247 15 -4 -6 5 4 -14 -1 4 7 -7 27 -8 -3 -3 2 -14
248 104 70 170 48
249 */
250
251 int main()
252 {
253
254     int n = 4;
255     matrix A = matrix (n, n);
256     matrix Alpha = matrix (n, n);
257     matrix Down = matrix (n, n);
258     matrix E = matrix (n, n, 1);
259     matrix B = matrix (n, 1);
260     matrix Beta = matrix(n, 1);
261     A = scanMatrix(n, n);
262     B = scanMatrix(n, 1);
263
264     //
265     for (int i = 0; i < n; ++i){
266         for (int j = 0; j < n; ++j){
267             double s = 0;
268             if (i != j){
269                 Alpha[i][j] = -A[i][j] / A[i][i];
270                 if (i > j){
271                     Down[i][j] = Alpha[i][j];
272                 }
273                 s += abs(A[i][j]);
274             }
275             else{
276                 Alpha[i][j] = 0;
277             }
278         }
279         Beta[i][0] = B[i][0] / A[i][i];
280     }
281
282     //
283     matrix X[2] = {Beta, Beta + mul(Alpha, Beta)};
284     int k = 1;
285     double eps = 0.01;
286     while (Norm(X[1] - X[0]) > eps){
287         X[0] = X[1];
288         X[1] = Beta + mul(Alpha, X[1]);
289         k += 1;
290     }
291     printMatrix(X[1]);
292     cout << "\n";

```

```

293 |     cout << k << "\n";
294 |
295 |     //
296 |     k = 1;
297 |     matrix C = E - Down;
298 |     C = inv(C);
299 |     C = mul(C, Beta);
300 |     X[0] = Beta;
301 |     X[1] = mul(Alpha, Beta) + C;
302 |     while (Norm(X[1] - X[0]) > eps){
303 |         X[0] = X[1];
304 |         X[1] = mul(Alpha, X[1]) + C;
305 |         k += 1;
306 |     }
307 |
308 |     cout << "\n";
309 |     printMatrix(X[1]);
310 |     cout << k << "\n";
311 | }

```

1.4 Метод вращений

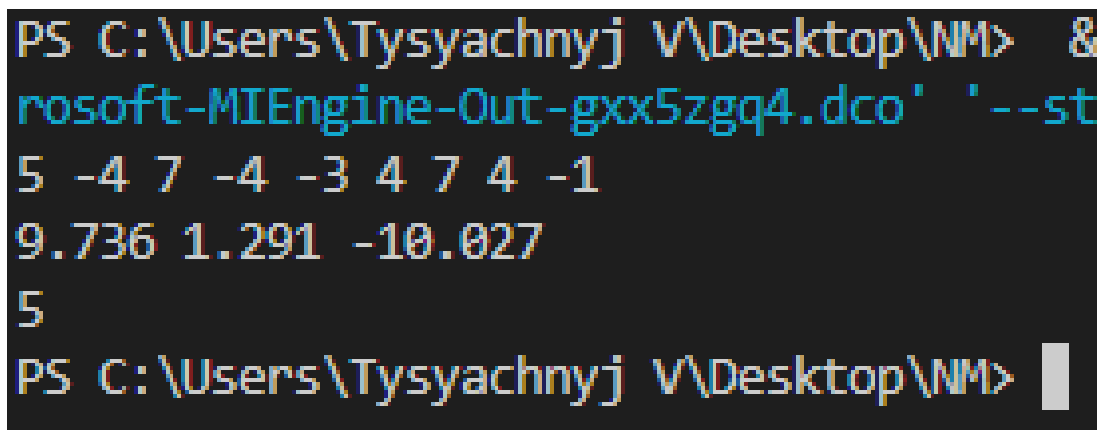
10 Постановка задачи

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: 25

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

11 Результаты работы



```
PS C:\Users\Tsyachnyj V\Desktop\NM> &
rossoft-MIEngine-Out-gxx5zgq4.dco' '--st
5 -4 7 -4 -3 4 7 4 -1
9.736 1.291 -10.027
5
PS C:\Users\Tsyachnyj V\Desktop\NM>
```

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 | {
10 | private:
11 |     double **a;
12 |     int n, m;
13 | public:
14 |     //
15 |     matrix () {
16 |         a = 0;
17 |         n = 0;
18 |         m = 0;
19 |     }
20 |
21 |     // NxM, E, ,
22 |     matrix (int N, int M, bool E = 0) {
23 |         n = N;
24 |         m = M;
25 |         a = new double *[n];
26 |         for (int i = 0; i < n; ++ i) {
27 |             a[i] = new double [m];
28 |             for (int j = 0; j < m; ++ j) {
29 |                 a[i][j] = (i == j) * E;
30 |             }
31 |         }
32 |     }
33 |
34 |     //
35 |     int get_n_rows() {
36 |         return n;
37 |     }
38 |     int get_n_cols() {
39 |         return m;
40 |     }
41 |
42 |     double* operator [] (int index) {
43 |         return getRow (index);
44 |     }
45 |
46 |
47 |     //
```

```

48 double* getRow(int index){
49     if (index >= 0 && index < n){
50         return a[index];
51     }
52     return 0;
53 }
54
55 //
56 double* getColumn(int index){
57     if (index < 0 || index >= m){
58         return 0;
59     }
60     double * c = new double [n];
61     for (int i = 0; i < n; ++ i){
62         c[i] = a[i][index];
63     }
64     return c;
65 }
66
67 //
68 void swapRows (int index1, int index2){
69     if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
70         return ;
71     }
72     for (int i = 0; i < m; ++ i){
73         swap (a[index1][i], a[index2][i]);
74     }
75 }
76 };
77
78 //
79 matrix scanMatrix(int n, int m){
80     matrix a = matrix (n, m);
81     for (int i = 0; i < n; ++ i){
82         for (int j = 0; j < m; ++ j){
83             scanf ("%lf", & a[i][j]);
84         }
85     }
86     return a;
87 }
88
89 //
90 void printMatrix (matrix & a){
91     for (int i = 0; i < a.get_n_rows (); ++ i){
92         for (int j = 0; j < a.get_n_cols (); ++ j){
93             printf ("%5.3lf ", a[i][j]);
94         }
95         puts ("");
96     }

```

```

97 }
98
99 //
100 matrix mul (matrix & a, double k){
101     matrix c = matrix (a.get_n_rows (), a.get_n_cols ());
102     for (int i = 0; i < a.get_n_rows (); ++ i){
103         for (int j = 0; j < a.get_n_cols (); ++ j){
104             c[i][j] = a[i][j] * k;
105         }
106     }
107     return c;
108 }
109
110 //
111 matrix mul (matrix & a, matrix & b){
112     if (a.get_n_cols () != b.get_n_rows ()){
113         throw "Error";
114     }
115     matrix c = matrix (a.get_n_rows (), b.get_n_cols ());
116     for (int i = 0; i < a.get_n_rows (); ++ i){
117         for (int j = 0; j < b.get_n_cols (); ++ j){
118             for (int k = 0; k < a.get_n_cols (); ++ k){
119                 c[i][j] += a[i][k] * b[k][j];
120             }
121         }
122     }
123     return c;
124 }
125
126 matrix operator+ (matrix A, matrix B){
127     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
128         throw "Matrix size not matched";
129     }
130     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
131     for (int i = 0; i < A.get_n_rows(); ++i){
132         for (int j = 0; j < A.get_n_cols(); ++j){
133             C[i][j] = A[i][j] + B[i][j];
134         }
135     }
136     return C;
137 }
138
139 matrix operator- (matrix A, matrix B){
140     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
141         throw "Matrix size not matched";
142     }
143     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
144     for (int i = 0; i < A.get_n_rows(); ++i){
145         for (int j = 0; j < A.get_n_cols(); ++j){

```

```

146         C[i][j] = A[i][j] - B[i][j];
147     }
148 }
149 return C;
150 }
151
152 //
153 matrix transpose(matrix A){
154     int n = A.get_n_rows();
155     int m = A.get_n_cols();
156     matrix C = matrix (m, n);
157     for (int i = 0; i < n; ++ i){
158         for (int j = 0; j < m; ++ j){
159             C[j][i] = A[i][j];
160         }
161     }
162     return C;
163 }
164
165 //
166 pair<int, int> find_max(matrix A){
167     int n = A.get_n_rows();
168     int i_max = 0;
169     int j_max = 1;
170     for (int i = 0; i < n - 1; ++i){
171         for (int j = i + 1; j < n; ++j){
172             if (abs(A[i][j]) > abs(A[i_max][j_max])){
173                 i_max = i;
174                 j_max = j;
175             }
176         }
177     }
178     return pair<int, int>(i_max, j_max);
179 }
180
181 //
182 double crit(matrix A){
183     double s = 0;
184     for (int i = 0; i < A.get_n_rows() - 1; ++i){
185         for (int j = i + 1; j < A.get_n_cols(); ++j){
186             s += pow(A[i][j], 2);
187         }
188     }
189     return pow(s, 0.5);
190 }
191
192
193
194 /*

```



```

195 5 -4 7 -4 -3 4 7 4 -1
196 */
197
198 int main()
199 {
200     int n = 3;
201     matrix A = matrix (n, n);
202     A = scanMatrix(n, n);
203
204     matrix X = matrix (n, n, 1);
205     double eps = 0.01;
206     int k = 0;
207
208     //
209     while (crit(A) > eps){
210         matrix U = matrix(n, n, 1);
211         matrix U_T = matrix(n, n, 1);
212         auto [i_max, j_max] = find_max(A);
213         double phi = atan(2 * A[i_max][j_max] / (A[i_max][i_max] - A[j_max][j_max])) / 2;
214         U[i_max][i_max] = cos(phi);
215         U[j_max][j_max] = cos(phi);
216         U[i_max][j_max] = -sin(phi);
217         U[j_max][i_max] = sin(phi);
218         U_T = transpose(U);
219         A = mul(U_T, A);
220         A = mul(A, U);
221         X = mul(X, U);
222         k += 1;
223     }
224
225     matrix Lambda = matrix(1, n);
226     for (int i = 0; i < n; ++i){
227         Lambda[0][i] = A[i][i];
228     }
229
230     printMatrix(Lambda);
231     cout << k << "\n";
232 }

```

1.5 QR – разложение матриц

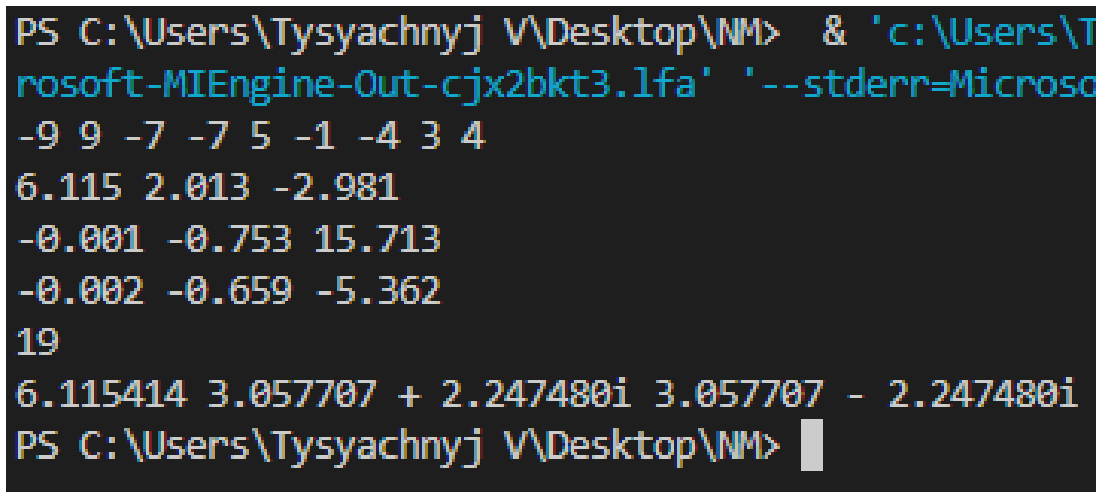
13 Постановка задачи

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: 25

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

14 Результаты работы



```
PS C:\Users\Tysyachnyj V\Desktop\NM> & 'c:\Users\Tysyachnyj V\Desktop\Microsoft-MIEngine-Out-cjx2bkt3.lfa' '--stderr=Microsoft-MIEngine-Out-cjx2bkt3.lfa'
-9 9 -7 -7 5 -1 -4 3 4
6.115 2.013 -2.981
-0.001 -0.753 15.713
-0.002 -0.659 -5.362
19
6.115414 3.057707 + 2.247480i 3.057707 - 2.247480i
PS C:\Users\Tysyachnyj V\Desktop\NM>
```

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  {
10 private:
11     double **a;
12     int n, m;
13 public:
14     //
15     matrix () {
16         a = 0;
17         n = 0;
18         m = 0;
19     }
20
21     // NxM, E, ,
22     matrix (int N, int M, bool E = 0) {
23         n = N;
24         m = M;
25         a = new double *[n];
26         for (int i = 0; i < n; ++ i) {
27             a[i] = new double[m];
28             for (int j = 0; j < m; ++ j) {
29                 a[i][j] = (i == j) * E;
30             }
31         }
32     }
33
34     //
35     int get_n_rows() {
36         return n;
37     }
38     int get_n_cols() {
39         return m;
40     }
41
42     double* operator [] (int index) {
43         return getRow (index);
44     }
45
46
47     //
```

```

48     double* getRow(int index){
49         if (index >= 0 && index < n){
50             return a[index];
51         }
52         return 0;
53     }
54
55     //
56     double* getColumn(int index){
57         if (index < 0 || index >= m){
58             return 0;
59         }
60         double * c = new double [n];
61         for (int i = 0; i < n; ++ i){
62             c[i] = a[i][index];
63         }
64         return c;
65     }
66
67     //
68     void swapRows (int index1, int index2){
69         if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
70             return ;
71         }
72         for (int i = 0; i < m; ++ i){
73             swap (a[index1][i], a[index2][i]);
74         }
75     }
76 };
77
78 //
79 matrix scanMatrix(int n, int m){
80     matrix a = matrix (n, m);
81     for (int i = 0; i < n; ++ i){
82         for (int j = 0; j < m; ++ j){
83             scanf ("%lf", & a[i][j]);
84         }
85     }
86     return a;
87 }
88
89 //
90 void printMatrix (matrix & a){
91     for (int i = 0; i < a.get_n_rows (); ++ i){
92         for (int j = 0; j < a.get_n_cols (); ++ j){
93             printf ("%5.3lf ", a[i][j]);
94         }
95         puts ("");
96     }

```

```

97 }
98
99 //
100 matrix mul (matrix & a, double k){
101     matrix c = matrix (a.get_n_rows (), a.get_n_cols ());
102     for (int i = 0; i < a.get_n_rows (); ++ i){
103         for (int j = 0; j < a.get_n_cols (); ++ j){
104             c[i][j] = a[i][j] * k;
105         }
106     }
107     return c;
108 }
109
110 //
111 matrix mul (matrix & a, matrix & b){
112     if (a.get_n_cols () != b.get_n_rows ()){
113         throw "Error";
114     }
115     matrix c = matrix (a.get_n_rows (), b.get_n_cols ());
116     for (int i = 0; i < a.get_n_rows (); ++ i){
117         for (int j = 0; j < b.get_n_cols (); ++ j){
118             for (int k = 0; k < a.get_n_cols (); ++ k){
119                 c[i][j] += a[i][k] * b[k][j];
120             }
121         }
122     }
123     return c;
124 }
125
126 // ,
127 double scal(matrix &A, matrix &B){
128     if (A.get_n_cols () != B.get_n_rows () or A.get_n_rows () != 1 or B.get_n_cols ()
129         != 1){
130         throw "Error";
131     }
132     double c;
133     for (int i = 0; i < A.get_n_rows (); ++ i){
134         for (int j = 0; j < B.get_n_cols (); ++ j){
135             for (int k = 0; k < A.get_n_cols (); ++ k){
136                 c += A[i][k] * B[k][j];
137             }
138         }
139     }
140     return c;
141 }
142
143 matrix operator+ (matrix A, matrix B){
144     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
145         throw "Matrix size not matched";
146     }
147 }

```

```

145     }
146     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
147     for (int i = 0; i < A.get_n_rows(); ++i){
148         for (int j = 0; j < A.get_n_cols(); ++j){
149             C[i][j] = A[i][j] + B[i][j];
150         }
151     }
152     return C;
153 }
154
155 matrix operator- (matrix A, matrix B){
156     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
157         throw "Matrix size not matched";
158     }
159     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
160     for (int i = 0; i < A.get_n_rows(); ++i){
161         for (int j = 0; j < A.get_n_cols(); ++j){
162             C[i][j] = A[i][j] - B[i][j];
163         }
164     }
165     return C;
166 }
167
168 //
169 matrix transpose(matrix A){
170     int n = A.get_n_rows();
171     int m = A.get_n_cols();
172     matrix C = matrix (m, n);
173     for (int i = 0; i < n; ++ i){
174         for (int j = 0; j < m; ++ j){
175             C[j][i] = A[i][j];
176         }
177     }
178     return C;
179 }
180
181 //
182 double sign(double n){
183     if (n < 0){
184         return -1;
185     }
186     else if (n == 0){
187         return 0;
188     }
189     else{
190         return 1;
191     }
192 }
193

```

```

194 // QR-
195 pair<matrix, matrix> QR_decomposition(matrix A){
196     int n = A.get_n_rows();
197     matrix E = matrix(n, n, 1);
198     matrix Q = matrix(n, n, 1);
199     matrix H = matrix(n, n);
200     for (int j = 0; j < n - 1; ++j){
201         matrix v = matrix(n, 1);
202         for (int i = j; i < n; ++i){
203             v[i][0] = A[i][j];
204             if (i == j){
205                 double s = 0;
206                 for (int k = j; k < n; ++k){
207                     s += pow(A[k][j], 2);
208                 }
209                 v[i][0] += sign(A[i][j]) * pow(s, 0.5);
210             }
211         }
212         matrix v_T = transpose(v);
213         matrix v1 = mul(v, v_T);
214         H = mul(v1, 2);
215         H = E - mul(H, 1 / scal(v_T, v));
216         Q = mul(Q, H);
217         A = mul(H, A);
218     }
219     return pair<matrix, matrix> (Q, A);
220 }
221
222 //
223 pair<double, double> complex(double b, double c){
224     if (pow(b, 2) - 4 * c < 0){
225         return pair<double, double> (-b / 2, pow(- pow(b, 2) + 4 * c, 0.5) / 2);
226     }
227     else{
228         return pair<double, double> (-b / 2 + pow(pow(b, 2) - 4 * c, 0.5) / 2, 0);
229     }
230 }
231
232 //
233 bool check(matrix A[2], double eps){
234     int n = A[1].get_n_rows();
235     int j = 0;
236     while (j < n - 1){
237         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]);
238         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]);
239         if (z1.second == 0){
240             double s = 0;

```

```

241         for (int i = j + 1; i < n; ++i){
242             s += pow(A[1][i][j], 2);
243         }
244         s = pow(s, 0.5);
245         if (s > eps){
246             return true;
247         }
248     }
249     else{
250         if (pow(pow(z1.first - z0.first, 2) + pow(z1.second - z0.second, 2), 0.5) >
251             eps){
252             return true;
253         }
254         else{
255             j += 1;
256         }
257     }
258     j += 1;
259     return false;
260 }
261
262
263
264
265
266 /*
267 -9 9 -7 -7 5 -1 -4 3 4
268 */
269
270 int main()
271 {
272     int n = 3;
273     matrix A = matrix (n, n);
274     A = scanMatrix(n, n);
275
276     // QR-
277     double eps = 0.01;
278     matrix AA[2] = {A, matrix(n, n)};
279     auto [Q, R] = QR_decomposition(A);
280     AA[1] = mul(R, Q);
281     int k = 1;
282
283     while (check(AA, eps)){
284         auto [Q, R] = QR_decomposition(AA[1]);
285         AA[0] = AA[1];
286         AA[1] = mul(R, Q);
287         k += 1;
288     }

```



```

289 //
290 string eigenvalue[n];
291 int i = 0;
292 while (i < n - 1){
293     pair<double, double> z = complex(AA[1][i][i] + AA[1][i + 1][i + 1], AA[1][i][i]
294         * AA[1][i + 1][i + 1] - AA[1][i][i + 1] * AA[1][i + 1][i]);
295     if (z.second == 0){
296         eigenvalue[i] = to_string(AA[1][i][i]);
297     }
298     else{
299         eigenvalue[i] = to_string(z.first) + " + " + to_string(z.second) + "i";
300         i += 1;
301         eigenvalue[i] = to_string(z.first) + " - " + to_string(z.second) + "i";
302         i += 1;
303     }
304     i += 1;
305 }
306
307 printMatrix(AA[1]);
308 cout << k << "\n";
309 for (int i = 0; i < n; ++i){
310     cout << eigenvalue[i] << " ";
311 }
312 }

```