

**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: Голошумов М.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: 4

$$\begin{cases} -x_1 - 7x_2 - 3x_3 - 2x_4 = -12 \\ -8x_1 + x_2 - 9x_3 = -60 \\ 8x_1 + 2x_2 - 5x_3 - 3x_4 = -91 \\ -5x_1 + 3x_2 + 5x_3 - 9x_4 = -43 \end{cases}$$

2 Результаты работы

```
Консоль отладки Microsoft Visual Studio
Solve of equation
[-2.00000]
[-4.00000]
[8.00000]
[9.00000]
size: 4 x 1
det(A) = -10339.0000
Inverse matrix A(-1)
[0.00203 -0.04807 0.06364 -0.02167]
[-0.12216 0.03395 0.02950 0.01731]
[-0.01538 -0.06461 -0.05329 0.02118]
[-0.05039 0.00213 -0.05513 -0.08154]
size: 4 x 4

C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab1\goloshumov\Lab1\task_1.1\x64\Debug\lab1-1.exe (процесс 10608) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "read.h"
5
6  static inline double absolute(const double a) {
7      return a > 0 ? a : -a;
8  }
9
10 //LU-
11 // L , U , P
12 Matrix** LU_decomposition(Matrix* matrix) {
13     int size = matrix->height, max;
14     Matrix** LUP = (Matrix**)malloc(sizeof(Matrix*) * 3);
15     // L P U
16     for (int i = 0; i < 3; i++) {
17         LUP[i] = create_matrix();
18         resize_matrix(LUP[i], size, size);
19     }
20
21     for (int i = 0; i < size; i++)
22         LUP[2]->data[i][i] = LUP[0]->data[i][i] = 1;
23
24     for (int i = 0; i < size; i++)
25         for (int j = 0; j < size; j++)
26             LUP[1]->data[i][j] = matrix->data[i][j];
27
28     // LU
29     for (int j = 0; j < size; j++) {
30         max = j;
31         for (int i = j + 1; i < size; i++)
32             if (absolute(LUP[1]->data[i][j]) > absolute(LUP[1]->data[max][j]))
33                 max = i;
34         matrix_exchange_strings(LUP[1], j, max);
35         matrix_exchange_strings(LUP[2], j, max);
36         matrix_exchange_strings(LUP[0], j, max);
37         matrix_exchange_columns(LUP[0], j, max);
38         for (int i = j + 1; i < size; i++) {
39             LUP[0]->data[i][j] = LUP[1]->data[i][j] / LUP[1]->data[j][j];
40             for (int k = j; k < size; k++)
41                 LUP[1]->data[i][k] -= LUP[0]->data[i][j] * LUP[1]->data[j][k];
42         }
43     }
44     return LUP;
45 }
46
47 // LU
```

```

48 Matrix* LU_solve(Matrix** LUP, Matrix* vector) {
49     Matrix* result;
50     int i, j;
51     if (!LUP)
52         return NULL;
53     result = multiplication(LUP[2], vector);
54     for (i = 0; i < result->height; i++)
55         for (j = 0; j < i; j++)
56             result->data[i][0] -= result->data[j][0] * LUP[0]->data[i][j];
57     for (i = result->height - 1; i >= 0; i--) {
58         for (j = result->height - 1; j > i; j--)
59             result->data[i][0] -= result->data[j][0] * LUP[1]->data[i][j];
60         result->data[i][0] /= LUP[1]->data[i][i];
61     }
62     return result;
63 }
64
65 // , LU
66 Matrix* gauss_method(Matrix* matrix, Matrix* vector) {
67     Matrix** LUP, * result;
68     int i;
69     LUP = LU_decomposition(matrix);
70     result = LU_solve(LUP, vector);
71     for (i = 0; i < 3; i++) {
72         remove_matrix(LUP[i]);
73         free(LUP[i]);
74     }
75     free(LUP);
76     return result;
77 }
78
79 // LU .
80 double determinant(Matrix* matrix) {
81     Matrix** LUP = LU_decomposition(matrix);
82     int i, j = 0;
83     double det;
84     if (!LUP)
85         return 0;
86     for (i = 0; i < LUP[2]->height; i++)
87         if (!LUP[2]->data[i][i])
88             j++;
89     det = (j % 2 == 0) ? 1 : -1;
90     for (i = 0; i < LUP[1]->height; i++)
91         det *= LUP[1]->data[i][i];
92     return det;
93 }
94
95 //
96 Matrix* matrix_inversion(Matrix* matrix) {

```

```

97     Matrix** LUP = LU_decomposition(matrix), * inverse, * right_part, * temp;
98     int i, j;
99     if (!LUP)
100         return NULL;
101     inverse = create_matrix();
102     resize_matrix(inverse, matrix->height, matrix->width);
103     right_part = create_matrix();
104     resize_matrix(right_part, matrix->height, 1);
105     right_part->data[0][0] = 1;
106     for (j = 0; j < inverse->width; j++) {
107         temp = LU_solve(LUP, right_part);
108         for (i = 0; i < inverse->height; i++)
109             inverse->data[i][j] = temp->data[i][0];
110         if (j != inverse->width - 1)
111             matrix_exchange_strings(right_part, j + 1, j);
112         remove_matrix(temp);
113         free(temp);
114     }
115     {
116         for (i = 0; i < 3; i++) {
117             remove_matrix(LUP[i]);
118             free(LUP[i]);
119         }
120         free(LUP);
121         remove_matrix(right_part);
122         free(right_part);
123     }
124     return inverse;
125 }
126
127 int main(void) {
128     int i;
129     Matrix* matrix = create_matrix(), * vector = create_matrix(), * result;
130     FILE* fmatrix, * fvector;
131
132     fmatrix = fopen("lab01-1matrix.txt", "r");
133     fvector = fopen("lab01-1vector.txt", "r");
134
135     if (fmatrix == NULL || fvector == NULL) {
136         fprintf(stderr, "Invalid name of file\n");
137         return 0;
138     }
139
140     scan_matrix(matrix, fmatrix);
141     fclose(fmatrix);
142     scan_matrix(vector, fvector);
143     fclose(fvector);
144
145     if (result = gauss_method(matrix, vector)) {

```

```

146     printf("Solve of equation\n");
147     print_matrix(result, stdout, "\n");
148     remove_matrix(result);
149     free(result);
150 }
151 if (matrix->height == matrix->width) {
152     printf("det(A) = %.4f\n", determinant(matrix));
153 }
154 if (result = matrix_inversion(matrix)) {
155     printf("Inverse matrix A(-1)\n");
156     print_matrix(result, stdout);
157     remove_matrix(result);
158     free(result);
159 }
160 remove_matrix(vector);
161 remove_matrix(matrix);
162 free(vector);
163 free(matrix);
164
165     return 0;
166 }

1 #include "read.h"
2 #include <stdlib.h>
3
4 //
5 Matrix* create_matrix(void) {
6     Matrix* new_matrix = (Matrix*)malloc(sizeof(Matrix));
7     new_matrix->width = new_matrix->height = 0;
8     new_matrix->data = NULL;
9     return new_matrix;
10 }
11
12 //
13 void remove_matrix(Matrix* matrix) {
14     if (!matrix)
15         return;
16     for (int i = 0; i < matrix->height; i++)
17         free(matrix->data[i]);
18     free(matrix->data);
19     matrix->data = NULL;
20     matrix->height = matrix->width = 0;
21 }
22
23 // - - 0
24 void resize_matrix(Matrix* matrix, const int height, const int width) {
25     if (height > 0) {
26         matrix->data = (double**) realloc (matrix->data, sizeof(double*) * height);
27         for (int i = matrix->height; i < height; i++) {
28             matrix->data[i] = (double*) malloc(sizeof(double) * (width <= 0 ? matrix->

```

```

29         width : width));
30     for (int j = 0; j < width; j++)
31         matrix->data[i][j] = 0;
32 }
33 if (width > 0)
34     for (int i = 0; i < matrix->height; i++) {
35         matrix->data[i] = (double*) realloc (matrix->data[i], sizeof(double) *
36             width);
37         for (int j = matrix->width; j < width; j++)
38             matrix->data[i][j] = 0;
39     }
40 if (width > 0)
41     matrix->width = width;
42 if (height > 0)
43     matrix->height = height;
44 }
45 //
46 void print_matrix(Matrix* matrix, FILE* stream) {
47     int i, j;
48     if (!matrix->data)
49         return;
50     for (i = 0; i < matrix->height; i++) {
51         fputc('[', stream);
52         for (j = 0; j < matrix->width; j++)
53             fprintf(stream, "%.5f ", matrix->data[i][j]);
54         fprintf(stream, "\b\b]\n");
55     }
56     fprintf(stream, "size: %d x %d\n", matrix->height, matrix->width);
57 }
58 //
59 void scan_matrix(Matrix* matrix, FILE* stream) {
60     int c = 0;
61     float a;
62     for (int i = 0; c != EOF; i++) {
63         resize_matrix(matrix, i + 1, -1);
64         c = 0;
65         for (int j = 0; c != '\n'; j++) {
66             if (!i)
67                 resize_matrix(matrix, -1, j + 1);
68             fscanf(stream, "%f", &a);
69             matrix->data[i][j] = a;
70             c = getc(stream);
71             if (c == EOF) {
72                 resize_matrix(matrix, i, -1);
73                 return;
74             }
75         }

```

```

76     }
77 }
78 }
79
80 //
81 Matrix* multiplication(Matrix* A, Matrix* B) {
82     Matrix* result = create_matrix();
83     int i, j, k;
84     if (A->width != B->height)
85         return NULL;
86     resize_matrix(result, A->height, B->width);
87     for (i = 0; i < result->height; i++)
88         for (j = 0; j < result->width; j++)
89             for (k = 0; k < A->width; k++)
90                 result->data[i][j] += A->data[i][k] * B->data[k][j];
91     return result;
92 }
93
94 //
95 void matrix_exchange_strings(Matrix* matrix, int i1, int i2) {
96     double temp;
97     for (int j = 0; j < matrix->width; j++) {
98         temp = matrix->data[i1][j];
99         matrix->data[i1][j] = matrix->data[i2][j];
100        matrix->data[i2][j] = temp;
101    }
102 }
103
104 //
105 void matrix_exchange_columns(Matrix* matrix, int j1, int j2) {
106     double temp;
107     for (int i = 0; i < matrix->width; i++) {
108         temp = matrix->data[i][j1];
109         matrix->data[i][j1] = matrix->data[i][j2];
110         matrix->data[i][j2] = temp;
111     }
112 }
113
114 //
115 Matrix* matrix_transposition(Matrix* matrix) {
116     Matrix* result;
117     int i, j;
118     if (!matrix)
119         return NULL;
120     result = create_matrix();
121     resize_matrix(result, matrix->width, matrix->height);
122     for (i = 0; i < result->height; i++)
123         for (j = 0; j < result->width; j++)
124             result->data[i][j] = matrix->data[j][i];

```



```

125 |     return result;
126 | }

1 | #ifndef _LAB1_
2 | #define _LAB1_
3 |
4 | #include <stdio.h>
5 |
6 | typedef struct Matrix {
7 |     double** data;
8 |     unsigned int width;
9 |     unsigned int height;
10 | } Matrix;
11 |
12 | Matrix* create_matrix(void);
13 | void remove_matrix(Matrix*);
14 | void resize_matrix(Matrix*, const int, const int);
15 | void print_matrix(Matrix*, FILE*);
16 | void scan_matrix(Matrix*, FILE*);
17 | Matrix* multiplication(Matrix*, Matrix*);
18 | void matrix_exchange_strings(Matrix*, int, int);
19 | void matrix_exchange_columns(Matrix*, int, int);
20 | Matrix* matrix_transposition(Matrix*);
21 |
22 | #endif

```

Коэффициенты перед иксами системы:

```

1 | -1 -7 -3 -2
2 | -8 1 -9 0
3 | 8 2 -5 -3
4 | -5 3 5 -9

```

Вектор b:

```

1 | -12
2 | -60
3 | -91
4 | -43

```

1.2 Метод прогонки

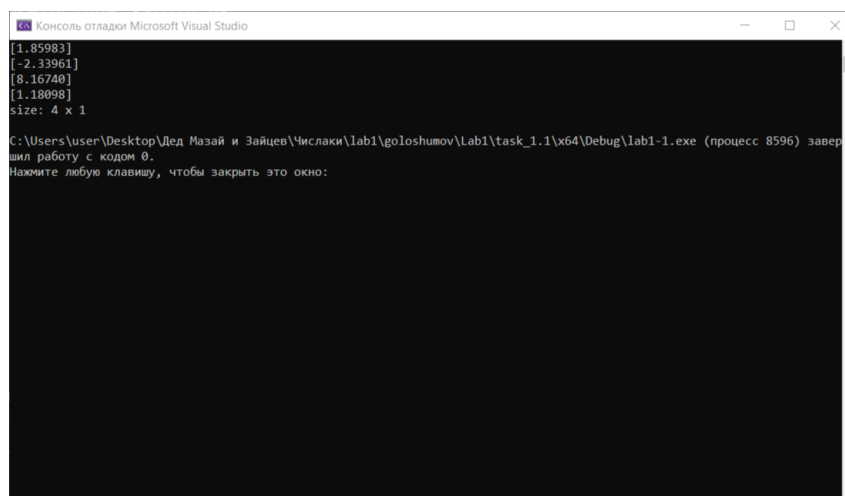
4 Постановка задачи

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: 4

$$\begin{cases} -14x_1 - 6x_2 = -78 \\ -9x_1 + 15x_2 - x_3 = -73 \\ x_2 - 11x_3 + x_4 = -38 \\ -7x_3 - 12x_4 + 3x_5 = 77 \\ 6x_4 - 7x_5 = 91 \end{cases}$$

5 Результаты работы



```
Консоль отладки Microsoft Visual Studio
[1.85983]
[-2.33961]
[8.16740]
[1.18098]
size: 4 x 1
C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab1\goloshumov\Lab1\task_1.1\x64\Debug\lab1-1.exe (процесс 8596) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 2: Вывод программы в консоли

6 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "read.h"
4
5  static inline double absolute(double a) {
6      return a > 0 ? a : -a;
7  }
8
9  //
10 Matrix* tridiagonal_matrix_algorithm(Matrix* matrix, Matrix* vector) {
11     Matrix* PQ, * result;
12     int i;
13     if (!matrix || !vector || matrix->width != 3 || matrix->height != vector->height ||
14         vector->width != 1) {
15         fprintf(stderr, "Matrix or vector have has incorrect size\n");
16         return NULL;
17     }
18     PQ = create_matrix();
19     resize_matrix(PQ, matrix->height - 1, 2);
20     result = create_matrix();
21     resize_matrix(result, matrix->height, 1);
22     //
23     PQ->data[0][0] = -matrix->data[0][2] / matrix->data[0][1];
24     PQ->data[0][1] = vector->data[0][0] / matrix->data[0][1];
25     if (absolute(matrix->data[0][1]) < absolute(matrix->data[0][2]) ||
26         absolute(matrix->data[matrix->height - 1][1]) < absolute(matrix->data[matrix->
27             height - 1][2])) {
28         fprintf(stderr, "Singular matrix\n");
29         return NULL;
30     }
31     //
32     for (i = 1; i < PQ->height; i++) {
33         if (absolute(matrix->data[i][1]) < absolute(matrix->data[i][0]) + absolute(
34             matrix->data[i][2])) {
35             fprintf(stderr, "Singular matrix\n");
36             return NULL;
37         }
38         double temp = matrix->data[i][0] * PQ->data[i - 1][0] + matrix->data[i][1];
39         PQ->data[i][0] = -matrix->data[i][2] / temp;
40         PQ->data[i][1] = (vector->data[i][0] - matrix->data[i][0] * PQ->data[i - 1][1])
41             / temp;
42     }
43     //
44     i = result->height - 1;
45     result->data[i][0] = (vector->data[i][0] - matrix->data[i][0] * PQ->data[i - 1][1])
46         /
47         (matrix->data[i][0] * PQ->data[i - 1][0] + matrix->data[i][1]);
```

```

43     for (i = result->height - 2; i >= 0; i--)
44         result->data[i][0] = PQ->data[i][0] * result->data[i + 1][0] + PQ->data[i][1];
45
46     remove_matrix(PQ);
47     free(PQ);
48     return result;
49 }
50 Matrix* (* const TDMA)(Matrix*, Matrix*) = tridiagonal_matrix_algorithm;
51
52 int main(void) {
53     int i;
54     Matrix* matrix = create_matrix(), * vector = create_matrix(), * result;
55     FILE* fmatrix, * fvector;
56
57     fmatrix = fopen("lab01-2matrix.txt", "r");
58     fvector = fopen("lab01-2vector.txt", "r");
59
60     if (!fmatrix || !fvector) {
61         fprintf(stderr, "Invalid name of file\n");
62         return 0;
63     }
64
65     scan_matrix(matrix, fmatrix);
66     fclose(fmatrix);
67     scan_matrix(vector, fvector);
68     fclose(fvector);
69     if (result = TDMA(matrix, vector)) {
70         print_matrix(result, stdout);
71         remove_matrix(result);
72         free(result);
73     }
74     remove_matrix(vector);
75     remove_matrix(matrix);
76     free(vector);
77     free(matrix);
78
79     return 0;
80 }

```

Коэффициенты перед искомыми системами:

```

1 || 0 -14 -6
2 || -9 15 -1
3 || 1 -11 1
4 || -7 12 3
5 || 6 -7 0

```

Вектор b:

```

1 || -78
2 || -73

```

$$\begin{array}{r|l} 3 & -38 \\ 4 & 77 \\ 5 & 91 \end{array}$$

1.3 Метод простых итераций. Метод Зейделя

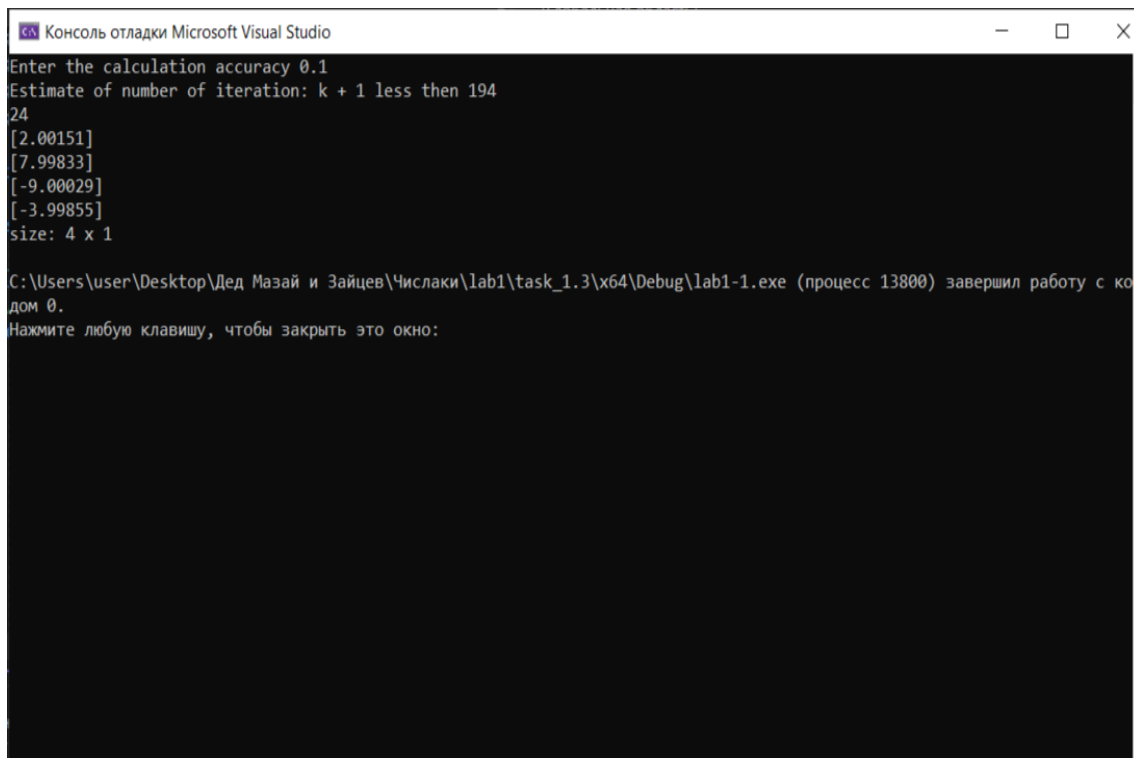
7 Постановка задачи

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: 4

$$\begin{cases} 26x_1 - 9x_2 - 8x_3 + 8x_4 = 20 \\ 9x_1 - 21x_2 - 2x_3 + 8x_4 = -164 \\ -3x_1 + 2x_2 - 18x_3 + 8x_4 = 140 \\ x_1 - 6x_2 - x_3 + 11x_4 = -81 \end{cases}$$

8 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Enter the calculation accuracy 0.1
Estimate of number of iteration: k + 1 less then 194
24
[2.00151]
[7.99833]
[-9.00029]
[-3.99855]
size: 4 x 1

C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab1\task_1.3\x64\Debug\lab1-1.exe (процесс 13800) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 3: Вывод программы в консоли

9 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "read.h"
5
6  int SEIDEL = 0;
7
8  static inline double absolute(const double a) {
9      return a > 0 ? a : -a;
10 }
11 // .
12 double matrix_norm(Matrix* matrix) {
13     double sum = 0, norm = 0;
14     int i, j;
15     if (!matrix)
16         return 0;
17     for (i = 0; i < matrix->height; i++) {
18         for (j = 0; j < matrix->width; j++)
19             sum += absolute(matrix->data[i][j]);
20     }
21     norm = sum > norm ? sum : norm;
22     sum = 0;
23     return norm;
24 }
25
26 //
27 int estimation_of_number_of_iteration(double eps, double a, double b) {
28     return (int)(log(eps * (1 - a) / b) / log(a));
29 }
30
31 //
32 double error(Matrix* vector_prev, Matrix* vector_cur, double alpha_norm) {
33     int i;
34     double epsilon_k;
35     Matrix* temp = create_matrix();
36     resize_matrix(temp, vector_prev->height, 1);
37     for (i = 0; i < vector_prev->height; i++)
38         temp->data[i][0] = vector_prev->data[i][0] - vector_cur->data[i][0];
39     if (alpha_norm < 1)
40         epsilon_k = matrix_norm(temp) * alpha_norm / (1 - alpha_norm);
41     else
42         epsilon_k = matrix_norm(temp);
43     remove_matrix(temp);
44     free(temp);
45     return epsilon_k;
46 }
47
```

```

48 //
49 Matrix* seidel_method(Matrix* alpha, Matrix* betta, double epsilon) {
50     Matrix* current, * previous;
51     double alpha_norm = matrix_norm(alpha), err = epsilon + 1;
52     int i, j, step;
53     if (alpha_norm >= 1) {
54         fprintf(stderr, "Norm of ALPHA is %.4f\n", alpha_norm);
55         return NULL;
56     }
57     current = create_matrix();
58     resize_matrix(current, betta->height, 1);
59     for (i = 0; i < betta->height; i++)
60         current->data[i][0] = betta->data[i][0];
61     previous = create_matrix();
62     resize_matrix(previous, betta->height, 1);
63     for (step = 1; err >= epsilon; step++) {
64         for (i = 0; i < betta->height; i++)
65             previous->data[i][0] = current->data[i][0];
66         for (i = 0; i < betta->height; i++) {
67             current->data[i][0] = betta->data[i][0];
68             for (j = 0; j < alpha->width; j++)
69                 current->data[i][0] += alpha->data[i][j] * (j < i ? current->data[j][0]
70                     : previous->data[j][0]);
71         }
72         err = error(previous, current, alpha_norm);
73     }
74     printf("%d\n", step);
75     remove_matrix(previous);
76     free(previous);
77     return current;
78 }
79 //
80 Matrix* jacobi_method(Matrix* alpha, Matrix* betta, double epsilon) {
81     Matrix* current, * previous;
82     double alpha_norm = matrix_norm(alpha), err = epsilon + 1;
83     int i, step;
84     if (alpha_norm > 1) {
85         fprintf(stderr, "Norm of ALPHA is %.4f\n", alpha_norm);
86         return NULL;
87     }
88     if (alpha_norm != 1)
89         printf("Estimate of number of iteration: k + 1 less than %d\n",
90             estimation_of_number_of_iteration(epsilon, alpha_norm, matrix_norm(betta)));
91     ;
92     current = create_matrix();
93     resize_matrix(current, betta->height, 1);
94     for (i = 0; i < betta->height; i++)
95         current->data[i][0] = betta->data[i][0];

```



```

95     previous = create_matrix();
96     resize_matrix(previous, betta->height, 1);
97     for (step = 1; err >= epsilon; step++) {
98         for (i = 0; i < betta->height; i++)
99             previous->data[i][0] = current->data[i][0];
100         remove_matrix(current);
101         free(current);
102         current = multiplication(alpha, previous);
103         for (i = 0; i < betta->height; i++)
104             current->data[i][0] += betta->data[i][0];
105         err = error(previous, current, alpha_norm);
106     }
107     printf("%d\n", step);
108     remove_matrix(previous);
109     free(previous);
110     return current;
111 }
112
113 //
114 Matrix* simple_iteration_method(Matrix* matrix, Matrix* vector, double epsilon) {
115     Matrix* alpha, * betta, * result;
116     double norm;
117     int i, j;
118     if (!matrix || !vector || matrix->width != matrix->height || matrix->height !=
119         vector->height || vector->width != 1) {
120         fprintf(stderr, "Matrix or vector have has the incorrect size\n");
121         return NULL;
122     }
123     for (i = 0; i < matrix->height; i++)
124         if (!matrix->data[i][i]) {
125             fprintf(stderr, "Singular matrix\n");
126             return NULL;
127         }
128     //
129     alpha = create_matrix();
130     resize_matrix(alpha, matrix->height, matrix->width);
131     betta = create_matrix();
132     resize_matrix(betta, vector->height, 1);
133     for (i = 0; i < matrix->height; i++) {
134         for (j = 0; j < matrix->width; j++)
135             if (i == j)
136                 alpha->data[i][j] = 0;
137             else
138                 alpha->data[i][j] = -matrix->data[i][j] / matrix->data[i][i];
139         betta->data[i][0] = vector->data[i][0] / matrix->data[i][i];
140     }
141     result = SEIDEL ? seidel_method(alpha, betta, epsilon) : jacobi_method(alpha, betta
        , epsilon);

```

```

142     remove_matrix(alpha);
143     remove_matrix(betta);
144     free(alpha);
145     free(betta);
146
147     return result;
148 }
149
150 int main(void) {
151     int i;
152     double epsilon;
153     Matrix* matrix = create_matrix(), * vector = create_matrix(), * result;
154     FILE* fmatrix, * fvector;
155
156     fmatrix = fopen("lab01-3matrix.txt", "r");
157     fvector = fopen("lab01-3vector.txt", "r");
158
159     printf("Enter the calculation accuracy ");
160     scanf("%lf", &epsilon);
161     if (epsilon <= 0) {
162         fprintf(stderr, "Negative value of error\n");
163         return 0;
164     }
165     if (!fmatrix || !fvector) {
166         fprintf(stderr, "Invalid name of file\n");
167         return 0;
168     }
169
170     scan_matrix(matrix, fmatrix);
171     fclose(fmatrix);
172     scan_matrix(vector, fvector);
173     fclose(fvector);
174     if (result = simple_iteration_method(matrix, vector, epsilon)) {
175         print_matrix(result, stdout);
176         remove_matrix(result);
177         free(result);
178     }
179     remove_matrix(vector);
180     remove_matrix(matrix);
181     free(vector);
182     free(matrix);
183     return 0;
184 }

```

Коэффициенты перед иксами системы:

```

1 | 26 -9 -8 8
2 | 9 -21 -2 8
3 | -3 2 -18 8
4 | 1 -6 -1 11

```

Вектор b :

$$\begin{array}{l|l} 1 & 20 \\ 2 & -164 \\ 3 & 140 \\ 4 & -81 \end{array}$$

1.4 Метод вращений

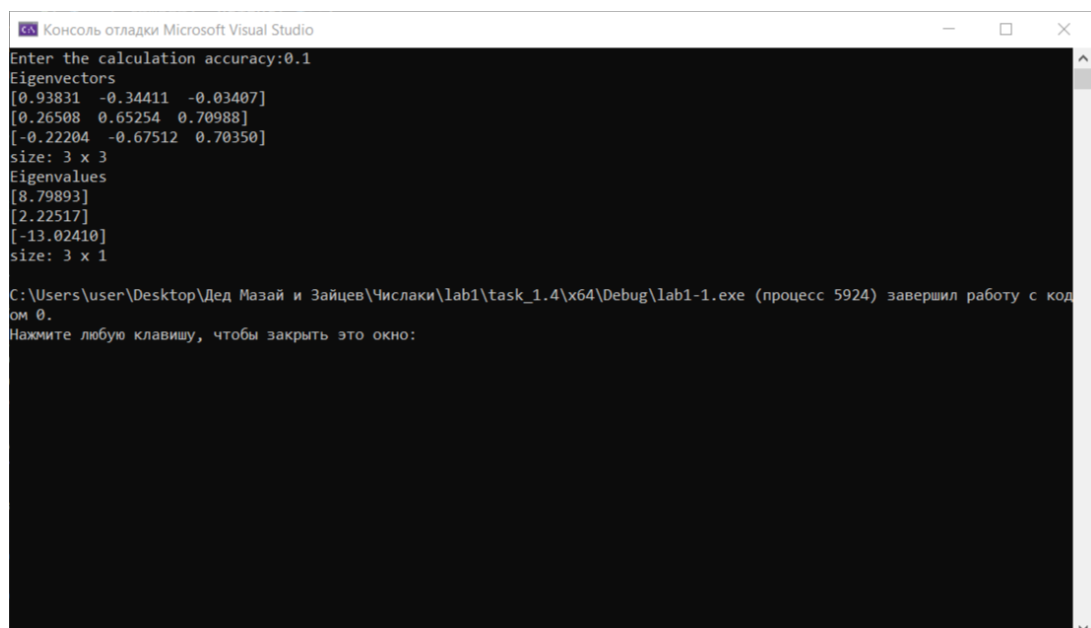
10 Постановка задачи

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: 4

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

11 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Enter the calculation accuracy:0.1
Eigenvectors
[0.93831 -0.34411 -0.03407]
[0.26508 0.65254 0.70988]
[-0.22204 -0.67512 0.70350]
size: 3 x 3
Eigenvalues
[8.79893]
[2.22517]
[-13.02410]
size: 3 x 1
C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab1\task_1.4\x64\Debug\lab1-1.exe (процесс 5924) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 4: Вывод программы в консоли

12 Исходный код

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include "read.h"
5
6 static inline double absolute(const double a) {
7     return a > 0 ? a : -a;
8 }
9
10 //      phi
11 Matrix* create_rotation_matrix(unsigned int size, unsigned int l, unsigned int m,
12     double phi) {
13     Matrix* rotate;
14     int i;
15     if (size <= 1 || size <= m || l == m)
16         return NULL;
17     rotate = create_matrix();
18     resize_matrix(rotate, size, size);
19     for (i = 0; i < size; i++)
20         rotate->data[i][i] = 1;
21     rotate->data[l][l] = rotate->data[m][m] = cos(phi);
22     rotate->data[l][m] = -(rotate->data[m][l] = sin(phi));
23     return rotate;
24 }
25
26 //
27 double error(Matrix* matrix) {
28     int i, j;
29     double err = 0;
30     if (!matrix)
31         return 0;
32     for (i = 0; i < matrix->height; i++)
33         for (j = i + 1; j < matrix->width; j++)
34             err += matrix->data[i][j] * matrix->data[i][j];
35     return sqrt(err);
36 }
37
38 //
39 Matrix* rotation_method(Matrix* matrix, Matrix** lambda, double epsilon) {
40     int i, j, i_max, j_max, step;
41     double max = 0, err = epsilon + 1;
42     Matrix* result, * rotate, * temp, * temp2;
43     if (!matrix || matrix->height != matrix->width) {
44         fprintf(stderr, "Not square matrix\n");
45         return NULL;
46     }
47     *lambda = create_matrix();
```

```

47     resize_matrix(*lambda, matrix->height, matrix->width);
48     result = create_rotation_matrix((*lambda)->height, 1, 0, 0);
49     for (i = 0; i < matrix->height; i++)
50         for (j = 0; j < matrix->width; j++)
51             (*lambda)->data[i][j] = matrix->data[i][j];
52
53     for (step = 1; err >= epsilon; step++) {
54         for (i = 0; i < (*lambda)->height; i++)
55             for (j = i + 1; j < (*lambda)->width; j++)
56                 if (max < absolute((*lambda)->data[i][j])) {
57                     max = absolute((*lambda)->data[i][j]);
58                     i_max = i;
59                     j_max = j;
60                 }
61         rotate = create_rotation_matrix((*lambda)->height, i_max, j_max,
62             0.5 * atan(2 * (*lambda)->data[i_max][j_max] /
63                 ((*lambda)->data[i_max][i_max] - (*lambda)->data[j_max][j_max]]));
64         max = 0;
65         temp = multiplication(result, rotate);
66         for (i = 0; i < result->height; i++)
67             for (j = 0; j < result->width; j++)
68                 result->data[i][j] = temp->data[i][j];
69         remove_matrix(temp);
70         free(temp);
71         temp = matrix_transposition(rotate);
72         temp2 = multiplication(temp, *lambda);
73         remove_matrix(temp);
74         free(temp);
75         remove_matrix(*lambda);
76         free(*lambda);
77         *lambda = multiplication(temp2, rotate);
78         remove_matrix(temp2);
79         free(temp2);
80         remove_matrix(rotate);
81         free(rotate);
82         err = error(*lambda);
83     }
84     for (i = 0; i < (*lambda)->height; i++)
85         (*lambda)->data[i][0] = (*lambda)->data[i][i];
86     resize_matrix(*lambda, (*lambda)->height, 1);
87     return result;
88 }
89
90 int main(void) {
91     int i;
92     float epsilon;
93     Matrix* matrix = create_matrix(), * vector, * result;
94     FILE* fmatrix;
95

```

```

96     fmatrix = fopen("lab01-4matrix.txt", "r");
97
98     printf("Enter the calculation accuracy:");
99     scanf("%f", &epsilon);
100     if (epsilon <= 0) {
101         fprintf(stderr, "Negative value of error\n");
102         return 0;
103     }
104     if (!fmatrix) {
105         fprintf(stderr, "Invalid name of file\n");
106         return 0;
107     }
108
109     scan_matrix(matrix, fmatrix);
110     fclose(fmatrix);
111
112     if (result = rotation_method(matrix, &vector, epsilon)) {
113         printf("Eigenvectors\n");
114         print_matrix(result, stdout);
115         printf("Eigenvalues\n");
116         print_matrix(vector, stdout);
117         remove_matrix(result);
118         free(result);
119         remove_matrix(vector);
120         free(vector);
121     }
122     remove_matrix(matrix);
123     free(matrix);
124     return 0;
125 }

```

Матрица:

```

1 | 8 2 -1
2 | 2 -5 -8
3 | -1 -8 -5

```

1.5 QR – разложение матриц

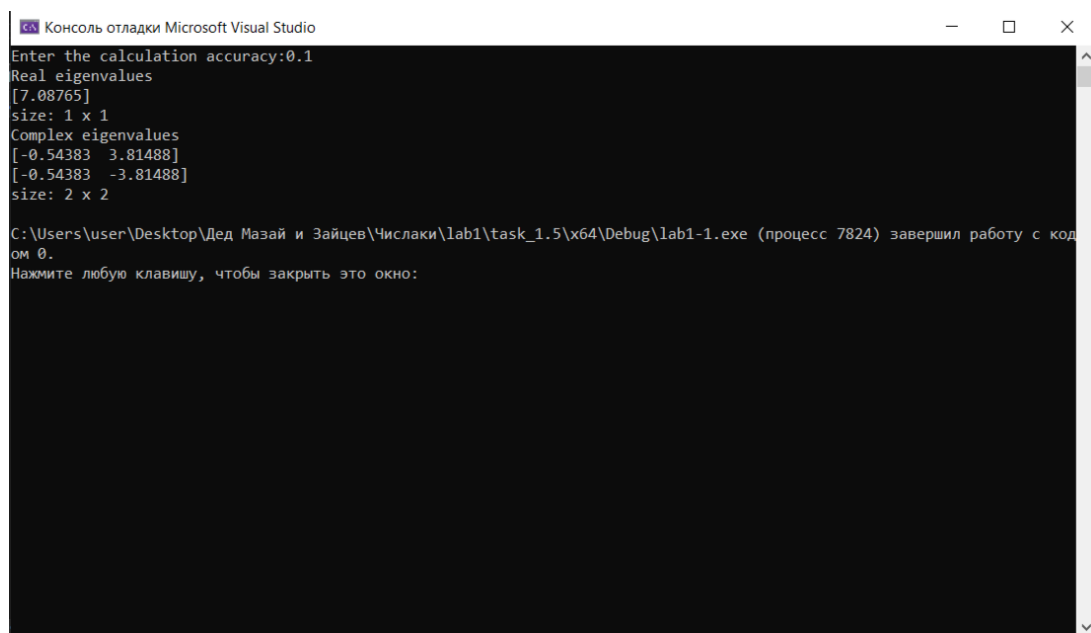
13 Постановка задачи

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: 4

$$\begin{pmatrix} -4 & -6 & -3 \\ -1 & 5 & -5 \\ 6 & 2 & 5 \end{pmatrix}$$

14 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Enter the calculation accuracy:0.1
Real eigenvalues
[7.08765]
size: 1 x 1
Complex eigenvalues
[-0.54383 3.81488]
[-0.54383 -3.81488]
size: 2 x 2
C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab1\task_1.5\x64\Debug\lab1-1.exe (процесс 7824) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 5: Вывод программы в консоли

15 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "read.h"
5
6  static inline double absolute(const double a) {
7      return a > 0 ? a : -a;
8  }
9  static inline sign(const double a) {
10     return a > 0 ? 1 : (a < 0 ? -1 : 0);
11 }
12
13 // QR
14 Matrix** QR_decomposition(Matrix* matrix) {
15     Matrix** QR, * vector, * temp, * temp2;
16     int i, j, k;
17     if (!matrix || matrix->height != matrix->width) {
18         fprintf(stderr, "Not square matrix\n");
19         return NULL;
20     }
21     QR = (Matrix**)malloc(sizeof(Matrix*) * 2);
22     QR[0] = create_matrix();
23     resize_matrix(QR[0], matrix->height, matrix->width);
24     QR[1] = create_matrix();
25     resize_matrix(QR[1], matrix->height, matrix->width);
26
27     for (i = 0; i < matrix->height; i++) {
28         for (j = 0; j < matrix->width; j++)
29             QR[1]->data[i][j] = matrix->data[i][j];
30         QR[0]->data[i][i] = 1;
31     }
32
33     for (j = 0; j < matrix->width - 1; j++) {
34         double sum = 0;
35         vector = create_matrix();
36         resize_matrix(vector, matrix->height, 1);
37         for (i = 0; i < matrix->height; i++)
38             vector->data[i][0] = i < j ? 0 : QR[1]->data[i][j];
39         for (i = j; i < vector->height; i++)
40             sum += vector->data[i][0] * vector->data[i][0];
41         vector->data[j][0] += sign(vector->data[j][0]) * sqrt(sum);
42
43         sum = 0;
44         temp = matrix_transposition(vector);
45         temp2 = multiplication(vector, temp);
46         remove_matrix(temp);
47         free(temp);
```

```

48
49     for (i = j; i < vector->height; i++)
50         sum += temp2->data[i][i];
51     for (i = 0; i < vector->height; i++)
52         for (k = 0; k < vector->height; k++)
53             temp2->data[i][k] = (i == k ? 1 : 0) - 2 * temp2->data[i][k] / sum;
54     temp = multiplication(QR[0], temp2);
55
56     for (i = 0; i < QR[0]->height; i++)
57         for (k = 0; k < QR[0]->width; k++)
58             QR[0]->data[i][k] = temp->data[i][k];
59
60     remove_matrix(temp);
61     free(temp);
62
63     temp = multiplication(temp2, QR[1]);
64     for (i = 0; i < QR[1]->height; i++)
65         for (k = 0; k < QR[1]->width; k++)
66             QR[1]->data[i][k] = temp->data[i][k];
67
68     remove_matrix(temp);
69     free(temp);
70     remove_matrix(temp2);
71     free(temp2);
72     remove_matrix(vector);
73     free(vector);
74 }
75 return QR;
76 }
77
78 // QR
79 int stop_criterion(Matrix* matrix, double epsilon) {
80     int mark = 0, i, j;
81
82     for (j = 0; j < matrix->width - 1; j++) {
83         double error = 0;
84         for (i = j + 1; i < matrix->height; i++)
85             error += matrix->data[i][j] * matrix->data[i][j];
86         if (sqrt(error) > epsilon)
87             if (sqrt(error - matrix->data[j + 1][j] * matrix->data[j + 1][j]) > epsilon
88                 || mark)
89                 return 0;
90             else
91                 mark = 1;
92         else
93             mark = 0;
94     }
95     return 1;
96 }

```

```

96
97 // QR
98 //
99 Matrix** QR_algorithm(Matrix* matrix, double epsilon) {
100     Matrix** QR, * lambda;
101     int step, i, j;
102     lambda = create_matrix();
103     resize_matrix(lambda, matrix->height, matrix->width);
104     for (i = 0; i < matrix->height; i++)
105         for (j = 0; j < matrix->width; j++)
106             lambda->data[i][j] = matrix->data[i][j];
107     for (step = 1; !stop_criterion(lambda, epsilon); step++) {
108         QR = QR_decomposition(lambda);
109         remove_matrix(lambda);
110         free(lambda);
111         lambda = multiplication(QR[1], QR[0]);
112
113         remove_matrix(QR[0]);
114         free(QR[0]);
115         remove_matrix(QR[1]);
116         free(QR[1]);
117         free(QR);
118     }
119     QR = (Matrix**)malloc(sizeof(Matrix*) * 2);
120     QR[0] = create_matrix();
121     QR[1] = create_matrix();
122     for (j = 0; j < lambda->width - 1; j++) {
123         double sum = 0;
124         for (i = j + 1; i < lambda->height; i++)
125             sum += lambda->data[i][j] * lambda->data[i][j];
126         if (epsilon > sqrt(sum)) {
127             resize_matrix(QR[0], QR[0]->height + 1, 1);
128             QR[0]->data[QR[0]->height - 1][0] = lambda->data[j][j];
129         }
130         else {
131             double b = lambda->data[j][j] + lambda->data[j + 1][j + 1];
132             double c = lambda->data[j][j] * lambda->data[j + 1][j + 1] - lambda->data[j
133                 ][j + 1] * lambda->data[j + 1][j];
134             resize_matrix(QR[1], QR[1]->height + 2, 2);
135             QR[1]->data[QR[1]->height - 1][0] = QR[1]->data[QR[1]->height - 2][0] = 0.5
136                 * b;
137             QR[1]->data[QR[1]->height - 1][1] = -(QR[1]->data[QR[1]->height - 2][1] =
138                 0.5 * sqrt(4 * c - b * b));
139             j++;
140         }
141     }
142     if (QR[0]->height + QR[1]->height != lambda->height) {
143         resize_matrix(QR[0], QR[0]->height + 1, 1);
144     }

```

```

141     QR[0]->data[QR[0]->height - 1][0] = lambda->data[lambda->height - 1][lambda->
        height - 1];
142 }
143 remove_matrix(lambda);
144 free(lambda);
145 return QR;
146 }
147
148 int main(void) {
149     int i;
150     float epsilon;
151     Matrix* matrix = create_matrix(), ** result;
152     FILE* fmatrix;
153
154     fmatrix = fopen("lab01-5matrix.txt", "r");
155
156     printf("Enter the calculation accuracy:");
157     scanf("%f", &epsilon);
158     if (epsilon <= 0) {
159         fprintf(stderr, "Negative value of error\n");
160         return 0;
161     }
162     if (!fmatrix) {
163         fprintf(stderr, "Invalid name of file\n");
164         return 0;
165     }
166
167     scan_matrix(matrix, fmatrix);
168     fclose(fmatrix);
169     if (result = QR_algorithm(matrix, epsilon)) {
170         if (result[0]->height) {
171             printf("Real eigenvalues\n");
172             print_matrix(result[0], stdout);
173         }
174         if (result[1]->height) {
175             printf("Complex eigenvalues\n");
176             print_matrix(result[1], stdout);
177         }
178         remove_matrix(result[0]);
179         free(result[0]);
180         remove_matrix(result[1]);
181         free(result[1]);
182         free(result);
183     }
184     remove_matrix(matrix);
185     free(matrix);
186     return 0;
187 }

```

Матрица:

```

1  -4 -6 -3
2  -1 5 -5
3  6 2 5

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "read.h"
5
6  static inline double absolute(const double a) {
7      return a > 0 ? a : -a;
8  }
9  static inline sign(const double a) {
10     return a > 0 ? 1 : (a < 0 ? -1 : 0);
11 }
12
13 // QR
14 Matrix** QR_decomposition(Matrix* matrix) {
15     Matrix** QR, * vector, * temp, * temp2;
16     int i, j, k;
17     if (!matrix || matrix->height != matrix->width) {
18         fprintf(stderr, "Not square matrix\n");
19         return NULL;
20     }
21     QR = (Matrix**)malloc(sizeof(Matrix*) * 2);
22     QR[0] = create_matrix();
23     resize_matrix(QR[0], matrix->height, matrix->width);
24     QR[1] = create_matrix();
25     resize_matrix(QR[1], matrix->height, matrix->width);
26
27     for (i = 0; i < matrix->height; i++) {
28         for (j = 0; j < matrix->width; j++)
29             QR[1]->data[i][j] = matrix->data[i][j];
30         QR[0]->data[i][i] = 1;
31     }
32
33     for (j = 0; j < matrix->width - 1; j++) {
34         double sum = 0;
35         vector = create_matrix();
36         resize_matrix(vector, matrix->height, 1);
37         for (i = 0; i < matrix->height; i++)
38             vector->data[i][0] = i < j ? 0 : QR[1]->data[i][j];
39         for (i = j; i < vector->height; i++)
40             sum += vector->data[i][0] * vector->data[i][0];
41         vector->data[j][0] += sign(vector->data[j][0]) * sqrt(sum);
42
43         sum = 0;
44         temp = matrix_transposition(vector);
45         temp2 = multiplication(vector, temp);
46         remove_matrix(temp);

```

```

47     free(temp);
48
49     for (i = j; i < vector->height; i++)
50         sum += temp2->data[i][i];
51     for (i = 0; i < vector->height; i++)
52         for (k = 0; k < vector->height; k++)
53             temp2->data[i][k] = (i == k ? 1 : 0) - 2 * temp2->data[i][k] / sum;
54     temp = multiplication(QR[0], temp2);
55
56     for (i = 0; i < QR[0]->height; i++)
57         for (k = 0; k < QR[0]->width; k++)
58             QR[0]->data[i][k] = temp->data[i][k];
59
60     remove_matrix(temp);
61     free(temp);
62
63     temp = multiplication(temp2, QR[1]);
64     for (i = 0; i < QR[1]->height; i++)
65         for (k = 0; k < QR[1]->width; k++)
66             QR[1]->data[i][k] = temp->data[i][k];
67
68     remove_matrix(temp);
69     free(temp);
70     remove_matrix(temp2);
71     free(temp2);
72     remove_matrix(vector);
73     free(vector);
74 }
75 return QR;
76 }
77
78 // QR
79 int stop_criterion(Matrix* matrix, double epsilon) {
80     int mark = 0, i, j;
81
82     for (j = 0; j < matrix->width - 1; j++) {
83         double error = 0;
84         for (i = j + 1; i < matrix->height; i++)
85             error += matrix->data[i][j] * matrix->data[i][j];
86         if (sqrt(error) > epsilon)
87             if (sqrt(error - matrix->data[j + 1][j] * matrix->data[j + 1][j]) > epsilon
88                 || mark)
89                 return 0;
90             else
91                 mark = 1;
92         else
93             mark = 0;
94     }
95     return 1;

```

```

95 }
96
97 // QR
98 // , -
99 Matrix** QR_algorithm(Matrix* matrix, double epsilon) {
100     Matrix** QR, * lambda;
101     int step, i, j;
102     lambda = create_matrix();
103     resize_matrix(lambda, matrix->height, matrix->width);
104     for (i = 0; i < matrix->height; i++)
105         for (j = 0; j < matrix->width; j++)
106             lambda->data[i][j] = matrix->data[i][j];
107     for (step = 1; !stop_criterion(lambda, epsilon); step++) {
108         QR = QR_decomposition(lambda);
109         remove_matrix(lambda);
110         free(lambda);
111         lambda = multiplication(QR[1], QR[0]);
112
113         remove_matrix(QR[0]);
114         free(QR[0]);
115         remove_matrix(QR[1]);
116         free(QR[1]);
117         free(QR);
118     }
119     QR = (Matrix**)malloc(sizeof(Matrix*) * 2);
120     QR[0] = create_matrix();
121     QR[1] = create_matrix();
122     for (j = 0; j < lambda->width - 1; j++) {
123         double sum = 0;
124         for (i = j + 1; i < lambda->height; i++)
125             sum += lambda->data[i][j] * lambda->data[i][j];
126         if (epsilon > sqrt(sum)) {
127             resize_matrix(QR[0], QR[0]->height + 1, 1);
128             QR[0]->data[QR[0]->height - 1][0] = lambda->data[j][j];
129         }
130         else {
131             double b = lambda->data[j][j] + lambda->data[j + 1][j + 1];
132             double c = lambda->data[j][j] * lambda->data[j + 1][j + 1] - lambda->data[j][j + 1] * lambda->data[j + 1][j];
133             resize_matrix(QR[1], QR[1]->height + 2, 2);
134             QR[1]->data[QR[1]->height - 1][0] = QR[1]->data[QR[1]->height - 2][0] = 0.5 * b;
135             QR[1]->data[QR[1]->height - 1][1] = -(QR[1]->data[QR[1]->height - 2][1] = 0.5 * sqrt(4 * c - b * b));
136             j++;
137         }
138     }
139     if (QR[0]->height + QR[1]->height != lambda->height) {
140         resize_matrix(QR[0], QR[0]->height + 1, 1);

```

```

141     QR[0]->data[QR[0]->height - 1][0] = lambda->data[lambda->height - 1][lambda->
        height - 1];
142 }
143 remove_matrix(lambda);
144 free(lambda);
145 return QR;
146 }
147
148 int main(void) {
149     int i;
150     float epsilon;
151     Matrix* matrix = create_matrix(), ** result;
152     FILE* fmatrix;
153
154     fmatrix = fopen("lab01-5matrix.txt", "r");
155
156     printf("Enter the calculation accuracy:");
157     scanf("%f", &epsilon);
158     if (epsilon <= 0) {
159         fprintf(stderr, "Negative value of error\n");
160         return 0;
161     }
162     if (!fmatrix) {
163         fprintf(stderr, "Invalid name of file\n");
164         return 0;
165     }
166
167     scan_matrix(matrix, fmatrix);
168     fclose(fmatrix);
169     if (result = QR_algorithm(matrix, epsilon)) {
170         if (result[0]->height) {
171             printf("Real eigenvalues\n");
172             print_matrix(result[0], stdout);
173         }
174         if (result[1]->height) {
175             printf("Complex eigenvalues\n");
176             print_matrix(result[1], stdout);
177         }
178         remove_matrix(result[0]);
179         free(result[0]);
180         remove_matrix(result[1]);
181         free(result[1]);
182         free(result);
183     }
184     remove_matrix(matrix);
185     free(matrix);
186     return 0;
187 }

```