

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

Москва, 2024

1.1 LU - разложение матриц

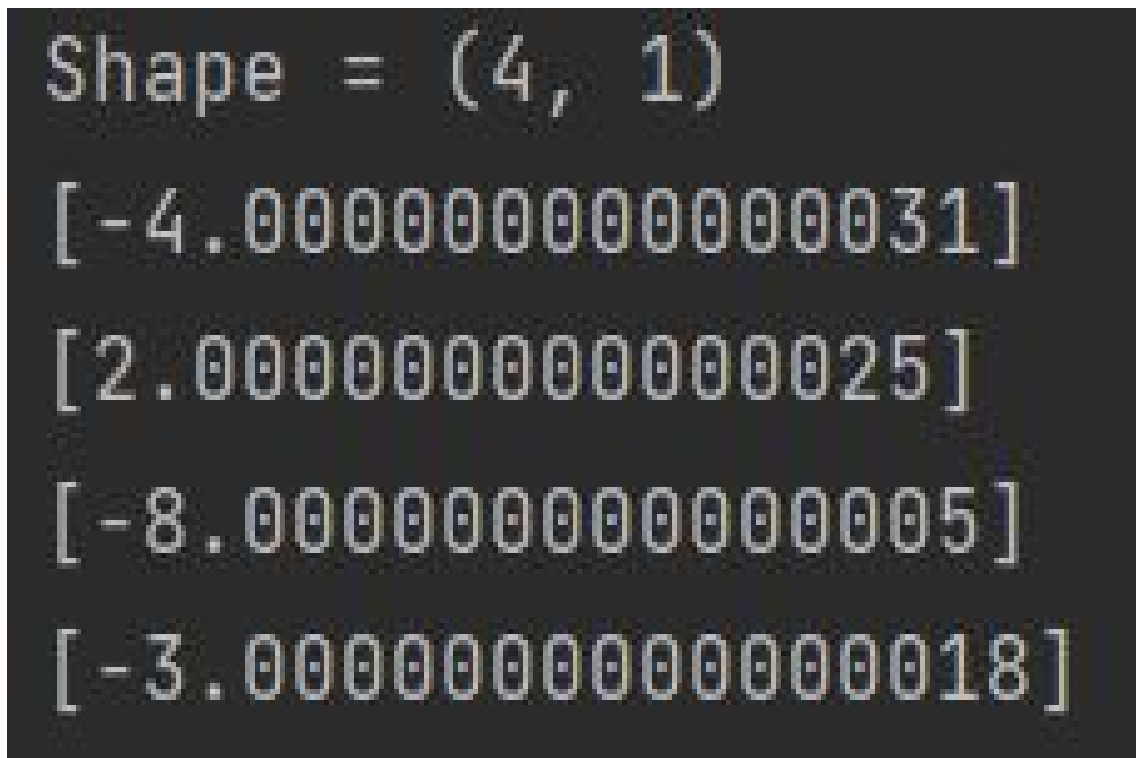
1 Постановка задачи

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: 9

$$\begin{cases} -2x_1 - 9x_2 - 3x_3 + 7x_4 = -26 \\ -7x_1 + 8x_2 + 2x_3 + 5x_4 = -25 \\ -6x_1 + 2x_2 = -16 \\ -3x_2 + 8x_3 - 3x_4 = -5 \end{cases}$$

2 Результаты работы



```
Shape = (4, 1)
[-4.00000000000000000000000031]
[2.00000000000000000000000025]
[-8.00000000000000000000000005]
[-3.00000000000000000000000018]
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1 class Vector:
2     def __init__(self, values: list[None | int | float]):
3         self.values = [value for value in values]
4         self.n = len(values)
5
6     def __str__(self) -> str:
7         return self.values.__str__()
8
9     def copy(self):
10        return Vector([value for value in self.values])
11
12    def __len__(self) -> int:
13        return self.values.__len__()
14
15    def __getitem__(self, index: int):
16        return self.values[index]
17
18    def __setitem__(self, index: int, value: int | float):
19        self.values[index] = value
20
21    def upload_values(self, values: list[list[None | int | float]]):
22        self.values = [value for value in values]
23
24
25 class Matrix:
26     def __init__(self, values: list[list[None | int | float] | Vector]):
27         if type(values[0]) == Vector:
28             self.values = [vector.copy() for vector in values]
29         else:
30             self.values = [Vector(row) for row in values]
31         self.n: int = len(self.values)
32         self.m: int = len(self.values[0])
33
34     def __str__(self) -> str:
35         m = '\n'.join([row.__str__() for row in self.values])
36         return f"Shape = {self.get_shape()} \n{m}"
37
38     def get_shape(self) -> tuple[int, int]:
39         return self.n, self.m
40
41     def copy(self):
42         return Matrix([vect.copy() for vect in self.values])
43
44     def upload_values(self, values: list[list[None | int | float]]):
45         self.values = [[val for val in row] for row in values]
46
47     def transposed(self):
```

```

48     res = [[None] * self.n for _ in range(self.m)]
49     for i in range(self.n):
50         for j in range(self.m):
51             res[j][i] = self.values[i][j]
52     return Matrix(res)
53
54     def __setitem__(self, index: int, value: Vector):
55         self.values[index] = value
56
57     def __getitem__(self, index: int):
58         return self.values[index]
59
60
61 def multiple_matrix(matrix1: Matrix, matrix2: Matrix) -> None | Matrix:
62     n1, m1 = matrix1.get_shape()
63     n2, m2 = matrix2.get_shape()
64     if m1 != n2: # TODO: implement custom Error for matrices
65         print("Incorrect shapes of matrices")
66         return
67     res: list[list[None | int | float]] = [[None]*m2 for _ in range(n1)]
68     n, m, h = n1, m2, m1
69     del n1, n2, m1, m2
70     for i in range(n):
71         for j in range(m):
72             cntr = 0
73             for k in range(h):
74                 cntr += matrix1.values[i][k] * matrix2.values[k][j]
75             res[i][j] = cntr
76
77     return Matrix(res)
78
79
80 def plus_matrix(matrix1: Matrix, matrix2: Matrix) -> Matrix:
81     res = [[matrix1[i][j]+matrix2[i][j] for j in range(matrix1.m)] for i in range(
82         matrix1.n)]
83     return Matrix(res)
84
85
86 from matrix import *
87
88
89 def lu_decomposition(coefficients: Matrix, results: Matrix | None = None) -> tuple[
90     Matrix, Matrix]:
91     # coefficients and results might be changed in outer code
92     L = Matrix([[0] * coefficients.m for _ in range(coefficients.n)])
93     U = Matrix([[value for value in row] for row in coefficients.values])
94
95     # Straight Gaussian stroke
96     for k in range(coefficients.n):
97         if U[k][k] == 0:
98             for i in range(k+1, coefficients.n):

```

```

13         if U[i][k] != 0:
14             U[k], U[i] = U[i], U[k]
15             L[k], L[i] = L[i], L[k]
16             coefficients[k], coefficients[i] = coefficients[i], coefficients[k]
17             if results:
18                 results[k], results[i] = results[i], results[k]
19             break
20     else:
21         print("There are not solutions") # TODO: create custom Exception
22         raise Exception
23     L[k][k] = 1
24     for i in range(k+1, coefficients.n):
25         L[i][k] = U[i][k]/U[k][k]
26         if U[i][k] == 0:
27             continue
28         for j in range(k, coefficients.m):
29             U[i][j] -= L[i][k]*U[k][j]
30
31     return L, U
32
33
34 def get_determinant(coefficients: Matrix) -> int | float:
35     _, U = lu_decomposition(coefficients)
36     det = 1
37     for i in range(coefficients.n):
38         det *= U[i][i]
39     return det
40
41
42 def calculate_decisions(coefficients: Matrix, results: Matrix) -> Matrix:
43     L, U = lu_decomposition(coefficients, results)
44     res = results.copy()
45     for k in range(res.m):
46         for i in range(res.n):
47             for j in range(i):
48                 res[i][k] -= res[j][k]*L[i][j]
49
50     for k in range(res.m):
51         for i in range(coefficients.n-1, -1, -1):
52             for j in range(i+1, results.n):
53                 res[i][k] -= res[j][k]*U[i][j]
54             res[i][k] /= U[i][i]
55     return res
56
57
58 def get_inverse_matrix(matrix: Matrix) -> Matrix:
59     E = Matrix([[1 if i == j else 0 for j in range(matrix.n)] for i in range(matrix.n)])
60     return calculate_decisions(matrix, E)
61

```

```

62
63 if __name__ == "__main__":
64     coefficient_matrix = Matrix([
65         [-7, -9, 1, -9],
66         [-6, -8, -5, 2],
67         [-3, 6, 5, -9],
68         [-2, 0, -5, -9]
69     ])
70
71     equation_roots = Matrix([
72         [29],
73         [42],
74         [11],
75         [75]
76     ])
77
78     print(calculate_decisions(coefficient_matrix, equation_roots))

```


6 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using matrix = vector<vector<double> >;
5
6
7  matrix tridiagonal_algorithm(matrix& coefficients, matrix& results) {
8      double a, b, c, d;
9      a = 0;
10     b = coefficients[0][0];
11     c = coefficients[0][1];
12     d = results[0][0];
13     vector<double> P(coefficients[0].size(), 0), Q(coefficients[0].size(), 0);
14
15     P[0] = -c/b;
16     Q[0] = d/b;
17     for (int i=1; i < coefficients.size() - 1; i++){
18         a = coefficients[i][i-1];
19         b = coefficients[i][i];
20         c = coefficients[i][i+1];
21         d = results[i][0];
22
23         P[i] = -c/(b + a*P[i-1]);
24         Q[i] = (d - a*Q[i-1])/(b + a*P[i-1]);
25     }
26
27     a = coefficients[coefficients.size()-1][coefficients[0].size()-2];
28     b = coefficients[coefficients.size()-1][coefficients[0].size()-1];
29     c = 0;
30     d = results[results.size()-1][0];
31
32     Q[Q.size()-1] = (d - a * Q[Q.size()-2]) / (b + a * P[P.size()-2]);
33
34     matrix decision(results.size());
35     for(int i=0; i<decision.size(); i++)
36         decision[i].push_back(0);
37
38     decision[decision.size()-1][0] = Q[Q.size()-1];
39     for (int i = decision.size()-2; i > -1; i--)
40         decision[i][0] = P[i]*decision[i+1][0] + Q[i];
41
42     return decision;
43 }
44
45
46 void print_matrix(matrix& matrix1) {
47     for(const auto& vect: matrix1) {
```



```

48     for (auto x: vect)
49         cout << x << " ";
50     cout << endl;
51 }
52 }
53
54 int main() {
55     matrix coefficient_matrix{
56         {8, -4, 0, 0, 0},
57         {-2, 12, -7, 0, 0},
58         {0, 2, -9, 1, 0},
59         {0, 0, -8, 17, -4},
60         {0, 0, 0, -7, 13}
61     };
62
63     matrix equation_roots = {
64         {32},
65         {15},
66         {-10},
67         {133},
68         {-76}
69     };
70
71     matrix res_matrix = tridiagonal_algorithm(coefficient_matrix, equation_roots);
72
73     cout << "Desicion" << endl;
74     for (int i=0; i<res_matrix.size(); i++)
75         printf("x%d = %f \n", i+1, res_matrix[i][0]);
76
77     return 0;
78 }

```

```

1  from matrix import *
2
3
4  def tridiagonal_algorithm(coefficients: Matrix, results: Matrix) -> Matrix:
5      if coefficients.n < 3 or coefficients.m < 3:
6          print("Incorrect shapes of matrices")
7          raise Exception
8      a, b, c, d = 0, coefficients[0][0], coefficients[0][1], results.values[0][0]
9      P, Q = [0]*coefficients.m, [0]*coefficients.m
10
11      P[0], Q[0] = -c/b, d/b
12      for i in range(1, coefficients.n-1):
13          a, b, c, d = coefficients[i][i-1], coefficients[i][i], coefficients[i][i+1],
14                      results[i][0]
15          P[i] = -c/(b + a*P[i-1])
16          Q[i] = (d - a*Q[i-1])/(b + a*P[i-1])
17      a, b, c, d = coefficients[-1][-2], coefficients[-1][-1], 0, results[-1][0]
18      Q[-1] = (d - a * Q[-2]) / (b + a * P[-2])

```

```

18
19     decisions = [0]*results.n
20     decisions[-1] = Q[-1]
21     for i in range(decisions.__len__()-2, -1, -1):
22         decisions[i] = P[i]*decisions[i+1] + Q[i]
23
24     return Matrix([[i] for i in decisions])
25
26
27 if __name__ == "__main__":
28     coefficient_matrix = Matrix([
29         [8, -4, 0, 0, 0],
30         [-2, 12, -7, 0, 0],
31         [0, 2, -9, 1, 0],
32         [0, 0, -8, 17, -4],
33         [0, 0, 0, -7, 13]
34     ])
35
36     equation_roots = Matrix([
37         [32],
38         [15],
39         [-10],
40         [133],
41         [-76]
42     ])
43
44     print(tridiagonal_algorithm(coefficient_matrix, equation_roots))

```

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: 9

$$\begin{cases} 18x_1 - 2x_3 + 7x_4 = 50 \\ -x_1 + 14x_2 - 3x_3 + 2x_4 = 2 \\ 5x_1 + 5x_2 + 26x_3 + 7x_4 = 273 \\ -2x_1 - 6x_2 + 9x_3 + 24x_4 = 111 \end{cases}$$

8 Результаты работы

```
12
Shape = (4, 1)
[-0.0005290275783376153]
[7.000377489396968]
[-5.000571089584128]
[-5.000147032888273]

10
Shape = (4, 1)
[4.2599228980533255e-05]
[6.999919682993307]
[-4.9999863321702325]
[-4.999965036430815]
```

Рис. 3: Вывод программы в консоли

9 Исходный код

```
1 from matrix import *
2
3
4 def preprocessing(coefficients: Matrix, results: Matrix) -> None:
5     n, m = coefficients.get_shape()
6     for i in range(n):
7         a = coefficients[i][i]
8         if a == 0:
9             continue
10        results[i][0] /= a
11        for j in range(m):
12            if i == j:
13                coefficients[i][j] = 0
14            else:
15                coefficients[i][j] /= -a
16
17
18 def get_current_eps(vect1: Matrix, vect2: Matrix) -> float:
19     eps = 0
20     for i in range(vect1.get_shape()[0]):
21         eps += (vect1[i][0] - vect2[i][0]) ** 2
22     return eps ** 0.5
23
24
25 def simple_iterations(a: Matrix, b: Matrix, EPS0: float) -> tuple[int, Matrix]:
26     x_previous = Matrix(b.values)
27     k = 0
28     while get_current_eps(x_current := plus_matrix(b, multiple_matrix(a, x_previous)),
29                           x_previous) > EPS0:
30         k += 1
31         x_current, x_previous = None, x_current
32     return k, x_previous
33
34 def seidel_method(a: Matrix, b: Matrix, EPS0: float) -> tuple[int, Matrix]:
35     x_previous = Matrix(b.values)
36     x_current = x_previous.copy()
37     flag = True
38     k, eps = 0, 0
39     while flag or eps > EPS0:
40         k += 1
41         flag = False
42         for i in range(b.get_shape()[0]):
43             x_current[i][0] = 0
44             for j in range(b.get_shape()[0]):
45                 if i == j:
46                     x_current[i][0] += b[i][0]
```

```

47         elif i < j:
48             x_current[i][0] += a[i][j]*x_previous[j][0]
49         else:
50             x_current[i][0] += a[i][j]*x_current[j][0]
51     eps = get_current_eps(x_current, x_previous)
52     x_current, x_previous = x_current, x_current.copy()
53     return k, x_previous
54
55
56 if __name__ == "__main__":
57     coefficient_matrix = Matrix([
58         [12, -3, -1, 3],
59         [5, 20, 9, 1],
60         [6, -3, -21, -7],
61         [8, -7, 3, -27]
62     ])
63
64     equation_roots = Matrix([
65         [-31],
66         [90],
67         [119],
68         [71]
69     ])
70
71     preprocessing(coefficient_matrix, equation_roots)
72
73     last_iteration, res_matrix = simple_iterations(coefficient_matrix, equation_roots,
74         0.001)
75     print(last_iteration)
76     print(res_matrix)
77     print('\n\n')
78
79     last_iteration, res_matrix = seidel_method(coefficient_matrix, equation_roots,
80         0.001)
81     print(last_iteration)
82     print(res_matrix)
83
84
85 1 #include <bits/stdc++.h>
86 2
87 3 using namespace std;
88 4 using matrix = vector<vector<double> >;
89 5
90 6
91 7 matrix multiply_matrix(const matrix& matrix1, const matrix& matrix2) {
92 8     int n1 = matrix1.size();
93 9     int m1 = matrix1[0].size();
94 10    int n2 = matrix2.size();
95 11    int m2 = matrix2[0].size();
96 12

```

```

13     if (m1 != n2) {
14         cout << "Incorrect shapes of matrices" << endl;
15         return matrix();
16     }
17
18     matrix res(n1, vector<double>(m2, 0));
19
20     for (int i = 0; i < n1; ++i) {
21         for (int j = 0; j < m2; ++j) {
22             double cntr = 0;
23             for (int k = 0; k < m1; ++k) {
24                 cntr += matrix1[i][k] * matrix2[k][j];
25             }
26             res[i][j] = cntr;
27         }
28     }
29
30     return res;
31 }
32
33
34 matrix plus_matrix(const matrix& matrix1, const matrix& matrix2) {
35     int n = matrix1.size();
36     int m = matrix1[0].size();
37
38     matrix res(n, vector<double>(m));
39     for (int i = 0; i < n; ++i) {
40         for (int j = 0; j < m; ++j) {
41             res[i][j] = matrix1[i][j] + matrix2[i][j];
42         }
43     }
44
45     return res;
46 }
47
48 matrix minus_matrix(const matrix& matrix1, const matrix& matrix2) {
49     int n = matrix1.size();
50     int m = matrix1[0].size();
51
52     matrix res(n, vector<double>(m));
53     for (int i = 0; i < n; ++i) {
54         for (int j = 0; j < m; ++j) {
55             res[i][j] = matrix1[i][j] - matrix2[i][j];
56         }
57     }
58
59     return res;
60 }
61

```

```

62
63 void preprocessing(matrix& coefficients, matrix& results) {
64     double n = coefficients.size(), m = coefficients[0].size();
65     for(int i=0; i < n; i++) {
66         double a = coefficients[i][i];
67         if (a == 0)
68             continue;
69         results[i][0] /= a;
70         for (int j=0; j < m; j++)
71             if (i == j)
72                 coefficients[i][j] = 0;
73             else
74                 coefficients[i][j] /= -a;
75     }
76 }
77
78
79 double get_current_eps(const matrix& vect1, const matrix& vect2) {
80     double eps = 0;
81     for(int i=0; i<vect1.size(); i++)
82         eps += pow(vect1[i][0] - vect2[i][0], 2);
83     return sqrt(eps);
84 }
85
86
87 void print_matrix(const matrix& matrix1) {
88     for(const auto& vect: matrix1) {
89         cout << '\t';
90         for (auto x: vect)
91             cout << x << " ";
92         cout << endl;
93     }
94 }
95
96
97 pair<int, matrix> simple_iterations(matrix& a, matrix& b, double EPS0) {
98     matrix x_previous = b;
99     matrix x_current;
100     int k = 0;
101     while (get_current_eps(x_current = plus_matrix(b, multiply_matrix(a, x_previous)),
102         x_previous) > EPS0) {
103         k += 1;
104         x_previous = x_current;
105     }
106     return make_pair(k, x_previous);
107 }
108
109 pair<int, matrix> seidel_method(const matrix& a, const matrix& b, double EPS0) {

```

```

110     matrix x_previous = b;
111     matrix x_current = x_previous;
112     bool flag = true;
113     int k = 0;
114     double eps = 0;
115     while (flag or eps > EPS0) {
116         k += 1;
117         flag = false;
118         for(int i = 0; i < b.size(); i++) {
119             x_current[i][0] = 0;
120             for(int j=0; j < b.size(); j++) {
121                 if (i == j)
122                     x_current[i][0] += b[i][0];
123                 else if (i < j)
124                     x_current[i][0] += a[i][j]*x_previous[j][0];
125                 else
126                     x_current[i][0] += a[i][j]*x_current[j][0];
127             }
128         }
129         eps = get_current_eps(x_current, x_previous);
130         x_previous = x_current;
131     }
132     return make_pair(k, x_previous);
133 }
134
135
136 int main() {
137     matrix coefficient_matrix = {
138         {12, -3, -1, 3},
139         {5, 20, 9, 1},
140         {6, -3, -21, -7},
141         {8, -7, 3, -27}
142     };
143
144     matrix equation_roots = {
145         {-31},
146         {90},
147         {119},
148         {71}
149     };
150
151     preprocessing(coefficient_matrix, equation_roots);
152
153     int last_iteration;
154     matrix res_matrix;
155
156     tie(last_iteration, res_matrix) = simple_iterations(coefficient_matrix,
157                                                         equation_roots, 0.001);
157     cout << "Simple iterations method:" << endl;

```



```

158     cout << "\tNumber of iterations = " << last_iteration << endl;
159     cout << "\tSolution=" << endl;
160     print_matrix(res_matrix);
161     cout << endl << endl;
162
163
164
165     tie(last_iteration, res_matrix) = seidel_method(coefficient_matrix, equation_roots,
        0.001);
166     cout << "Seidel's method:" << endl;
167     cout << "\tNumber of iterations = " << last_iteration << endl;
168     cout << "\tSolution = " << endl;
169     print_matrix(res_matrix);
170
171     return 0;
172 }

```

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: 9

$$\begin{pmatrix} 4 & 7 & -1 \\ 7 & -9 & -6 \\ -1 & -6 & -4 \end{pmatrix}$$

11 Результаты работы

```
Собственные значения:  
λ_1 = 8.071918520499581  
λ_2 = -14.623607766264337  
λ_3 = -2.4483107542352407  
  
Собственные векторы:  
x_1 = [0.8440417011154279, 0.449064021683711, -0.2931469106223472]  
x_2 = [-0.2930937368771562, 0.8440481813931533, 0.44908654944263804]  
x_3 = [0.4490987287685852, -0.2931282516513245, 0.8440297150588233]
```

Рис. 4: Вывод программы в консоли

12 Исходный код

```
1 import math
2
3 from matrix import *
4 from matrix import Vector, Matrix
5
6
7 def get_indexes(matrix1: Matrix) -> tuple[int, int]:
8     """
9     return indexes of maximum non-diagonal element
10    """
11    m: float = matrix1[0][1]
12    indexes: tuple[int, int] = (0, 1)
13    for i in range(matrix1.n):
14        for j in range(i+1, matrix1.m):
15            if abs(matrix1[i][j]) > m:
16                m = abs(matrix1[i][j])
17                indexes = (i, j)
18    return indexes
19
20
21 def get_phi(a_ij: float, a_ii: float, a_jj: float) -> float:
22     if a_ii == a_jj:
23         return math.pi/4
24     return math.atan(2*a_ij/(a_ii-a_jj)) / 2
25
26
27 def get_U_matrix(matrix1: Matrix) -> Matrix:
28     res = [[1 if i == j else 0 for j in range(matrix1.n)] for i in range(matrix1.n)]
29     i_max, j_max = get_indexes(matrix1)
30     phi = get_phi(a_ij=matrix1[i_max][j_max], a_ii=matrix1[i_max][i_max], a_jj=matrix1[
31         j_max][j_max])
32     res[i_max][i_max] = math.cos(phi)
33     res[j_max][j_max] = math.cos(phi)
34     res[i_max][j_max] = -math.sin(phi)
35     res[j_max][i_max] = math.sin(phi)
36     return Matrix(res)
37
38 def get_eps(matrix1: Matrix) -> float:
39     eps = 0
40     for i in range(matrix1.n):
41         for j in range(i+1, matrix1.m):
42             eps += matrix1[i][j]**2
43     return math.sqrt(eps)
44
45
46 def Jacobi_method(coeff_matrix: Matrix, EPS: float) -> tuple[Vector, Matrix]:
```

```

47     eigenvectors = Matrix([[1 if i == j else 0 for j in range(coeff_matrix.n)] for i in
48                             range(coeff_matrix.n)])
49     while get_eps(coeff_matrix) > EPS:
50         U = get_U_matrix(coeff_matrix)
51         eigenvectors = multiple_matrix(eigenvectors, U)
52         coeff_matrix = multiple_matrix(multiple_matrix(U.transposed(), coeff_matrix), U)
53     eigenvalues = Vector([coeff_matrix[i][i] for i in range(coeff_matrix.n)])
54     return eigenvalues, eigenvectors
55
56 if __name__ == "__main__":
57     coefficient_matrix = Matrix([
58         [4, 7, -1],
59         [7, -9, -6],
60         [-1, -6, -4]
61     ])
62
63     values, vectors = Jacobi_method(coefficient_matrix, EPS=0.01)
64     print(" :", *[f'_{i+1} = {values[i]}' for i in range(values.n)], sep='\n', end='\n\n')
65
66     print(" :")
67     for i in range(vectors.n):
68         print(f'x_{i+1} = {[vectors[j][i] for j in range(vectors.n)]}')

```

1.5 QR – разложение матриц

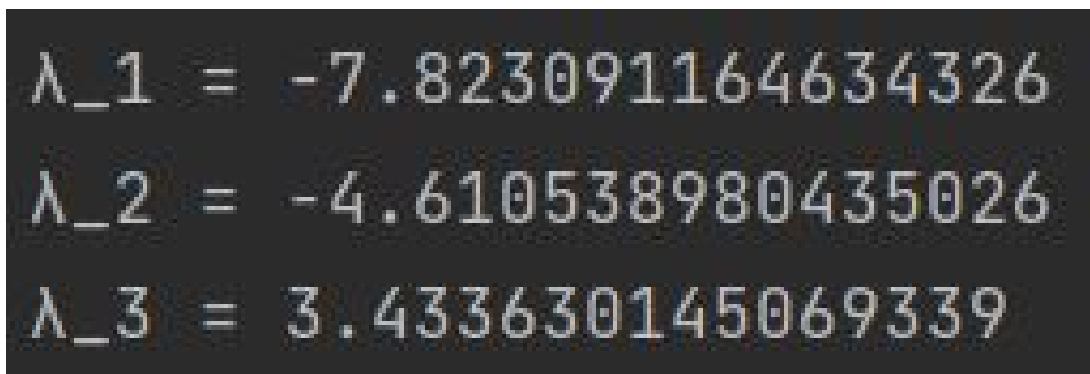
13 Постановка задачи

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: 26

$$\begin{pmatrix} -5 & -8 & 4 \\ 4 & 2 & 6 \\ -2 & 5 & -6 \end{pmatrix}$$

14 Результаты работы



```
λ_1 = -7.823091164634326
λ_2 = -4.610538980435026
λ_3 = 3.433630145069339
```

Рис. 5: Вывод программы в консоли

15 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using matrix = vector<vector<double> >;
5
6
7  void print_matrix(const matrix& matrix1) {
8      for(const auto& vect: matrix1) {
9          for (auto x: vect)
10             cout << x << " ";
11             cout << endl;
12     }
13 }
14
15
16 matrix plus_matrix(const matrix& matrix1, const matrix& matrix2) {
17     matrix res;
18     int n = matrix1.size();
19     for (int i = 0; i < n; i++) {
20         vector<double> row;
21         for (int j = 0; j < n; j++) {
22             row.push_back(matrix1[i][j] + matrix2[i][j]);
23         }
24         res.push_back(row);
25     }
26
27     return res;
28 }
29
30
31 matrix transposed(const matrix& matrix1){
32     int n = matrix1.size(), m = matrix1[0].size();
33     matrix res(m, vector<double>(n));
34     for (int i=0; i<n; i++)
35         for (int j=0; j<m; j++)
36             res[j][i] = matrix1[i][j];
37     return res;
38 }
39
40 matrix multiple_matrix(matrix& matrix1, matrix& matrix2) {
41     int n1 = matrix1.size(), m1 = matrix1[0].size(), m2 = matrix2[0].size();
42     matrix res(n1);
43     for (int i=0; i<n1; i++)
44         for (int j=0; j<m2; j++)
45             res[i].push_back(0);
46
47     for (int i=0; i<n1; i++) {
```

```

48     for (int j=0; j<m2; j++) {
49         double cntr = 0;
50         for (int k=0; k<m1; k++)
51             cntr += matrix1[i][k] * matrix2[k][j];
52         res[i][j] = cntr;
53     }
54 }
55 return res;
56 }
57
58
59 int sign(double a){
60     return (a >= 0) ? 1 : -1;
61 }
62
63
64 double get_eps(const matrix& matrix1){
65     double eps = 0;
66     int n = matrix1.size();
67     for(int i=0; i<n; i++)
68         for(int j=0; j<i-1; j++)
69             eps += matrix1[i][j]*matrix1[i][j];
70     return sqrt(eps);
71 }
72
73
74 matrix get_E_matrix(int n){
75     matrix E(n, vector<double>(n, 0));
76     for(int i=0; i<n; i++)
77         E[i][i] = 1;
78     return E;
79 }
80
81
82 matrix get_H_matrix(const matrix& coefficients, int ind){
83     int n = coefficients.size();
84     matrix v(n, vector<double>(1));
85
86     for(int i=0; i<n; i++){
87         if (i < ind)
88             v[i][0] = 0;
89         else if (i == ind){
90             double sum = 0;
91             for (int j=ind; j < n; j++)
92                 sum += coefficients[j][i]*coefficients[j][i];
93             v[i][0] = coefficients[i][i] + sign(coefficients[i][i]) * sqrt(sum);
94         }
95         else
96             v[i][0] = coefficients[i][ind];

```

```

97     }
98
99     matrix transposed_v = transposed(v);
100     double k = -multiple_matrix(transposed_v, v)[0][0]/2;
101     v = multiple_matrix(v, transposed_v);
102     for (int i=0; i<n; i++)
103         for (int j=0; j<n; j++)
104             v[i][j] /= k;
105
106     matrix E = get_E_matrix(n);
107     return plus_matrix(E, v);
108 }
109
110
111 pair<matrix, matrix> QR_decomposition(const matrix& coeff){
112     matrix coefficients = coeff;
113     matrix Q = get_H_matrix(coefficients, 0);
114     coefficients = multiple_matrix(Q, coefficients);
115     int n = coefficients.size();
116     for (int i=1; i<n-1; i++){
117         matrix H = get_H_matrix(coefficients, i);
118         Q = multiple_matrix(Q, H);
119         coefficients = multiple_matrix(H, coefficients);
120     }
121     return make_pair(Q, coefficients);
122 }
123
124
125 vector<double> get_eigenvalues(matrix& coefficients, double EPS){
126     while (get_eps(coefficients) > EPS){
127         pair<matrix, matrix> QR = QR_decomposition(coefficients);
128         coefficients = multiple_matrix(QR.second, QR.first);
129     }
130     int n = coefficients.size();
131     vector<double> result(n);
132     for (int i=0; i<n; i++)
133         result[i] = coefficients[i][i];
134     return result;
135 }
136
137
138 int main() {
139     matrix coefficient_matrix{
140         {-5, -8, 4},
141         {4, 2, 6},
142         {-2, 5, -6}
143     };
144
145

```



```

146     vector<double> eigenvalues = get_eigenvalues(coefficient_matrix, 0.01);
147     int n = eigenvalues.size();
148
149     for (int i=0; i<n; i++)
150         cout << "lambda_" << i+1 << " = " << eigenvalues[i] << endl;
151
152     return 0;
153 }

```



```

1  from matrix import *
2  from matrix import Vector, Matrix
3
4
5  coefficient_matrix: Matrix
6
7
8  def get_eps(matrix1: Matrix) -> float:
9      eps = 0
10     for i in range(matrix1.n):
11         for j in range(i-1):
12             eps += matrix1[i][j]**2
13     return eps**0.5
14
15
16 def sign(a: float) -> int:
17     return 1 if a >= 0 else -1
18
19
20 def get_H_matrix(coefficients: Matrix, ind: int) -> Matrix:
21     E = Matrix([[1 if i == j else 0 for j in range(coefficients.n)] for i in range(
        coefficients.n)])
22     v = []
23     for i in range(coefficients.n):
24         if i < ind:
25             v.append([0])
26         elif i == ind:
27             v.append([coefficients[i][i] + sign(coefficients[i][i]) * (sum([
                coefficients[j][i]**2 for j in range(ind, coefficients.n)])**0.5)])
28         else:
29             v.append([coefficients[i][ind]])
30     v = Matrix(v)
31     k = -multiple_matrix(v.transposed(), v)[0][0]/2
32     V = multiple_matrix(v, v.transposed())
33     for i in range(V.n):
34         for j in range(V.n):
35             V[i][j] /= k
36     H = plus_matrix(E, V)
37     return H
38
39

```

```

40 def QR_decomposition(coefficients: Matrix) -> tuple[Matrix, Matrix]:
41     coefficients = coefficients.copy()
42     Q = get_H_matrix(coefficients, 0)
43     coefficients = multiple_matrix(Q, coefficients)
44     for i in range(1, coefficients.n-1):
45         H = get_H_matrix(coefficients, i)
46         Q = multiple_matrix(Q, get_H_matrix(coefficients, i))
47         coefficients = multiple_matrix(H, coefficients)
48     return Q, coefficients
49
50
51 def get_eigenvalues(coefficients: Matrix, EPS: float) -> list[float]:
52     while get_eps(coefficients) > EPS:
53         Q, R = QR_decomposition(coefficients)
54         coefficients = multiple_matrix(R, Q)
55     return [coefficients[i][i] for i in range(coefficients.n)]
56
57
58 if __name__ == "__main__":
59     coefficient_matrix = Matrix([
60         [-5, -8, 4],
61         [4, 2, 6],
62         [-2, 5, -6]
63     ])
64
65     eigenvalues = get_eigenvalues(coefficient_matrix, 0.01)
66
67     print(*[f'_{i+1} = {eigenvalues[i]}' for i in range(eigenvalues.__len__())], sep='\n')

```