

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

**Москва, 2024**

## 1.1 LU - разложение матриц

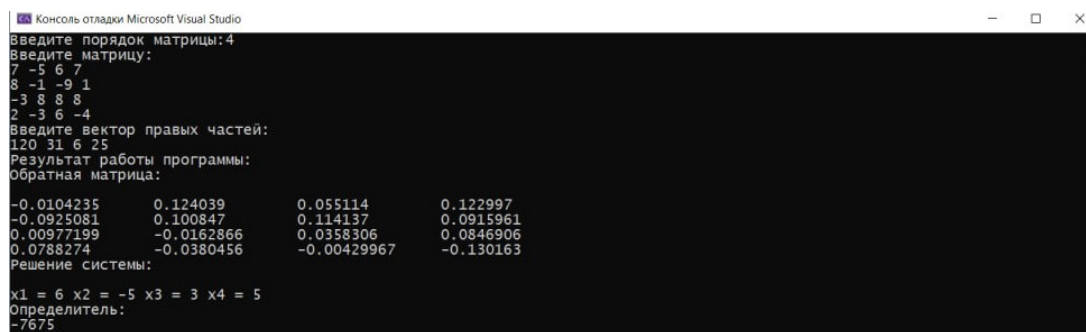
### 1 Постановка задачи

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: 22**

$$\begin{cases} 7x_1 - 5x_2 + 6x_3 + 7x_4 = 120 \\ 8x_1 - 1x_2 - 9x_3 + 1x_4 = 31 \\ -3x_1 + 8x_2 + 8x_3 + 8x_4 = 6 \\ 2x_1 - 3x_2 + 6x_3 - 4x_4 = 25 \end{cases}$$

### 2 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Введите порядок матрицы: 4
Введите матрицу:
7 -5 6 7
8 -1 -9 1
-3 8 8 8
2 -3 6 -4
Введите вектор правых частей:
120 31 6 25
Результат работы программы:
Обратная матрица:
-0.0104235    0.124039    0.055114    0.122987
-0.0925081    0.100847    0.114137    0.0915961
0.00977199    -0.0162866    0.0358306    0.0846906
0.0788274     -0.0380456    -0.00429967   -0.130163
Решение системы:
x1 = 6 x2 = -5 x3 = 3 x4 = 5
Определитель:
-7675
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1  #include <vector>
2  #include <iostream>
3  #include <locale.h>
4  #include "matrix.h"
5
6  using namespace std;
7
8
9  void compute_solution(const Matrix& U, const Matrix& L, const vector<double>& b,
10     vector<double>& x) {
11     x.resize(U.get_m());
12
13     vector<double> z(L.get_m());
14     for (int i = 0; i < L.get_m(); ++i) {
15         z[i] = b[i];
16         for (int j = 0; j < i; ++j) {
17             z[i] -= (L[i][j] * z[j]);
18         }
19
20         for (int i = U.get_m() - 1; i >= 0; --i) {
21             x[i] = z[i];
22             for (int j = i + 1; j < U.get_m(); ++j) {
23                 x[i] -= (x[j] * U[i][j]);
24             }
25             x[i] /= U[i][i];
26         }
27     }
28
29     double gauss_completely(const Matrix& matrix, Matrix& L, Matrix& U, Matrix& X, const
30     vector<double>& b, vector<double>& x) {
31         if (!matrix.is_quadratic()) {
32             throw "LU !";
33         }
34         U = matrix;
35         L = Matrix(matrix.get_n(), matrix.get_m());
36         X = Matrix(matrix.get_n(), matrix.get_m());
37         double determinant = 1.0;
38
39         for (int j = 0; j < U.get_m(); ++j) {
40             if (U[j][j] == 0.0) {
41                 throw "LU !!";
42             }
43             L[j][j] = 1.0;
44             for (int i = j + 1; i < U.get_n(); ++i) {
45                 double l_ij = U[i][j] / U[j][j];
46                 L[i][j] = l_ij;
```

```

46         for (int k = j; k < U.get_m(); ++k) {
47             U[i][k] -= U[j][k] * l_ij;
48         }
49     }
50 }
51
52 compute_solution(U, L, b, x);
53
54 vector<double> b_1(b.size());
55 for (int i = 0; i < U.get_n(); ++i) {
56     b_1[i] = 1.0;
57     compute_solution(U, L, b_1, X[i]);
58     b_1[i] = 0.0;
59 }
60 X.transpose();
61
62 for (int i = 0; i < U.get_m(); ++i) {
63     determinant *= U[i][i];
64 }
65
66 return determinant;
67 }
68
69 int size_init() {
70     int size;
71     cin >> size;
72     return size;
73 }
74
75 void matrix_init(Matrix& A, int size) {
76     A = Matrix(size, size);
77     for (int i = 0; i < size; ++i) {
78         for (int j = 0; j < size; ++j) {
79             cin >> A[i][j];
80         }
81     }
82 }
83
84 void vector_init(vector<double>& b, int size) {
85     b.resize(size);
86     for (int i = 0; i < size; ++i) {
87         cin >> b[i];
88     }
89 }
90
91 void print_vector_x(const vector<double>& x) {
92     for (unsigned i = 0; i < x.size(); ++i) {
93         cout << 'x' << i + 1 << " = " << x[i] << " ";
94     }

```

```

95     cout << endl;
96 }
97
98
99 int main() {
100     setlocale(0, "");
101     Matrix L, U, B, A;
102     vector<double> b, x;
103     cout << "  :";
104     int size = size_init();
105     cout << " : \n";
106     matrix_init(A, size);
107     cout << "  : \n";
108     vector_init(b, size);
109     double determinant = gauss_completely(A, L, U, B, b, x);
110
111     cout << "  : \n";
112     cout << " : \n";
113     cout << B ;
114     cout << " : \n\n";
115     print_vector_x(x);
116     cout << ": \n";
117     cout << determinant;
118     cout << "\n";
119     return 0;
120 }

1  #include "matrix.h"
2
3  const Matrix& Matrix::operator=(const Matrix& right) {
4      _matrix = right._matrix;
5      n_size = right.n_size;
6      m_size = right.m_size;
7      return *this;
8  }
9
10
11 vector<double>& Matrix::operator[](const int index) {
12     return _matrix[index];
13 }
14
15 const vector<double>& Matrix::operator[](const int index) const {
16     return _matrix[index];
17 }
18
19
20 std::ostream& operator<<(std::ostream& os, const Matrix& matrix) {
21     for (int i = 0; i < matrix.n_size; ++i) {
22         os << endl;
23         os.width(8);

```

```

24         os << matrix[i][0];
25         for (int j = 1; j < matrix.m_size; ++j) {
26             os << '\t';
27             os.width(8);
28             os << matrix[i][j];
29         }
30     }
31     os << endl;
32     return os;
33 }
34
35
36 const Matrix operator+(const Matrix& left, const Matrix& right) {
37     if (left.n_size != right.n_size || left.m_size != right.m_size) {
38         throw " !";
39     }
40     Matrix ans(left.n_size, left.m_size);
41     for (int i = 0; i < ans.n_size; ++i) {
42         for (int j = 0; j < ans.m_size; ++j) {
43             ans[i][j] = left._matrix[i][j] + right._matrix[i][j];
44         }
45     }
46     return ans;
47 }
48
49 const Matrix operator-(const Matrix& left, const Matrix& right) {
50     if (left.n_size != right.n_size || left.m_size != right.m_size) {
51         throw " !";
52     }
53     Matrix ans(left.n_size, left.m_size);
54     for (int i = 0; i < ans.n_size; ++i) {
55         for (int j = 0; j < ans.m_size; ++j) {
56             ans[i][j] = left._matrix[i][j] - right._matrix[i][j];
57         }
58     }
59     return ans;
60 }
61
62 const Matrix operator*(double left, const Matrix& right) {
63     Matrix ans = right;
64     for (int i = 0; i < ans.n_size; ++i) {
65         for (int j = 0; j < ans.m_size; ++j) {
66             ans[i][j] *= left;
67         }
68     }
69     return ans;
70 }
71
72 const Matrix operator*(const Matrix& left, double right) {

```

```

73     return right * left;
74 }
75
76
77
78
79 const Matrix operator*(const Matrix& left, const Matrix& right) {
80     if (left.m_size != right.n_size) {
81         throw "   !";
82     }
83     Matrix ans(left.n_size, right.m_size);
84     for (int i = 0; i < ans.n_size; ++i) {
85         for (int j = 0; j < ans.m_size; ++j) {
86             for (int k = 0; k < left.m_size; ++k) {
87                 ans[i][j] += left._matrix[i][k] * right._matrix[k][j];
88             }
89         }
90     }
91     return ans;
92 }
93
94 const vector<double> operator*(const Matrix& left, const vector<double>& right) {
95     if (left.m_size != (int)right.size()) {
96         throw "   !";
97     }
98     vector<double> ans(left.n_size, 0.0);
99     for (int i = 0; i < left.n_size; ++i) {
100         for (int j = 0; j < left.m_size; ++j) {
101             ans[i] += left._matrix[i][j] * right[j];
102         }
103     }
104     return ans;
105 }
106
107
108
109 int Matrix::get_m() const {
110     return m_size;
111 }
112
113 int Matrix::get_n() const {
114     return n_size;
115 }
116
117 void Matrix::make_ones() {
118     if (!is_quadratic()) {
119         throw "   ";
120     }
121     _matrix.assign(n_size, vector<double>(m_size, 0.0));

```

```

122     for (int i = 0; i < n_size; ++i) {
123         _matrix[i][i] = 1.0;
124     }
125 }
126
127 void Matrix::transpose() {
128     vector<vector<double>> temp(m_size, vector<double>(n_size));
129     for (int i = 0; i < n_size; ++i) {
130         for (int j = 0; j < m_size; ++j) {
131             temp[j][i] = _matrix[i][j];
132         }
133     }
134     swap(n_size, m_size);
135     _matrix.swap(temp);
136 }
137
138 Matrix::Matrix() {
139     _matrix.assign(1, vector<double>(1, 0));
140     n_size = m_size = 1;
141 }
142
143 Matrix::Matrix(int n, int m) {
144     _matrix.assign(n, vector<double>(m, 0));
145     n_size = n;
146     m_size = m;
147 }
148
149
150 bool Matrix::is_quadratic() const {
151     return n_size == m_size;
152 }
153
154
155 bool Matrix::is_three_diagonal() const {
156     if (!is_quadratic()) {
157         return false;
158     }
159     for (int i = 0; i < n_size; ++i) {
160         for (int j = 0; j < m_size; ++j) {
161             if ((abs(i - j) > 1) && _matrix[i][j]) {
162                 return false;
163             }
164         }
165     }
166     return true;
167 }
168
169 bool Matrix::is_simmetric() const {
170     if (!is_quadratic()) {

```



```

171         return false;
172     }
173     for (int i = 0; i < n_size; ++i) {
174         for (int j = i + 1; j < m_size; ++j) {
175             if (_matrix[i][j] != _matrix[j][i]) {
176                 return false;
177             }
178         }
179     }
180     return true;
181 }
182
183 double Matrix::get_norm() const {
184     double max = 0.0;
185     for (int i = 0; i < n_size; ++i) {
186         double ans = 0.0;
187         for (int j = 0; j < m_size; ++j) {
188             ans += abs(_matrix[i][j]);
189         }
190         max = max > ans ? max : ans;
191     }
192     return max;
193 }
194
195 double Matrix::get_upper_norm() const {
196     double max = 0.0;
197     for (int i = 0; i < n_size; ++i) {
198         double ans = 0.0;
199         for (int j = 0; j <= i; ++j) {
200             ans += abs(_matrix[i][j]);
201         }
202         max = max > ans ? max : ans;
203     }
204     return max;
205 }

```

```

1  #pragma once
2  #ifndef MATRIX_H
3  #define MATRIX_H
4
5  #include <vector>
6  #include <iostream>
7  #include <cmath>
8
9
10 using namespace std;
11
12 class Matrix {
13 public:
14     Matrix();

```

```

15     Matrix(int n, int m);
16
17     void make_ones();
18     void transpose();
19
20     vector<double>& operator[](const int index);
21     const vector<double>& operator[](const int index) const;
22
23     friend const Matrix operator+(const Matrix& left, const Matrix& right);
24     friend const Matrix operator-(const Matrix& left, const Matrix& right);
25
26     friend const Matrix operator*(const Matrix& left, const Matrix& right);
27     friend const vector<double> operator*(const Matrix& left, const vector<double>&
        right);
28     friend const Matrix operator*(const Matrix& left, double right);
29     friend const Matrix operator*(double left, const Matrix& right);
30
31     const Matrix& operator=(const Matrix& right);
32
33     friend std::ostream& operator<<(std::ostream& os, const Matrix& matrix);
34
35     double get_norm() const;
36     double get_upper_norm() const;
37
38     int get_n() const;
39     int get_m() const;
40
41     bool is_three_diagonal() const;
42     bool is_simmetric() const;
43
44     bool is_quadratic() const;
45
46 private:
47     vector<vector<double>>> _matrix;
48     int n_size;
49     int m_size;
50 };
51
52 #endif

```