

**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

1.3 Метод простых итераций. Метод Зейделя

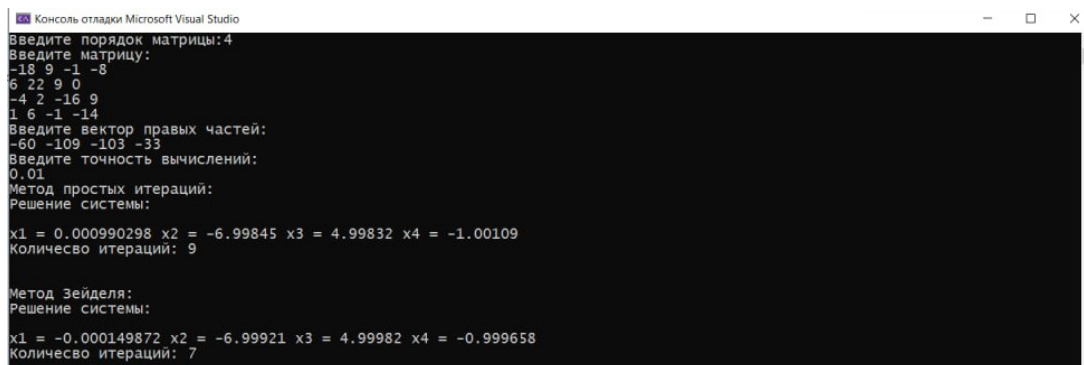
1 Постановка задачи

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: 22

$$\begin{cases} -18x_1 + 9x_2 - 1x_3 - 8x_4 = -60 \\ 6x_1 + 22x_2 + 9x_3 = -109 \\ -4x_1 + 2x_2 - 16x_3 + 9x_4 = -103 \\ 1x_1 + 6x_2 - 1x_3 - 14x_4 = -33 \end{cases}$$

2 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Введите порядок матрицы: 4
Введите матрицу:
-18 9 -1 -8
6 22 9 0
-4 2 -16 9
1 6 -1 -14
Введите вектор правых частей:
-60 -109 -103 -33
Введите точность вычислений:
0.01
Метод простых итераций:
Решение системы:
x1 = 0.000990298 x2 = -6.99845 x3 = 4.99832 x4 = -1.00109
Количество итераций: 9

Метод Зейделя:
Решение системы:
x1 = -0.000149872 x2 = -6.99921 x3 = 4.99982 x4 = -0.999658
Количество итераций: 7
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1 | #include <vector>
2 | #include <iostream>
3 | #include <locale.h>
4 | #include "matrix.h"
5 | #include <cmath>
6 |
7 | using namespace std;
8 |
9 | int size_init() {
10 |     int size;
11 |     cin >> size;
12 |     return size;
13 | }
14 |
15 | void matrix_init(Matrix& A, int size) {
16 |     A = Matrix(size, size);
17 |     for (int i = 0; i < size; ++i) {
18 |         for (int j = 0; j < size; ++j) {
19 |             cin >> A[i][j];
20 |         }
21 |     }
22 | }
23 |
24 | void vector_init(vector<double>& b, int size) {
25 |     b.resize(size);
26 |     for (int i = 0; i < size; ++i) {
27 |         cin >> b[i];
28 |     }
29 | }
30 |
31 | void print_vector_x(const vector<double>& x) {
32 |     for (unsigned i = 0; i < x.size(); ++i) {
33 |         cout << 'x' << i + 1 << " = " << x[i] << " ";
34 |     }
35 |     cout << endl;
36 | }
37 |
38 | vector<double> vector_minus(const vector<double>& a, const vector<double>& b) {
39 |     vector<double> minus = a;
40 |     for (unsigned i = 0; i < minus.size(); ++i) {
41 |         minus[i] -= b[i];
42 |     }
43 |     return minus;
44 | }
45 |
46 | vector<double> vector_plus(const vector<double>& a, const vector<double>& b) {
47 |     vector<double> plus = a;
```

```

48     for (unsigned i = 0; i < plus.size(); ++i) {
49         plus[i] += b[i];
50     }
51     return plus;
52 }
53
54 double norm_of_vector(const vector<double>& vec) {
55     double norm = 0.0;
56     for (unsigned i = 0; i < vec.size(); ++i) {
57         norm += vec[i] * vec[i];
58     }
59     return sqrt(norm);
60 }
61
62 int simple_itteration(const Matrix& A, const vector<double>& b, vector<double>& x,
63     double alfa) {
64     Matrix M = A;
65     x.resize(b.size());
66     vector<double> last(b.size(), 0.0), r = b;
67     double coeff = 0.0;
68     if (!M.is_quadratic()) {
69         throw " !";
70     }
71     for (int i = 0; i < M.get_n(); ++i) {
72         if (!A[i][i]) {
73             throw " !";
74         }
75         for (int j = 0; j < M.get_m(); ++j) {
76             M[i][j] = i == j ? 0.0 : -A[i][j] / A[i][i];
77         }
78         r[i] /= A[i][i];
79     }
80
81     x = r;
82     coeff = M.get_norm();
83     if (coeff < 1.0) {
84         coeff /= 1 - coeff;
85     }
86     else {
87         coeff = 1.0;
88     }
89
90     int itter = 0;
91
92     for (itter = 0; coeff * norm_of_vector(vector_minus(x, last)) > alfa; ++itter) {
93         x.swap(last);
94         x = vector_plus(r, M * last);
95     }

```

```

96
97     return itter;
98 }
99
100 int zeidels_method(const Matrix& A, const vector<double>& b, vector<double>& x, double
    alfa) {
101     Matrix M = A;
102     x.resize(b.size());
103     vector<double> last(b.size(), 0.0), r = b;
104     double coeff = 0.0;
105
106     if (!M.is_quadratic()) {
107         throw " !";
108     }
109     for (int i = 0; i < M.get_n(); ++i) {
110         if (!A[i][i]) {
111             throw " !";
112         }
113         for (int j = 0; j < M.get_m(); ++j) {
114             M[i][j] = i == j ? 0.0 : -A[i][j] / A[i][i];
115         }
116         r[i] /= A[i][i];
117     }
118
119     x = r;
120
121     coeff = M.get_norm();
122     if (coeff < 1) {
123         coeff = M.get_upper_norm() / (1 - coeff);
124     }
125     else {
126         coeff = 1.0;
127     }
128
129     int itter = 0;
130
131     for (itter = 0; coeff * norm_of_vector(vector_minus(x, last)) > alfa; ++itter) {
132         x.swap(last);
133         x = r;
134         for (int i = 0; i < M.get_n(); ++i) {
135             for (int j = 0; j < i; ++j) {
136                 x[i] += x[j] * M[i][j];
137             }
138             for (int j = i; j < M.get_m(); ++j) {
139                 x[i] += last[j] * M[i][j];
140             }
141         }
142     }
143

```

```

144     return itter;
145 }
146
147 void print_solution(const vector<double>& x, int itter) {
148     cout << " : \n" << endl;
149     print_vector_x(x);
150     cout << " : " << itter << endl;
151     cout << "\n" << endl;
152 }
153
154 int main() {
155     setlocale(0, "");
156     Matrix A;
157     vector<double> x, b;
158     vector<vector<double>> vec;
159     double accuracy = 0.001;
160     cout << " : ";
161     int size = size_init();
162
163     cout << " : \n";
164     matrix_init(A, size);
165     cout << " : \n";
166     vector_init(b, size);
167     cout << " : \n";
168     cin >> accuracy;
169
170     cout << " : " << endl;
171     int itter = simple_iteration(A, b, x, accuracy);
172     print_solution(x, itter);
173
174     cout << " : " << endl;
175     itter = zeidels_method(A, b, x, accuracy);
176     print_solution(x, itter);
177
178     return 0;
179 }

```



```

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

```