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

1.2 Метод прогонки

1 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: 22

$$\begin{cases} -14x_1 + 6x_2 = 82 \\ 2x_1 + 7x_2 = -51 \\ -7x_2 - 18x_3 - 9x_4 = -46 \\ 2x_3 - 13x_4 + 2x_5 = 111 \\ -7x_4 - 7x_5 = 35 \end{cases}$$

2 Результаты работы

Рис. 1: Вывод программы в консоли

3 Исходный код

```cpp
#include <vector>
#include <iostream>
#include <locale.h>
#include "matrix.h"

using namespace std;

int size_init() {
    int size;
    cin >> size;
    return size;
}

void matrix_init(Matrix& A, int size) {
    A = Matrix(size, size);
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            cin >> A[i][j];
        }
    }
}

void vector_init(vector<double>& b, int size) {
    b.resize(size);
    for (int i = 0; i < size; ++i) {
        cin >> b[i];
    }
}


void print_vector_x(const vector<double>& x) {
    for (unsigned i = 0; i < x.size(); ++i) {
        cout << 'x' << i + 1 << " = " << x[i] << " ";
    }
    cout << endl;
}

void matrix_to_vecs(const Matrix& matrix, vector<vector<double>>& vec) {
    vec.clear();
    vec.assign(matrix.get_n(), vector<double>(3, 0.0));

    for (int i = 0; i < matrix.get_n(); ++i) {
        vec[i][0] = i - 1 < 0 ? 0.0 : matrix[i][i - 1];
        vec[i][1] = matrix[i][i];
        vec[i][2] = i + 1 < matrix.get_m() ? matrix[i][i + 1] : 0.0;
    }
}
```

```
48
49 void race_method(vector<vector<double>>& vec, const vector<double>& b, vector<double>&
       x) {
50     x.assign(b.size(), 0.0);
51     vector<double> P(b.size()), Q(b.size());
52     P[0] = -vec[0][2] / vec[0][1];
53     Q[0] = b[0] / vec[0][1];
54
55     for (int i = 1; i < (int)x.size(); ++i) {
56         double z = (vec[i][1] + vec[i][0] * P[i - 1]);
57         P[i] = -vec[i][2];
58         P[i] /= z;
59         Q[i] = (b[i] - vec[i][0] * Q[i - 1]);
60         Q[i] /= z;
61     }
62     x.back() = Q.back();
63     for (int i = x.size() - 2; i >= 0; --i) {
64         x[i] = P[i] * x[i + 1] + Q[i];
65     }
66 }
67
68
69 int main() {
70     setlocale(0, "");
71     Matrix A;
72     vector<double> x, b;
73     vector<vector<double>> vec;
74     std::cout << "  : \n";
75     std::cout << " L = a b 0 0 0 \n";
76     std::cout << " c d e 0 0 \n";
77     std::cout << " 0 f g h 0 \n";
78     std::cout << " 0 0 i j k \n";
79     std::cout << " 0 0 0 l m \n";
80     cout << "  :";
81     int size = size_init();
82     cout << " : \n";
83     matrix_init(A, size);
84     cout << "   : \n";
85     vector_init(b, size);
86     if (A.is_three_diagonal()) {
87         matrix_to_vecs(A, vec);
88     }
89     else {
90         cout << "!    !" << endl;
91         return 0;
92     }
93     race_method(vec, b, x);
94
95     cout << " : \n";
```

```
    print_vector_x(x);

}
```

```
#include "matrix.h"

const Matrix& Matrix::operator=(const Matrix& right) {
    _matrix = right._matrix;
    n_size = right.n_size;
    m_size = right.m_size;
    return *this;
}


vector<double>& Matrix::operator[](const int index) {
    return _matrix[index];
}

const vector<double>& Matrix::operator[](const int index) const {
    return _matrix[index];
}


std::ostream& operator<<(std::ostream& os, const Matrix& matrix) {
    for (int i = 0; i < matrix.n_size; ++i) {
        os << endl;
        os.width(8);
        os << matrix[i][0];
        for (int j = 1; j < matrix.m_size; ++j) {
            os << '\t';
            os.width(8);
            os << matrix[i][j];
        }
    }
    os << endl;
    return os;
}


const Matrix operator+(const Matrix& left, const Matrix& right) {
    if (left.n_size != right.n_size || left.m_size != right.m_size) {
        throw "   !";
    }
    Matrix ans(left.n_size, left.m_size);
    for (int i = 0; i < ans.n_size; ++i) {
        for (int j = 0; j < ans.m_size; ++j) {
            ans[i][j] = left._matrix[i][j] + right._matrix[i][j];
        }
    }
    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
```