

**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

1.4 Метод вращений

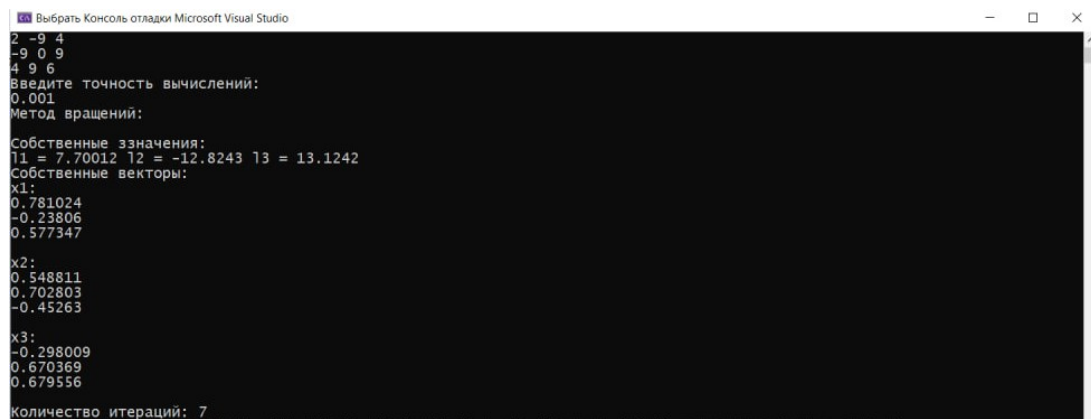
1 Постановка задачи

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: 22

$$\begin{pmatrix} 2 & -9 & 4 \\ -9 & 0 & 9 \\ 4 & 9 & 6 \end{pmatrix}$$

2 Результаты работы



```
Выбрать Консоль отладки Microsoft Visual Studio
2 -9 4
-9 0 9
4 9 6
Введите точность вычислений:
0.001
Метод вращений:
Собственные значения:
l1 = 7.70012 l2 = -12.8243 l3 = 13.1242
Собственные векторы:
x1:
0.781024
-0.23806
0.577347
x2:
0.548811
0.702803
-0.45263
x3:
-0.298009
0.670369
0.679556
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 | #define M_PI_4 0.785398163397448309616
7 |
8 | using namespace std;
9 |
10 | int size_init() {
11 |     int size;
12 |     cin >> size;
13 |     return size;
14 | }
15 |
16 | void matrix_init(Matrix& A, int size) {
17 |     A = Matrix(size, size);
18 |     for (int i = 0; i < size; ++i) {
19 |         for (int j = 0; j < size; ++j) {
20 |             cin >> A[i][j];
21 |         }
22 |     }
23 | }
24 |
25 | void print_vector_x(const vector<double>& x) {
26 |     for (unsigned i = 0; i < x.size(); ++i) {
27 |         cout << 'l' << i + 1 << " = " << x[i] << " ";
28 |     }
29 |     cout << endl;
30 | }
31 |
32 |
33 | int rotate_method(const Matrix& A, Matrix& U, vector<double>& x, double alfa) {
34 |     if (!A.is_simmetric()) {
35 |         throw " !";
36 |     }
37 |
38 |     Matrix U_k(A.get_n(), A.get_m()), A_k = A;
39 |     int i_max = 0, j_max = 0, itter = 0;
40 |     double v_max = 0.0, fitta = 0.0, check = 0.0;
41 |
42 |     U = Matrix(A.get_n(), A.get_m());
43 |     U.make_ones();
44 |     x.resize(A.get_m());
45 |
46 |     do {
47 |         U_k.make_ones();
```

```

48
49     v_max = 0.0;
50     i_max = j_max = 0;
51     for (int i = 0; i < A_k.get_n(); ++i) {
52         for (int j = i + 1; j < A_k.get_m(); ++j) {
53             if (v_max < abs(A_k[i][j])) {
54                 v_max = abs(A_k[i][j]);
55                 i_max = i;
56                 j_max = j;
57             }
58         }
59     }
60
61     fitta = A_k[i_max][i_max] == A_k[j_max][j_max] ?
62         M_PI_4 :
63         atan(2 * A_k[i_max][j_max] / (A_k[i_max][i_max] - A_k[j_max][j_max])) / 2;
64
65     U_k[i_max][j_max] = -sin(fitta);
66     U_k[i_max][i_max] = cos(fitta);
67     U_k[j_max][j_max] = cos(fitta);
68     U_k[j_max][i_max] = sin(fitta);
69
70     U = U * U_k;
71
72     A_k = A_k * U_k;
73     U_k.transpose();
74     A_k = U_k * A_k;
75
76     check = 0.0;
77     for (int i = 0; i < A_k.get_n(); ++i) {
78         for (int j = i + 1; j < A_k.get_m(); ++j) {
79             check += A_k[i][j] * A_k[i][j];
80         }
81     }
82     check = sqrt(check);
83     ++itter;
84 } while (check > alfa);
85
86 for (int i = 0; i < A_k.get_n(); ++i) {
87     x[i] = A_k[i][i];
88 }
89
90 return itter;
91 }
92
93 int main() {
94     setlocale(0, "");
95     Matrix A, U;
96     vector<double> x;

```

```

97     double accuracy = 0.01;
98     cout << " :";
99     int size = size_init();
100    cout << " : \n";
101    matrix_init(A, size);
102    cout << " : \n";
103    cin >> accuracy;
104
105    cout << " : \n" << endl;
106    int itter = rotate_method(A, U, x, accuracy);
107    cout << " : \n";
108    print_vector_x(x);
109    cout << " : \n";
110    for (int j = 0; j < U.get_m(); ++j) {
111        cout << "x" << j + 1 << " : " << endl;
112        for (int i = 0; i < U.get_n(); ++i) {
113            cout.width(8);
114            cout << U[i][j] << endl;
115        }
116        cout << endl;
117    }
118    cout << " : " << itter;
119
120    return 0;
121 }

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

```