Московский авиационный институт
(национальный исследовательский университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 1 по курсу «Численные методы»

Студент: Н. О. Тимофеева
Преподаватель: Д. Е. Пивоваров
Группа: М8О-308Б-19
Вариант: 19
Дата:
Оценка:

Москва, 2022

Лабораторная работа 1

Методы решения задач линейной алгебры

1.1.

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.

Исходный код

```
template<class T>
class lu_t {
private:
    const T EPS = 0.000001;
    matrix_t<T> l;
    matrix_t<T> u;
    T det;
    vector<pair<size_t, size_t>> swaps;

    void do_swaps(vector<T> & x) {
        for (pair<size_t, size_t> elem : swaps) {
            swap(x[elem.first], x[elem.second]);
        }
    }

    void decompose() {
        size_t n = u.rows();
        for (size_t i = 0; i < n; ++i) {
            size_t max_el_ind = i;
            for (size_t j = i + 1; j < n; ++j) {
                if (abs(u[j][i]) > abs(u[max_el_ind][i])) {
                    max_el_ind = j;
                }
            }
            if (max_el_ind != i) {
                pair<size_t, size_t> perm = make_pair(i, max_el_ind);
                swaps.push_back(perm);
                u.swap_rows(i, max_el_ind);
                l.swap_rows(i, max_el_ind);
                l.swap_cols(i, max_el_ind);
            }
            for (size_t j = i + 1; j < n; ++j) {
                if (abs(u[i][i]) < EPS) {
                    continue;
                }
                T mu = u[j][i] / u[i][i];
                l[j][i] = mu;
                for (size_t k = 0; k < n; ++k) {
                    u[j][k] -= mu * u[i][k];
                }
            }
        }
        det = (swaps.size() & 1 ? -1 : 1);
        for (size_t i = 0; i < n; ++i) {
            det *= u[i][i];
```

```
        }
    }
public:
    lu_t(const matrix_t<T> & matr) {
        if (matr.rows() != matr.cols()) {
            throw invalid_argument("Matrix is not square");
        }
        l = matrix_t<T>::identity(matr.rows());
        u = matrix_t<T>(matr);
        decompose();
    }

    vector<T> solve(vector<T> b) {
        int n = b.size();
        do_swaps(b);
        vector<T> z(n);
        for (int i = 0; i < n; ++i) {
            T summary = b[i];
            for (int j = 0; j < i; ++j) {
                summary -= z[j] * l[i][j];
            }
            z[i] = summary;
        }
        vector<T> x(n);
        for (int i = n - 1; i >= 0; --i) {
            if (abs(u[i][i]) < EPS) {
                continue;
            }
            T summary = z[i];
            for (int j = n - 1; j > i; --j) {
                summary -= x[j] * u[i][j];
            }
            x[i] = summary / u[i][i];
        }
        return x;
    }

    T get_det() {
        return det;
    }

    matrix_t<T> inv_matrix() {
        size_t n = l.rows();
        matrix_t<T> res(n);
        for (size_t i = 0; i < n; ++i) {
            vector<T> b(n);
            b[i] = T(1);
            vector<T> x = solve(b);
            for (size_t j = 0; j < n; ++j) {
                res[j][i] = x[j];
            }
        }
        return res;
    }

    friend ostream & operator << (ostream & out, const lu_t<T> & lu) {
        out << "Matrix L:\n" << lu.l << "\nMatrix U:\n" << lu.u << '\n';
        return out;
    }

    ~lu_t() = default;
};
```

```cpp

int main() {
    cout.precision(5);
    cout << fixed;
    int n;
    cin >> n;
    matrix_t<double> a(n);
    cin >> a;
    lu_t<double> lu_a(a);
    vector<double> b(n);
    for (int i = 0; i < n; ++i) {
        cin >> b[i];
    }
    vector<double> x = lu_a.solve(b);
    cout << lu_a;
    cout << " :" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "x" << i + 1 << " = " << x[i] << endl;
    }
    cout << "\ndet(A) = " << lu_a.get_det() << endl;
    cout << "\nA^(-1): " << endl;
    cout << lu_a.inv_matrix();
}
```

Входные данные

```
4
-8 5 8 -6
2 7 -8 -1
-5 -4 1 -6
5 -9 -2 8
-144 25 -21 103
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-1$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-1$ ./a.out <test.txt
Matrix L:
1.00000,0.00000,0.00000,0.00000
-0.25000,1.00000,0.00000,0.00000
0.62500,-0.86364,1.00000,0.00000
-0.62500,-0.71212,0.13861,1.00000

Matrix U:
-8.00000,5.00000,8.00000,-6.00000
0.00000,8.25000,-6.00000,-2.50000
0.00000,0.00000,-9.18182,-4.40909
0.00000,0.00000,0.00000,3.08086
```

```
Решение системы:
x1 = 9.00000
x2 = -6.00000
x3 = -6.00000
x4 = -1.00000

det(A) = 1867.00000

A^(-1):
-0.39261,-0.30316,-0.10177,-0.40868
0.04981,0.04392,-0.07713,-0.01500
-0.08945,-0.18640,-0.08731,-0.15587
0.27906,0.19229,-0.04499,0.32458
```

1.2.

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.

Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

template<class T>
class tridiag_t {
private:
    const T EPS = 0.000001;
    int n;
    vector<T> a;
    vector<T> b;
    vector<T> c;
public:
    tridiag_t(const int & _n) : n(_n), a(n), b(n), c(n) {}
    tridiag_t(const vector<T> & _a, const vector<T> & _b, const vector<T> & _c) {
        if (!(_a.size() == _b.size() and _a.size() == _c.size())) {
            throw invalid_argument("Sizes of a, b, c are invalid");
        }
        n = _a.size();
        a = _a;
        b = _b;
        c = _c;
    }
    vector<T> solve(const vector<T> & d) {
        int m = d.size();
        if (n != m) {
            throw invalid_argument("Size of vector d is invalid");
        }
        vector<T> p(n);
        p[0] = -c[0] / b[0];
        vector<T> q(n);
        q[0] = d[0] / b[0];
        for (int i = 1; i < n; ++i) {
            p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
            q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
        }
        vector<T> x(n);
        x.back() = q.back();
        for (int i = n - 2; i >= 0; --i) {
            x[i] = p[i] * x[i + 1] + q[i];
        }
        return x;
    }
    friend istream & operator >> (istream & in, tridiag_t<T> & tridiag) {
        in >> tridiag.b[0] >> tridiag.c[0];
        for (int i = 1; i < tridiag.n - 1; ++i) {
            in >> tridiag.a[i] >> tridiag.b[i] >> tridiag.c[i];
        }
        in >> tridiag.a.back() >> tridiag.b.back();
```

```
51          return in;
52      }
53      ~tridiag_t() = default;
54  };
55
56  int main() {
57      cout.precision(5);
58      cout << fixed;
59      int n;
60      cin >> n;
61      tridiag_t<double> tridiag_a(n);
62      cin >> tridiag_a;
63      vector<double> b(n);
64      for (int i = 0; i < n; ++i) {
65          cin >> b[i];
66      }
67      vector<double> x = tridiag_a.solve(b);
68      cout << " :" << endl;
69      for (int i = 0; i < n; ++i) {
70          cout << "x" << i + 1 << " = " << x[i] << endl;
71      }
72  }
```

Входные данные

```
5
10 -1
-8 16 1
6 -16 6
-8 16 -5
5 -13
16 -110 24 -3 87
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-2$ ./a.out <test.txt
Решение системы:
x1 = 1.00000
x2 = -6.00000
x3 = -6.00000
x4 = -6.00000
x5 = -9.00000
```

1.3.

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.

Исходный код

```
class solver_si_ze {
private:
    matrix_t<double> a;
    size_t n;
    double eps;
    static constexpr double INF = 1e18;
public:
    int iter_count;

    solver_si_ze(const matrix_t<double> & _a, double _eps = 0.000001) {
        if (_a.rows() != _a.cols()) {
            throw invalid_argument("Matrix is not square");
        }
        a = matrix_t<double>(_a);
        n = a.rows();
        eps = _eps;
    }

    static double norm(const vector<double> & v) {
        double res = -INF;
        for (double elem : v) {
            res = max(res, abs(elem));
        }
        return res;
    }

    static double norm(const matrix_t<double> & m) {
        double res = -INF;
        for (size_t i = 0; i < m.rows(); ++i) {
            double s = 0;
            for (double elem : m[i]) {
                s += abs(elem);
            }
            res = max(res, s);
        }
        return res;
    }

    pair<matrix_t<double>, vector<double>> precalc_ab(const vector<double> & b, matrix_t<double> & alpha,
         vector<double> & beta) {
        for (size_t i = 0; i < n; ++i) {
            beta[i] = b[i] / a[i][i];
            for (size_t j = 0; j < n; ++j) {
                if (i != j) {
                    alpha[i][j] = -a[i][j] / a[i][i];
                }
            }
        }
        return make_pair(alpha, beta);
```

```
49      }
50
51      vector<double> solve_simple(const vector<double> & b) {
52          matrix_t<double> alpha(n);
53          vector<double> beta(n);
54          precalc_ab(b, alpha, beta);
55          double eps_coef = 1.0;
56          if (norm(alpha) - 1.0 < eps) {
57              eps_coef = norm(alpha) / (1.0 - norm(alpha));
58          }
59          double eps_k = 1.0;
60          vector<double> x(beta);
61          iter_count = 0;
62          while (eps_k > eps) {
63              vector<double> x_k = beta + alpha * x;
64              eps_k = eps_coef * norm(x_k - x);
65              x = x_k;
66              ++iter_count;
67          }
68          return x;
69      }
70
71      vector<double> zeidel(const vector<double> & x, const matrix_t<double> & alpha, const vector<double> &
            beta) {
72          vector<double> x_k(beta);
73          for (size_t i = 0; i < n; ++i) {
74              for (size_t j = 0; j < i; ++j) {
75                  x_k[i] += x_k[j] * alpha[i][j];
76              }
77              for (size_t j = i; j < n; ++j) {
78                  x_k[i] += x[j] * alpha[i][j];
79              }
80          }
81          return x_k;
82      }
83
84      vector<double> solve_zeidel(const vector<double> & b) {
85          matrix_t<double> alpha(n);
86          vector<double> beta(n);
87          precalc_ab(b, alpha, beta);
88          matrix_t<double> c(n);
89          for (size_t i = 0; i < n; ++i) {
90              for (size_t j = i; j < n; ++j) {
91                  c[i][j] = alpha[i][j];
92              }
93          }
94          double eps_coef = 1.0;
95          if (norm(alpha) - 1.0 < eps) {
96              eps_coef = norm(c) / (1.0 - norm(alpha));
97          }
98          double eps_k = 1.0;
99          vector<double> x(beta);
100         iter_count = 0;
101         while (eps_k > eps) {
102             vector<double> x_k = zeidel(x, alpha, beta);
103             eps_k = eps_coef * norm(x_k - x);
104             x = x_k;
105             ++iter_count;
106         }
107         return x;
108     }
109
```

```
110     ~solver_si_ze() = default;
111 };
112 
113 int main() {
114     cout.precision(5);
115     cout << fixed;
116     int n;
117     double eps;
118     cin >> n >> eps;
119     matrix_t<double> a(n);
120     cin >> a;
121     solver_si_ze solver(a, eps);
122     vector<double> b(n);
123     for (int i = 0; i < n; ++i) {
124         cin >> b[i];
125     }
126     vector<double> x = solver.solve_simple(b);
127     cout << "  " << endl;
128     for (int i = 0; i < n; ++i) {
129         cout << "x" << i + 1 << " = " << x[i] << endl;
130     }
131     cout << "   " << solver.iter_count << " " << "\n\n";
132     vector<double> ze = solver.solve_zeidel(b);
133     cout << " " << endl;
134     for (int i = 0; i < n; ++i) {
135         cout << "x" << i + 1 << " = " << ze[i] << endl;
136     }
137     cout << "   " << solver.iter_count << " " << endl;
138 }
```

Входные данные

```
4 0.000001
15 0 7 5
-3 -14 -6 1
-2 9 13 2
4 -1 3 9
176 -111 74 76
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-3$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-3$ ./a.out <test.txt
Метод простых итераций
x1 = 9.00000
x2 = 5.00000
x3 = 3.00000
x4 = 4.00000
Решени получено за 88 итераций
```

```
Метод Зейделя
x1 = 9.00000
x2 = 5.00000
x3 = 3.00000
x4 = 4.00000
Решени получено за 31 итераций
```

1.4.

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.

Исходный код

```
class rotation {
private:
    static constexpr double GLOBAL_EPS = 1e-9;
    size_t n;
    matrix_t<double> a;
    matrix_t<double> v;
    double eps;

    matrix_t<double> create_rotation(size_t i, size_t j, double phi) {
        matrix_t<double> u = matrix_t<double>::identity(n);
        u[i][i] = cos(phi);
        u[i][j] = -sin(phi);
        u[j][i] = sin(phi);
        u[j][j] = cos(phi);
        return u;
    }

    double calc_phi(size_t i, size_t j) {
        if (abs(a[i][i] - a[j][j]) < GLOBAL_EPS) {
            return atan2(1.0, 1.0);
        }
        else {
            return 0.5 * atan2(2 * a[i][j], a[i][i] - a[j][j]);
        }
    }

    static double norm(const matrix_t<double> & m) {
        double res = 0;
        for (size_t i = 0; i < m.rows(); ++i) {
            for (size_t j = 0; j < m.cols(); ++j) {
                if (i == j) {
                    continue;
                }
                res += m[i][j] * m[i][j];
            }
        }
        return sqrt(res);
    }

    void build() {
        iter_count = 0;
        while (norm(a) > eps) {
            ++iter_count;
            size_t i = 0, j = 1;
            for (size_t ii = 0; ii < n; ++ii) {
                for (size_t jj = 0; jj < n; ++jj) {
                    if (ii == jj) {
                        continue;
                    }
```

```
                if (abs(a[ii][jj]) > abs(a[i][j])) {
                    i = ii;
                    j = jj;
                }
            }
        }
        double phi = calc_phi(i, j);
        matrix_t<double> u = create_rotation(i, j, phi);
        v = v * u;
        a = u.t() * a * u;
    }
}
public:
    int iter_count;

    rotation(const matrix_t<double> & _a, double _eps) {
        if (_a.rows() != _a.cols()) {
            throw invalid_argument("Matrix is not square");
        }
        a = matrix_t<double>(_a);
        n = a.rows();
        eps = _eps;
        v = matrix_t<double>::identity(n);
        build();
    };

    vector<double> get_values() {
        vector<double> res(n);
        for (size_t i = 0; i < n; ++i) {
            res[i] = a[i][i];
        }
        return res;
    }

    matrix_t<double> get_vectors() {
        return v;
    }

    ~rotation() = default;
};

int main() {
    cout.precision(6);
    cout << fixed;
    int n;
    double eps;
    cin >> n >> eps;
    matrix_t<double> a(n);
    cin >> a;
    rotation rot(a, eps);
    vector<double> lambda = rot.get_values();
    cout << " :" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "l_" << i + 1 << " = " << lambda[i] << endl;
    }
    matrix_t<double> v = rot.get_vectors();
    cout << "\n :" << endl << v;
    cout << "\n   " << rot.iter_count << " " << endl;
}
```

Входные данные

```
3 0.000001
-8 9 6
9 9 1
6 1 8
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-4$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-4$ ./a.out <test.txt
Собственные значения:
l_1 = -13.141391
l_2 = 7.384014
l_3 = 14.757377

Собственные векторы:
-0.903164,-0.005498,-0.429261
0.356301,0.548168,-0.756677
0.239468,-0.836350,-0.493127

Решение получено за 7 итераций
```

1.5.

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.

Исходный код

```
class qr {
private:
    static constexpr double INF = 1e18;
    static constexpr std::complex<double> COMPLEX_INF = std::complex<double>(INF, INF);
    size_t n;
    double eps;
    matrix_t<double> a;
    vector<std::complex<double>> eigen;

    double sign(double x) {
        if (x < eps) {
            return -1.0;
        }
        else if (x > eps) {
            return 1.0;
        }
        else {
            return 0.0;
        }
    }

    matrix_t<double> vvt(const vector<double> & b) {
        size_t n_b = b.size();
        matrix_t<double> res(n_b);
        for (size_t i = 0; i < n_b; ++i) {
            for (size_t j = 0; j < n_b; ++j) {
                res[i][j] = b[i] * b[j];
            }
        }
        return res;
    }

    double vtv(const vector<double> & v) {
        double res = 0;
        for (double elem : v) {
            res += elem * elem;
        }
        return res;
    }

    double norm(const vector<double> & v) {
        return sqrt(vtv(v));
    }

    matrix_t<double> householder(const vector<double> & b, int id) {
        vector<double> v(b);
        v[id] += sign(b[id]) * norm(b);
        return matrix_t<double>::identity(n) - (2.0 / vtv(v)) * vvt(v);
    }
```

```
50
51      pair<std::complex<double>, std::complex<double>> solve_sq(double a11, double a12, double a21, double
            a22) {
52          double a = 1.0;
53          double b = -(a11 + a22);
54          double c = a11 * a22 - a12 * a21;
55          double d_sq = b * b - 4.0 * a * c;
56          if (d_sq > eps) {
57              std::complex<double> bad(NAN, NAN);
58              return make_pair(bad, bad);
59          }
60          std::complex<double> d(0.0, sqrt(-d_sq));
61          std::complex<double> x1 = (-b + d) / (2.0 * a);
62          std::complex<double> x2 = (-b - d) / (2.0 * a);
63          return make_pair(x1, x2);
64      }
65
66      void calc_eigen() {
67          for (size_t i = 0; i < n; ++i) {
68              if (i < n - 1 and !(abs(a[i + 1][i]) < eps)) {
69                  auto [l1, l2] = solve_sq(a[i][i], a[i][i + 1], a[i + 1][i], a[i + 1][i + 1]);
70                  if (isnan(l1.real())) {
71                      eigen[i] = COMPLEX_INF;
72                      ++i;
73                      eigen[i] = COMPLEX_INF;
74                      continue;
75                  }
76                  eigen[i] = l1;
77                  eigen[++i] = l2;
78              }
79              else {
80                  eigen[i] = a[i][i];
81              }
82          }
83      }
84
85      bool check_diag() {
86          for (size_t i = 0; i < n; ++i) {
87              double col_sum = 0;
88              for (size_t j = i + 2; j < n; ++j) {
89                  col_sum += a[j][i] * a[j][i];
90              }
91              double norm = sqrt(col_sum);
92              if (!(norm < eps)) {
93                  return false;
94              }
95          }
96          return true;
97      }
98
99      bool check_eps() {
100         if (!check_diag()) {
101             return false;
102         }
103         vector<std::complex<double>> prev_eigen(eigen);
104         calc_eigen();
105         for (size_t i = 0; i < n; ++i) {
106             bool bad = (std::norm(eigen[i] - COMPLEX_INF) < eps);
107             if (bad) {
108                 return false;
109             }
110             double delta = std::norm(eigen[i] - prev_eigen[i]);
```

```
            if (delta > eps) {
                return false;
            }
        }
        return true;
    }

    void build() {
        iter_count = 0;
        while (!check_eps()) {
            ++iter_count;
            matrix_t<double> q = matrix_t<double>::identity(n);
            matrix_t<double> r(a);
            for (size_t i = 0; i < n - 1; ++i) {
                vector<double> b(n);
                for (size_t j = i; j < n; ++j) {
                    b[j] = r[j][i];
                }
                matrix_t<double> h = householder(b, i);
                q = q * h;
                r = h * r;
            }
            a = r * q;
        }
    }
public:
    int iter_count;

    qr(const matrix_t<double> & _a, double _eps) {
        if (_a.rows() != _a.cols()) {
            throw invalid_argument("Matrix is not square");
        }
        n = _a.rows();
        a = matrix_t<double>(_a);
        eps = _eps;
        eigen.resize(n, COMPLEX_INF);
        build();
    };

    vector<std::complex<double>> get_values() {
        calc_eigen();
        return eigen;
    }

    ~qr() = default;
};

string format(complex<double> c) {
    if (abs(c.imag()) < EPS) {
        return to_string(c.real());
    }
    else {
        return to_string(c.real()) + " + i * (" + to_string(c.imag()) + ")";
    }
}

int main() {
    int n;
    double eps;
    cin >> n >> eps;
    matrix_t<double> a(n);
    cin >> a;
```

```
173     qr my_qr(a, eps);
174     vector<complex<double>> lambda = my_qr.get_values();
175     cout << " :" << endl;
176     for (int i = 0; i < n; ++i) {
177         cout << "l_" << i + 1 << " = " << format(lambda[i]) << endl;
178     }
179     cout << "\n   " << my_qr.iter_count << " " << endl;
180 }
```

Входные данные

```
3 0.000001
0 -1 3
-1 6 -3
-8 4 2
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-5$ g++ main.cpp -std=c++17
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab1/lab1-5$ ./a.out <test.txt
Собственные значения:
l_1 = 2.311458 + i * (5.103449)
l_2 = 2.311458 + i * (-5.103449)
l_3 = 3.377085

Решение получено за 32 итерации
```