Московский авиационный институт
(национальный исследовательский университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 2 по курсу «Численные методы»

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

Москва, 2022

Лабораторная работа 2

Методы решения нелинейных уравнений и систем нелинейных уравнений

2.1.

Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

Исходный код

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int iter_count = 0;

double f(double x) {
    return x * x * x * x * x - 2.0 * x - 1.0;
}

double phi(double x) {
    return sqrt(sqrt(2 * x + 1));
}

double phi_s(double x) {
    return 1 / (2 * sqrt(sqrt((2 * x + 1) * (2 * x + 1) * (2 * x + 1))));
}

double f_s(double x) {
    return 4.0 * x * x * x * x - 2.0;
}

double f_ss(double x) {
    return 12.0 * x * x;
}

double iter_solve(double l, double r, double eps) {
    iter_count = 0;
    double x_k = r;
    double dx = 1.0;
    double q = max(abs(phi_s(l)), abs(phi_s(r)));
    double eps_coef = q / (1.0 - q);
    do {
        double x_k1 = phi(x_k);
        dx = eps_coef * abs(x_k1 - x_k);
        ++iter_count;
        x_k = x_k1;
    } while (dx > eps);
    return x_k;
```

```
}

double newton_solve(double l, double r, double eps) {
    double x0 = l;
    if (!(f(x0) * f_ss(x0) > eps)) {
        x0 = r;
    }
    iter_count = 0;
    double x_k = x0;
    double dx = 1.0;
    do {
        double x_k1 = x_k - f(x_k) / f_s(x_k);
        dx = abs(x_k1 - x_k);
        ++iter_count;
        x_k = x_k1;
    } while (dx > eps);
    return x_k;
}

int main() {
    cout.precision(9);
    double l, r, eps;
    cin >> l >> r >> eps;
    double root;
    root = iter_solve(l, r, eps);
    cout << "  " << endl;
    cout << "x0 = " << root << endl;
    cout << "   " << iter_count << " " << "\n\n";
    root = newton_solve(l, r, eps);
    cout << " " << endl;
    cout << "x0 = " << root << endl;
    cout << "   " << iter_count << " " << "\n\n";
}
```

Входные данные

```
test1:
1 2 0.000001

test2:
1 2 0.000000001
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-1$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-1$ ./a.out <test1
Метод простой итерации
x0 = 1.39533769
Решение получено за 8 итераций
```

```
Метод Ньютона
x0 = 1.39533699
Решение получена за 6 итераций

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-1$ ./a.out <test2
Метод простой итерации
x0 = 1.395337
Решение получено за 12 итераций

Метод Ньютона
x0 = 1.39533699
Решение получена за 6 итераций

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-1$
```

2.2.

Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Исходный код

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <utility>
#include <vector>

using namespace std;

template<class T>
vector<T> operator + (const vector<T> & a, const vector<T> & b) {
    size_t n = a.size();
    vector<T> c(n);
    for (size_t i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
    }
    return c;
}

template<class T>
vector<T> operator - (const vector<T> & a, const vector<T> & b) {
    size_t n = a.size();
    vector<T> c(n);
    for (size_t i = 0; i < n; ++i) {
        c[i] = a[i] - b[i];
    }
    return c;
}

template<class T>
class matrix_t {
private:
    size_t n, m;
    vector<vector<T>> data;
public:
    matrix_t() : n(1), m(1), data(1) {}

    matrix_t(size_t _n) : n(_n), m(_n) {
        data.resize(n, vector<T>(n));
    }

    matrix_t(size_t _n, size_t _m) : n(_n), m(_m) {
        data.resize(n, vector<T>(m));
    }

    matrix_t(const matrix_t<T> & other) {
        n = other.n;
        m = other.m;
```

```
48          data = other.data;
49      }
50
51      matrix_t<T> & operator = (const matrix_t<T> & other) {
52          if (this == &other) {
53              return *this;
54          }
55          n = other.n;
56          m = other.m;
57          data = other.data;
58          return *this;
59      }
60
61      static matrix_t<T> identity(size_t n) {
62          matrix_t<T> res(n, n);
63          for (size_t i = 0; i < n; ++i) {
64              res[i][i] = T(1);
65          }
66          return res;
67      }
68
69      matrix_t t() const {
70          matrix_t<T> res(m, n);
71          for (size_t i = 0; i < n; ++i) {
72              for (size_t j = 0; j < m; ++j) {
73                  res[j][i] = data[i][j];
74              }
75          }
76          return res;
77      }
78
79      size_t rows() const {
80          return n;
81      }
82
83      size_t cols() const {
84          return m;
85      }
86
87      void swap_rows(size_t i, size_t j) {
88          if (i == j) {
89              return;
90          }
91          for (size_t k = 0; k < m; ++k) {
92              swap(data[i][k], data[j][k]);
93          }
94      }
95
96      void swap_cols(size_t i, size_t j) {
97          if (i == j) {
98              return;
99          }
100         for (size_t k = 0; k < n; ++k) {
101             swap(data[k][i], data[k][j]);
102         }
103     }
104
105     friend matrix_t<T> operator + (const matrix_t<T> & a, const matrix_t<T> & b) {
106         if (a.rows() != b.rows() or a.cols() != b.cols()) {
107             throw invalid_argument("Sizes does not match");
108         }
109         size_t n = a.rows();
```

```
110         size_t m = a.cols();
111         matrix_t<T> res(n, m);
112         for (size_t i = 0; i < n; ++i) {
113             for (size_t j = 0; j < m; ++j) {
114                 res[i][j] = a[i][j] + b[i][j];
115             }
116         }
117         return res;
118     }
119 
120     friend matrix_t<T> operator - (const matrix_t<T> & a, const matrix_t<T> & b) {
121         if (a.rows() != b.rows() or a.cols() != b.cols()) {
122             throw invalid_argument("Sizes does not match");
123         }
124         size_t n = a.rows();
125         size_t m = a.cols();
126         matrix_t<T> res(n, m);
127         for (size_t i = 0; i < n; ++i) {
128             for (size_t j = 0; j < m; ++j) {
129                 res[i][j] = a[i][j] - b[i][j];
130             }
131         }
132         return res;
133     }
134 
135     friend matrix_t<T> operator * (const matrix_t<T> & a, const matrix_t<T> & b) {
136         if (a.cols() != b.rows()) {
137             throw invalid_argument("Sizes does not match");
138         }
139         size_t n = a.rows();
140         size_t k = a.cols();
141         size_t m = b.cols();
142         matrix_t<T> res(n, m);
143         for (size_t i = 0; i < n; ++i) {
144             for (size_t j = 0; j < m; ++j) {
145                 for (size_t ii = 0; ii < k; ++ii) {
146                     res[i][j] += a[i][ii] * b[ii][j];
147                 }
148             }
149         }
150         return res;
151     }
152 
153     friend vector<T> operator * (const matrix_t<T> & a, const vector<T> & b) {
154         if (a.cols() != b.size()) {
155             throw invalid_argument("Sizes does not match");
156         }
157         size_t n = a.rows();
158         size_t m = a.cols();
159         vector<T> c(n);
160         for (size_t i = 0; i < n; ++i) {
161             for (size_t j = 0; j < m; ++j) {
162                 c[i] += a[i][j] * b[j];
163             }
164         }
165         return c;
166     }
167 
168     friend matrix_t<T> operator * (T lambda, const matrix_t<T> & a) {
169         size_t n = a.rows();
170         size_t m = a.cols();
171         matrix_t<T> res(n, m);
```

```
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < m; ++j) {
                res[i][j] = lambda * a[i][j];
            }
        }
        return res;
    }

    vector<T> & operator [] (size_t i) {
        return data[i];
    }

    const vector<T> & operator [] (size_t i) const {
        return data[i];
    }

    friend istream & operator >> (istream & in, matrix_t<T> & matr) {
        for (size_t i = 0; i < matr.rows(); ++i) {
            for (size_t j = 0; j < matr.cols(); ++j) {
                in >> matr[i][j];
            }
        }
        return in;
    }

    friend ostream & operator << (ostream & out, const matrix_t<T> & matr) {
        for (size_t i = 0; i < matr.rows(); ++i) {
            for (size_t j = 0; j < matr.cols(); ++j) {
                if (j) {
                    out << ", ";
                }
                out << matr[i][j];
            }
            out << '\n';
        }
        return out;
    }

    ~matrix_t() = default;
};

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() {
```

```
296         return det;
297     }
298 
299     matrix_t<T> inv_matrix() {
300         size_t n = l.rows();
301         matrix_t<T> res(n);
302         for (size_t i = 0; i < n; ++i) {
303             vector<T> b(n);
304             b[i] = T(1);
305             vector<T> x = solve(b);
306             for (size_t j = 0; j < n; ++j) {
307                 res[j][i] = x[j];
308             }
309         }
310         return res;
311     }
312 
313     friend ostream & operator << (ostream & out, const lu_t<T> & lu) {
314         out << "Matrix L:\n" << lu.l << "\nMatrix U:\n" << lu.u << '\n';
315         return out;
316     }
317 
318     ~lu_t() = default;
319 };
320 
321 int iter_count = 0;
322 
323 const double a = 1;
324 
325 double f1(double x1, double x2) {
326     return x1 * x1 - 2 * log10(x2) - 1;
327 }
328 
329 double f2(double x1, double x2) {
330     return x1 * x1 - a * x1 * x2 + a;
331 }
332 
333 double phi1(double x1, double x2) {
334     return sqrt(2 * log10(x2) + 1);
335 }
336 
337 double phi1_der(double x1, double x2) {
338     return 1 / (x2 * sqrt(log(10)) * sqrt(2 * log(x2) + log(10)));
339 }
340 
341 double phi2(double x1, double x2) {
342     return (x1 * x1 + a)/(x1 * a);
343 }
344 
345 double phi2_der(double x1, double x2) {
346     return (a - 1 / (x1 * x1 * a));
347 }
348 
349 double phi(double x1, double x2) {
350     return phi1_der(x1, x2) * phi2_der(x1, x2);
351 }
352 
353 pair<double, double> iter_solve(double l1, double r1, double l2, double r2, double eps) {
354     iter_count = 0;
355     double x1_k = r1;
356     double x2_k = r2;
357     double q = -1;
```

```
    q = max(q, abs(phi(l1, r1)));
    q = max(q, abs(phi(l1, r2)));
    q = max(q, abs(phi(l2, r1)));
    q = max(q, abs(phi(l2, r2)));
    double eps_coeff = q / (1 - q);
    double dx = 1;
    while (dx > eps) {
        double x1_k1 = phi1(x1_k, x2_k);
        double x2_k1 = phi2(x1_k, x2_k);
        dx = eps_coeff * (abs(x1_k1 - x1_k) + abs(x2_k1 - x2_k));
        ++iter_count;
        x1_k = x1_k1;
        x2_k = x2_k1;
    }
    return make_pair(x1_k, x2_k);
}

matrix_t<double> Jacobi(double x1, double x2) {
    matrix_t<double> result(2);
    result[0][0] = 2 * x1;
    result[0][1] = -2 / (x2 * log(10));
    result[1][0] = 2 * x1 - a * x2;
    result[1][1] = -a * x1;
    return result;
}

double norm(const vector<double> &vec) {
    double res = 0;
    for (auto elem: vec) {
        res = max(res, abs(elem));
    }
    return res;
}

pair<double, double> newton_solve(double x1_0, double x2_0, double eps) {
    iter_count = 0;
    vector<double> x_k = {x1_0, x2_0};
    double dx = 1;
    while (dx > eps) {
        double x1 = x_k[0];
        double x2 = x_k[1];
        lu_t<double> J(Jacobi(x1, x2));
        vector<double> f_k = {f1(x1,x2), f2(x1,x2)};
        vector<double> d_x = J.solve(f_k);
        vector<double> x_k1 = x_k - d_x;
        dx = norm(x_k1 - x_k);
        ++iter_count;
        x_k = x_k1;
    }
    return make_pair(x_k[0], x_k[1]);
}

int main() {
    cout.precision(8);
    cout << fixed;
    double l1, r1, l2, r2, eps;
    cin >> l1 >> r1 >> l2 >> r2 >> eps;
    auto [x0, y0] = iter_solve(l1, r1, l2, r2, eps);
    cout << "x1 = " << x0 << ", x2 = " << y0 << endl;
    cout << "      " << iter_count << " " << "\n\n";
    auto [x0_n, y0_n] = newton_solve(r1, r2, eps);
    cout << "x1 = " << x0_n << ", x2 = " << y0_n << endl;
```

```
420     cout << "     " << iter_count << " " << "\n\n";
421 }
```

Входные данные

```
test1:
1 2
2 3
0.00001

test2:
1 2
2 3
0.000000001
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-2$ ./a.out <test1
x1 = 1.27576434,x2 = 2.05961418
Решение методом простой итерации получено за 9 итераций

x1 = 1.27576206,x2 = 2.05960729
Решение методом Ньютона получено за 5 итераций

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-2$ ./a.out <test2
x1 = 1.27576206,x2 = 2.05960729
Решение методом простой итерации получено за 15 итераций

x1 = 1.27576206,x2 = 2.05960729
Решение методом Ньютона получено за 6 итераций

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab2/lab2-2$
```