Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра математической кибернетики

Лабораторные работы по курсу «Численные методы»

Лабораторная работа №2

Студент: Ершов С.Г.
Преподаватель: Пивоваров Д.Е.
Дата:
Оценка:
Подпись:

Москва, 2024

2 Численные методы решения нелинейных уравнений

1 Решение нелинейных уравнений

1.1 Постановка задачи

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.

1.2 Консоль

```
$ make
g++ -g -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
$ cat tests/2.in
0.5  1  0.000001
$  ./solution  <tests/2.in
x_0 = 0.774356940
Решение методом простой итерации получено за 9 итераций
x_0 = 0.774356593
Решение методом Ньютона получена за 5 итераций
$ cat tests/3.in
0.5  1  0.000000001
$  ./solution  <tests/3.in
x_0 = 0.774356594
Решение методом простой итерации получено за 14 итераций
x_0 = 0.774356593
Решение методом Ньютона получена за 5 итераций
```

1.3 Исходный код

```cpp
#ifndef SOLVER_HPP
#define SOLVER_HPP

#include <cmath>

int iter_count = 0;

double f(double x) { return std::sin(x) - 2.0 * x * x + 0.5; }

double f_s(double x) { return std::cos(x) - 4.0 * x; }

double f_ss(double x) { return -std::sin(x) - 4.0; }

double phi(double x) { return std::sqrt(0.5 * std::sin(x) + 0.25); }

double phi_s(double x) { return std::cos(x) / (4.0 * phi(x)); }

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

```
48 }
49
50 #endif /* SOLVER_HPP */
```

2 Решение нелинейных систем уравнений

2.1 Постановка задачи

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.

2.2 Консоль

```
$ make
g++ -g -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
$ cat tests/2.in
0 1
1 2
0.000001
$ ./solution <tests/2.in
x_0 = 0.832187878,y0 = 1.739406197
Решение методом простой итерации получено за 77 итераций
x_0 = 0.832187922,y0 = 1.739406179
Решение методом Ньютона получена за 4 итераций
$ cat tests/3.in
0 1
1 2
0.000000001
$ ./solution <tests/3.in
x_0 = 0.832187922,y0 = 1.739406179
Решение методом простой итерации получено за 111 итераций
x_0 = 0.832187922,y0 = 1.739406179
Решение методом Ньютона получена за 4 итераций
```

2.3 Исходный код

```
#ifndef SYSTEM_SOLVER_HPP
#define SYSTEM_SOLVER_HPP

#include "../lab1_1/lu.hpp"

int iter_count = 0;

const double a = 1;

double phi1(double x1, double x2) {
    (void)x1;
    return a + std::cos(x2);
}

double phi1_s(double x1, double x2) {
    (void)x1;
    return -std::sin(x2);
}

double phi2(double x1, double x2) {
    (void)x2;
    return a + std::sin(x1);
}

double phi2_s(double x1, double x2) {
    (void)x2;
    return std::cos(x1);
}

double phi(double x1, double x2) { return phi1_s(x1, x2) * phi2_s(x1, x2); }

using pdd = std::pair<double, double>;

pdd iter_solve(double l1, double r1, double l2, double r2, double eps) {
    iter_count = 0;
    double x_1_k = r1;
    double x_2_k = r2;
    double q = -1;
    q = std::max(q, std::abs(phi(l1, r1)));
    q = std::max(q, std::abs(phi(l1, r2)));
    q = std::max(q, std::abs(phi(l2, r1)));
    q = std::max(q, std::abs(phi(l2, r2)));
    double eps_coef = q / (1 - q);
    double dx = 1;
    do {
        double x_1_k1 = phi1(x_1_k, x_2_k);
        double x_2_k1 = phi2(x_1_k, x_2_k);
```

```
48          dx = eps_coef * (std::abs(x_1_k1 - x_1_k) + std::abs(x_2_k1   - x_2_k));
49          ++iter_count;
50          x_1_k = x_1_k1;
51          x_2_k = x_2_k1;
52      } while (dx > eps);
53      return std::make_pair(x_1_k, x_2_k);
54  }
55
56  using matrix = matrix_t<double>;
57  using lu = lu_t<double>;
58  using vec = std::vector<double>;
59
60  double f1(double x1, double x2) { return x1 - std::cos(x2)   - a; }
61
62  double f2(double x1, double x2) { return x2 - std::sin(x1)   - a; }
63
64  matrix j(double x1, double x2) {
65      matrix res(2);
66      res[0][0] = 1.0;
67      res[0][1] = std::sin(x2);
68      res[1][0] = -std::cos(x1);
69      res[1][1] = 1.0;
70      return res;
71  }
72
73  double norm(const vec& v) {
74      double res = 0;
75      for (double elem : v) {
76          res = std::max(res, std::abs(elem));
77      }
78      return res;
79  }
80
81  pdd newton_solve(double x1_0, double x2_0, double eps) {
82      iter_count = 0;
83      vec x_k = {x1_0, x2_0};
84      double dx = 1;
85      do {
86          double x1 = x_k[0];
87          double x2 = x_k[1];
88          lu jacobi(j(x1, x2));
89          vec f_k = {f1(x1, x2), f2(x1, x2)};
90          vec delta_x = jacobi.solve(f_k);
91          vec x_k1 = x_k - delta_x;
92          dx = norm(x_k1 - x_k);
93          ++iter_count;
94          x_k = x_k1;
95      } while (dx > eps);
96      return std::make_pair(x_k[0], x_k[1]);
```

```
97  }
98
99  #endif /* SYSTEM_SOLVER_HPP */
```