

**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: Белов И.А.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

2.1 Методы простой итерации и Ньютона

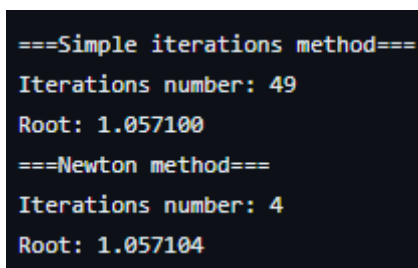
1 Постановка задачи

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.

Вариант: 2

$$\ln(x + 2) - x^2 = 0$$

2 Результаты работы (Нашло два корня, оба из них верные)



```
===Simple iterations method===
Iterations number: 49
Root: 1.057100
===Newton method===
Iterations number: 4
Root: 1.057104
```

Рис. 1: Вывод программы

3 Исходный код

```
1 | #include <cmath>
2 | #include <algorithm>
3 | #include <iostream>
4 | #include <fstream>
5 | #include <string>
6 | using namespace std;
7 |
8 | // ,      : ln(x+2) - x^2
9 | double F(double x) {
10 |     return log(x + 2) - pow(x, 2);
11 | }
12 |
13 | //      F
14 | double Diff_F(double x) {
15 |     return 1 / (x + 2) - 2 * x;
```

```

16 }
17
18 // Phi
19 double Phi(double x) {
20     return x + 0.1 * (log(x + 2) - pow(x, 2));
21 }
22
23 //
24 double Iterations_method(double x0, double eps, int& i) {
25     double x1 = x0;
26     do {
27         x0 = x1;
28         x1 = Phi(x1);
29         i += 1;
30     } while (abs(x1 - x0) > eps);
31     return x1;
32 }
33
34 //
35 double Newton_method(double x0, double eps, int& i) {
36     double x1 = x0;
37     do {
38         x0 = x1;
39         x1 = x0 - F(x1) / Diff_F(x1);
40         i += 1;
41     } while (abs(x1 - x0) >= eps);
42     return x1;
43 }
44
45 int main() {
46     cout.precision(9);
47     ofstream fout("answer.txt");
48     double eps=0.000001;
49     double X_iterations, x0_iterations = 1.0; //
50     int iterator_iterations = 0;
51     X_iterations = Iterations_method(x0_iterations, eps, iterator_iterations);
52     fout << "===Simple iterations method===\nIterations number: " <<
        iterator_iterations << "\nRoot: " << to_string(X_iterations) << '\n';
53     double X_Newton, x0_Newton = 1.0; //
54     int iterator_Newton = 0;
55     X_Newton = Newton_method(x0_Newton, eps, iterator_Newton);
56     fout << "===Newton method===\nIterations number: " << iterator_Newton << "\nRoot: "
        << to_string(X_Newton) << '\n';
57 }

```

2.2 Методы простой итерации и Ньютона

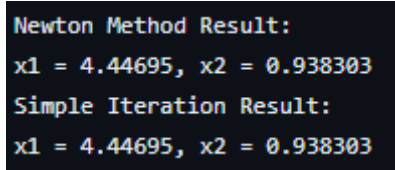
4 Постановка задачи

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.

Вариант: 2

$$\begin{cases} (x_1^2 + a^2) * x_2 - a^3 = 0 \\ (x_1 - \frac{a}{2})^2 + (x_2 - \frac{a}{2})^2 - a^2 = 0 \end{cases}$$

5 Результаты работы



```
Newton Method Result:
x1 = 4.44695, x2 = 0.938303
Simple Iteration Result:
x1 = 4.44695, x2 = 0.938303
```

Рис. 2: Вывод программы

6 Исходный код

```
1 | #include <iostream>
2 | #include <cmath>
3 | #include <fstream>
4 | #include <vector>
5 | using namespace std;
6 |
7 | //
8 | double f1(double x1, double x2) {
9 |     return (x1 * x1 + 9) * x2 - 27;
10 | }
11 |
12 | double f2(double x1, double x2) {
13 |     return (x1 - 1.5) * (x1 - 1.5) + (x2 - 1.5) * (x2 - 1.5) - 9;
14 | }
15 |
16 | //
17 | vector<vector<double>> jacobian(double x1, double x2) {
18 |     vector<vector<double>> J(2, vector<double>(2));
19 |     J[0][0] = 2 * x1 * x2;
20 |     J[0][1] = x1 * x1 + 9;
21 |     J[1][0] = 2 * (x1 - 1.5);
22 |     J[1][1] = 2 * (x2 - 1.5);
23 |     return J;
24 | }
25 |
26 | //
27 | vector<double> newtonMethod(double x1, double x2, double tol) {
28 |     vector<double> x = { x1, x2 };
29 |     int iteration = 0;
30 |     while (true) {
31 |         vector<vector<double>> J = jacobian(x[0], x[1]);
32 |         double det = J[0][0] * J[1][1] - J[0][1] * J[1][0];
33 |         if (fabs(det) < 1e-6) break;
34 |
35 |         vector<vector<double>> invJ(2, vector<double>(2));
36 |         invJ[0][0] = J[1][1] / det;
37 |         invJ[0][1] = -J[0][1] / det;
38 |         invJ[1][0] = -J[1][0] / det;
39 |         invJ[1][1] = J[0][0] / det;
40 |
41 |         vector<double> f = { f1(x[0], x[1]), f2(x[0], x[1]) };
42 |         vector<double> dx = { invJ[0][0] * f[0] + invJ[0][1] * f[1], invJ[1][0] * f[0]
43 |             + invJ[1][1] * f[1] };
44 |         x[0] -= dx[0];
45 |         x[1] -= dx[1];
46 |         iteration++;
```

```

47     if (sqrt(dx[0] * dx[0] + dx[1] * dx[1]) < tol) break;
48 }
49 return x;
50 }
51
52 //
53 vector<double> simpleIteration(double x1, double tol) {
54     double x2 = 27 / (x1 * x1 + 9);
55     int iteration = 0;
56     double x1_new;
57     do {
58         x1_new = x1;
59         x1 = sqrt(9 - (x2 - 1.5) * (x2 - 1.5)) + 1.5;
60         x2 = 27 / (x1 * x1 + 9);
61         iteration++;
62     } while (fabs(x1 - x1_new) > tol && iteration < 10000);
63     return { x1, x2 };
64 }
65
66 int main() {
67     ofstream fout("answer.txt");
68     double x1_initial = 2.0;
69     double x2_initial = 2.0;
70     double tol = 1e-6;
71
72     vector<double> result_newton = newtonMethod(x1_initial, x2_initial, tol);
73     vector<double> result_si = simpleIteration(x1_initial, tol);
74
75     fout << "Newton Method Result:\n";
76     fout << "x1 = " << result_newton[0] << ", x2 = " << result_newton[1] << endl;
77     fout << "Simple Iteration Result:\n";
78     fout << "x1 = " << result_si[0] << ", x2 = " << result_si[1] << endl;
79     fout.close();
80
81     return 0;
82 }

```