

**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. Р. Жалялетдинов
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

2.1 Методы простой итерации и Ньютона

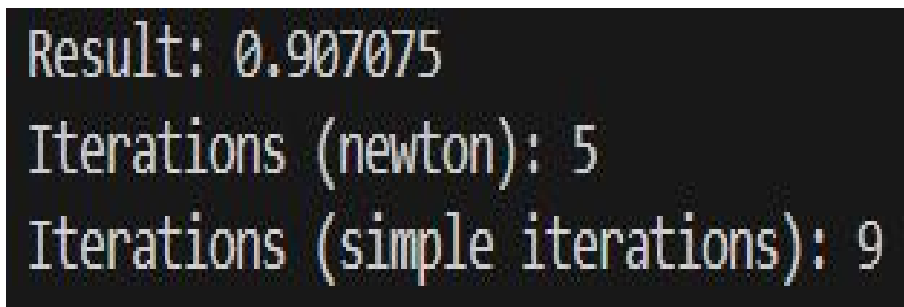
1 Постановка задачи

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: 8

$$\ln(x + 1) - 2x^2 + 1 = 0$$

2 Результаты работы



```
Result: 0.907075
Iterations (newton): 5
Iterations (simple iterations): 9
```

Рис. 1: Вывод программы в консоли

3 Исходный код

```
1 | #include <bits/stdc++.h>
2 |
3 | using namespace std;
4 |
5 |
6 | //
7 | double f(double x){
8 |     return log(x+1) - 2*pow(x, 2) + 1;
9 | }
10 |
11 | //
12 | double df(double x){
13 |     return 1/(x+1) - 4*x;
14 | }
15 |
16 | //
17 | double phi(double x) {
18 |     return sqrt((log(x+1)+1)/2);
19 | };
20 |
21 | //      ( )
22 | pair<double, int> newton(double x0, double EPS){
23 |     /*
24 |         x0 -      ( )
25 |         EPS -
26 |
27 |         k -
28 |         prev -      ( )
29 |         last -      ( )
30 |     */
31 |
32 |     int k = 1;
33 |     double last = x0 - f(x0)/df(x0), prev = x0; //
34 |
35 |     while (EPS <= abs(last - prev)){ //
36 |         k += 1;
37 |         prev = last;
38 |         last = last - f(last)/df(last); //
39 |     }
40 |     return make_pair(last, k);
41 | }
42 |
43 |
44 | //      ( )
45 | pair<double, int> iteration_method(double x0, double q, double EPS){
46 |     /*
47 |         x0 -      ( )
```

```

48     q -
49     EPS -
50
51     k -
52     prev -    ()
53     last -    ()
54 */
55 int k = 1;
56 double last = x0*2 + 1, prev = x0; //
57
58 while (EPS <= q*abs(last - prev)/(1-q)){ //
59     k += 1;
60     prev = last;
61     last = phi(last); //
62 }
63 return make_pair(last, k);
64 }
65
66
67 int main(){
68     double x_res;
69     int iters_count;
70
71     tie(x_res, iters_count) = newton(2, 1e-5);
72     cout << "Result: " << x_res << endl << "Iterations (newton): " << iters_count <<
        endl;
73
74     tie(x_res, iters_count) = iteration_method(0.5, 0.8, 1e-5);
75     cout << "Iterations (simple iterations): " << iters_count << endl << endl;
76 }

```

2.2 Методы простой итерации и Ньютона

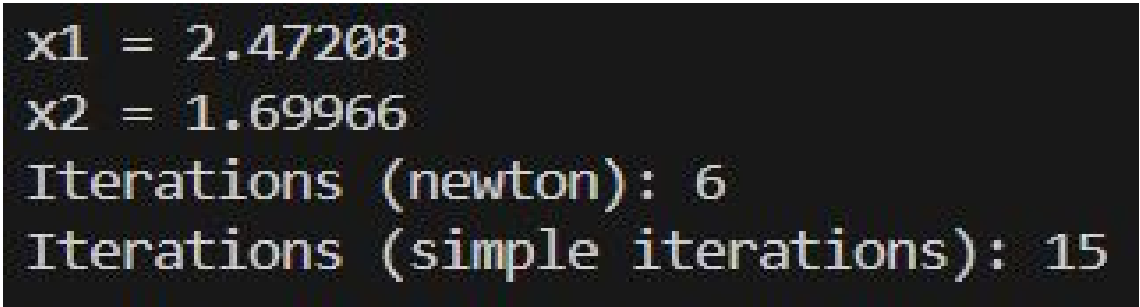
4 Постановка задачи

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: 8

$$\begin{cases} x_1^2 + x_2^2 - 9 = 0 \\ x_1 - e^{x_2} + 3 = 0 \end{cases}$$

5 Результаты работы



```
x1 = 2.47208
x2 = 1.69966
Iterations (newton): 6
Iterations (simple iterations): 15
```

Рис. 2: Вывод программы в консоли

6 Исходный код

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using float_vect = vector<double>;
5
6  //  $x_1^2 + x_2^2 - 9 = 0$ 
7  auto f1 = [](float_vect x) {
8      return pow(x[0], 2) + pow(x[1], 2) - 9;
9  };
10
11 //  $x_1 - e^{(x_2)} + 3 = 0$ 
12 auto f2 = [](float_vect x) {
13     return x[0] - exp(x[1]) + 3;
14 };
15
16 //
17 pair<float_vect, int> nm(float_vect x_0, double deviation) {
18
19     // ( )
20     auto eps = [](float_vect& val1, float_vect& val2) {
21         double d = 0.0;
22         for (int i = 0; i < val1.size(); ++i) {
23             d = max(d, abs(val1[i] - val2[i]));
24         }
25         return d;
26     };
27
28     //
29     auto det = [](float_vect& x, const vector<vector<function<double(float_vect)>>>& m)
30     {
31         return m[0][0](x) * m[1][1](x) - m[0][1](x) * m[1][0](x);
32     };
33
34     //
35     auto f1x1 = [](const float_vect& x) { return 2 * x[0]; };
36     auto f1x2 = [](const float_vect& x) { return 2 * x[1]; };
37     auto f2x1 = [](const float_vect& x) { return 1; };
38     auto f2x2 = [](const float_vect& x) { return -exp(x[1]); };
39
40     // (J )
41     vector<vector<function<double(float_vect)>>> A1 = {{f1, f1x2}, {f2, f2x2}};
42
43     vector<vector<function<double(float_vect)>>> A2 = {{f1x1, f1}, {f2x1, f2}};
44
45     vector<vector<function<double(float_vect)>>> J = {{f1x1, f1x2}, {f2x1, f2x2}};
```

```

46     float_vect x_next = { x_0[0] - det(x_0, A1)/det(x_0, J), x_0[1] - det(x_0, A2)/det(
      x_0, J) }, x_curr = x_0; //      ( )
47     int k = 1;
48
49     while (deviation <= eps(x_curr, x_next)){
50         k += 1;
51         x_curr = x_next;
52         x_next = { x_next[0] - det(x_next, A1)/det(x_next, J), x_next[1] - det(x_next,
      A2)/det(x_next, J) };
53     }
54     return make_pair(x_next, k);
55 }
56
57 //
58 pair<float_vect, int> sim(float_vect x_0, double q, double deviation) {
59
60     //      ( )
61     auto eps = [](float_vect& val1, float_vect& val2, double q) {
62         double d = 0.0;
63         for (int i = 0; i < val1.size(); ++i) {
64             d = max(d, abs(val1[i] - val2[i]));
65         }
66         return d*q/(1-q);
67     };
68
69     float_vect x_next = x_0, x_curr = {x_0[0] + 5, x_0[1]*3 + 5}; //      ( )
70     int k = 1;
71
72     while (deviation <= eps(x_curr, x_next, q)){
73         k += 1;
74         x_curr = x_next;
75         x_next = { sqrt(9-pow(x_next[1], 2)), log(x_next[0]+3) }; //
76     }
77     return make_pair(x_next, k);
78 }
79
80
81 int main(){
82     float_vect r;
83     int iter_count;
84     //
85     tie(r, iter_count) = nm({1.0, 1.0}, 1e-5);
86     cout << "x1 = " << r[0] << endl << "x2 = " << r[1] << endl << "Iterations (newton):
      " << iter_count << endl;
87     //
88     tie(r, iter_count) = sim({1.0, 1.0}, 0.7, 1e-5);
89     cout << "Iterations (simple iterations): " << iter_count << endl << endl;
90 }

```