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: Первухин А.С.
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: 17

$4^x - 5x - 2 = 0$

2 Результаты работы

```
Начальное приближение x_0 = -0.3
Решение методом Ньютона:-0.260652 | Количество итераций: 3
Решение методом простых итераций:-0.260662 | Количество итераций: 5
------------------------------------------------------------------------
Начальное приближение x_0 = 1.5
Решение методом Ньютона:1.69414 | Количество итераций: 4
Решение методом простых итераций:-0.260647 | Количество итераций: 10
```

Рис. 1: Вывод программы

3 Исходный код

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

using namespace std;

double F(double x){
    return pow(4,x) - 5 * x - 2;
}

double DF(double x){
    return log(4) * pow(4,x) - 5;
}

```

```cpp
double phi(double x){
    return (pow(4,x) - 2)/5;
}

pair <double, int> Newton(double x0, double eps){
    double xn, x;
    x = x0;
    int k = 0;
    for (int i = 0; i < 10000; i++) {

        xn = x - F(x) / DF(x);
        k++;
        if (fabs(xn - x) < eps)
            break;
        x = xn;
    }
    return make_pair(xn, k);
}

pair <double, int> SimpleIterations(double x0, double eps){
    double xn, x;
    x = x0;
    int k = 0;
    for (int i = 0; i < 10000; i++) {
        xn = phi(x);
        k++;
        if (fabs(xn - x) < eps)
            break;
        x = xn;
    }
    return make_pair(xn, k);
}

int main() {
    ofstream fout;
    fout.open("output.txt");
    double x0 = -0.3;
    double x1 = 1.5;
    double eps = 0.0001;
    auto [newton, k] = Newton(x0, eps);
    auto [simpIt, n] = SimpleIterations(x0, eps);
    fout << "  x_0 = "<< x0 << endl;
    fout << "  :" << newton << " |  : " << k << endl;
    fout << "   :" << simpIt << " |  : " << n << endl;
    fout << "-----------------------------------------------------------------------" <<
         endl;
    auto [newton1, k1] = Newton(x1, eps);
    auto [simpIt1, n1] = SimpleIterations(x1, eps);
    fout << "  x_0 = "<< x1 << endl;
```

```
63      fout << "  :" << newton1 << " |  : " << k1 << endl;
64      fout << "   :" << simpIt1 << " |  : " << n1 << endl;
65
66      return 0;
67  }
```

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: 17

$$\begin{cases} 3x_1 - cos(x_2) = 0 \\ 3x_2 - exp(x_1) = 0 \end{cases}$$

5 Результаты работы

```
Решение методом простых итераций:
x1 = 0.300148 | x2 = 0.449989
Количество итераций:7
-----------------------------------
Решение методом Ньютона:
x1 = 0.300147 | x2 = 0.450017
Количество итераций:4
```

Рис. 2: Вывод программы

6 Исходный код

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

using namespace std;

double F1(double x1, double x2){
    return 3*x1 - cos(x2);
}

double F2(double x1, double x2){
    return 3*x2 - exp(x1);
}

double DF1_dx1(double x1, double x2){
    return 3;
}

double DF2_dx1(double x1, double x2){
    return -exp(x2);
}

double DF1_dx2(double x1, double x2){
    return sin(x2);
}

double DF2_dx2(double x1, double x2){
    return 3;
}

double Phi1(double x1, double x2){
    return cos(x2)/3;
}

double Phi2(double x1, double x2){
    return exp(x1)/3;
}

tuple <double, double, int> SimpleIterations(double x1_0, double x2_0, double eps){
    double x1 = x1_0, x2 = x2_0;
    double xn1, xn2;
    int k = 0;
    for (int i = 1; i < 1000; i++){
        xn1 = Phi1(x1, x2);
        xn2 = Phi2(x1,x2);
        k++;
        if (max(fabs(xn1 - x1), fabs(xn2 - x2)) < eps){
```

```
48            break;
49        }
50        x1 = xn1;
51        x2 = xn2;
52    }
53    return make_tuple(xn1, xn2, k);
54 }
55
56 tuple <double, double, int> Newton(double x1_0, double x2_0, double eps){
57    double x1 = x1_0, x2 = x2_0;
58    double xn1, xn2;
59    double detA1, detA2, detJ;
60    int k = 0;
61    for (int i = 0; i < 1000; i++){
62        k++;
63        detA1 = F1(x1, x2) * DF2_dx2(x1, x2) - F2(x1, x2) * DF1_dx2(x1, x2);
64        detA2 = DF1_dx1(x1, x2) * F2(x1, x2) - DF2_dx1(x1, x2) * F1(x1, x2);
65        detJ = DF1_dx1(x1, x2) * DF2_dx2(x1, x2) - DF2_dx1(x1, x2) * DF1_dx2(x1, x2);
66        xn1 = x1 - detA1 / detJ;
67        xn2 = x2 - detA2 / detJ;
68        if (max(fabs(xn1 - x1), fabs(xn2 - x2)) < eps){
69            break;
70        }
71        x1 = xn1;
72        x2 = xn2;
73    }
74    return make_tuple(x1, x2, k);
75 }
76
77 int main() {
78    ofstream fout;
79    fout.open("output.txt");
80    double x1 = 0.5, x2 = 0.5;
81    double eps = 0.0001;
82    auto [res1, res2, iter] = SimpleIterations(x1, x2, eps);
83    fout << "   :" << endl;
84    fout << "x1 = " << res1 << " | x2 = " << res2 << endl;
85    fout << " :" << iter << endl;
86    fout << "-----------------------------------" << endl;
87    auto [res12, res22, iter2] = Newton(x1, x2, eps);
88    fout << "  :" << endl;
89    fout << "x1 = " << res12 << " | x2 = " << res22 << endl;
90    fout << " :" << iter2 << endl;
91    return 0;
92 }
```