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:	В. В. Хрушкова
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: 27

$x^6 - 5x^3 - 2 = 0$

2 Результаты работы

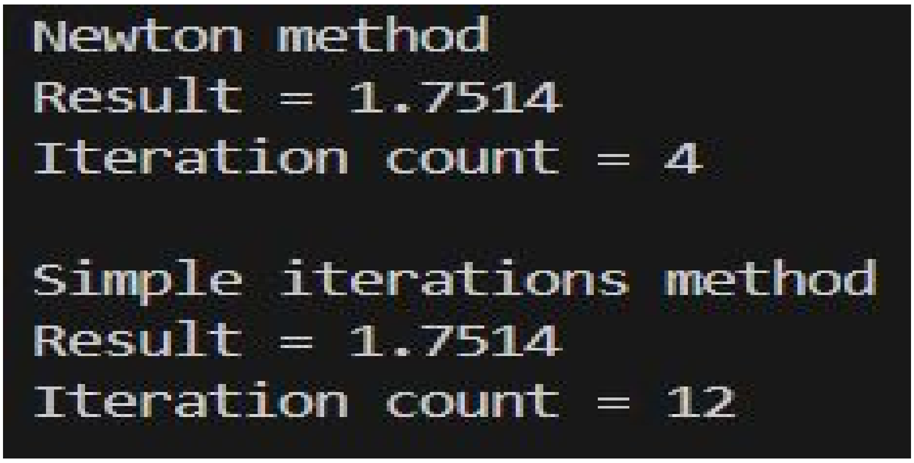

Рис. 1: Вывод программы в консоли

3 Исходный код

```
#include <bits/stdc++.h>

using namespace std;


double f(const double x){
    return pow(x, 6) - 5 * pow(x, 3) - 2;
}

double df (const double x) {
    return 6 * pow(x, 5) - 15 * pow(x, 2);
}

double eps (const double val1, const double val2, double q) {
    if (q == -1)
        return abs(val1 - val2);
    return q*abs(val1 - val2)/(1-q);
}

double fi(double x) {
    return pow(5*pow(x, 3)+2, 1.0/6);
}

pair<double, int> newton(double start_value, double EPS){
    double current_val = start_value - f(start_value)/df(start_value), previous_val =
        start_value;
    int counter=0;

    while (eps(previous_val, current_val, -1) >= EPS){
        counter++;
        previous_val = current_val;
        current_val = current_val - f(current_val)/df(current_val);
    }
    return {current_val, counter};
}

pair<double, int> simple_iter(double start_value, double q, double EPS){
    double current_val = start_value, previous_val = start_value*5;
    int counter=0;

    while (eps(previous_val, current_val, q) >= EPS){
        counter++;
        previous_val = current_val;
        current_val = fi(current_val);
    }
    return {current_val, counter};
}
```

```
int main(){
    double epsillon = 0.00001, res;
    int counter;

    tie(res, counter) = newton(2, epsillon);
    cout << endl << "Newton method" << endl << "Result = " << res << endl << "Iteration
         count = " << counter << endl << endl;

    tie(res, counter) = simple_iter(1.7, 0.6, epsillon);
    cout << "Simple iterations method" << endl << "Result = " << res << endl << "
        Iteration count = " << counter << endl << endl;

    return 1;
}
```

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: 27

$$\begin{cases} 3x_1^2 - x_1 + x_2^2 - 3 = 0 \\ x_1 - \sqrt{x_2 + 3} + 1 = 0 \end{cases}$$

5 Результаты работы

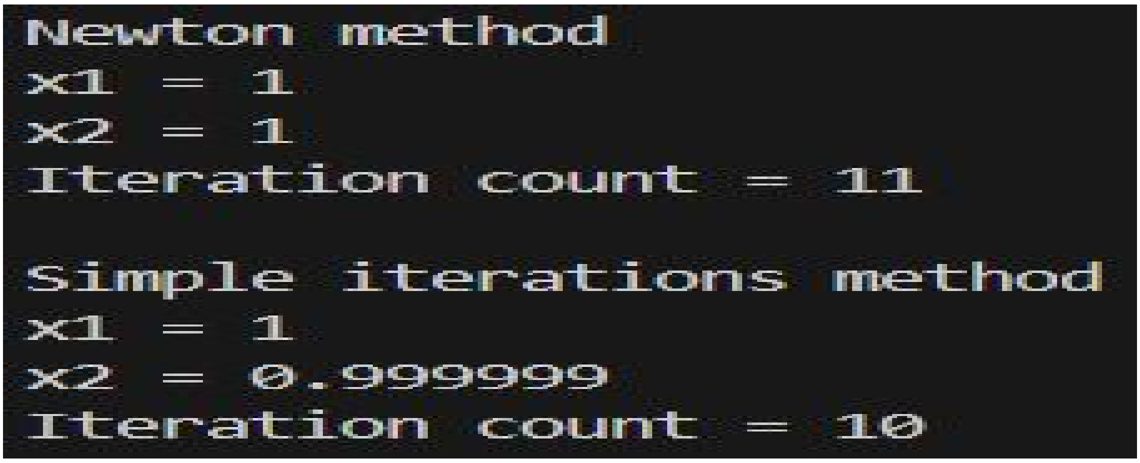

Рис. 2: Вывод программы в консоли

6 Исходный код

```cpp
#include <bits/stdc++.h>

using namespace std;


double f1(double x1, double x2) {
    return x1 - sqrt(x2+3) + 1;
}

double f2(double x1, double x2) {
    return 3*x1*x1 - x2 + x2*x2 - 3;
}

double df1x1(double x1, double x2){
    return 1;
}

double df1x2(double x1, double x2){
    return 1/(2*sqrt(x2+3));
}

double df2x1(double x1, double x2){
    return 6*x1 + x2*x2;
}

double df2x2(double x1, double x2){
    return 3*x1*x1 + 2*x2;
}

double fi1(double x1, double x2) {
    return sqrt(x2+3) - 1;
}

double fi2(double x1, double x2) {
    // return 3*x1*x1 + x2*x2 - 3;
    return sqrt(x2-3*x1*x1+3);
}

double eps(const vector<double>& vect1, const vector<double>& vect2, double q) {
    double d = 0.0;
    for(int i = 0; i < vect1.size(); i++)
        d = max(d, abs(vect1[i] - vect2[i]));
    if (q == -1)
        return d;
    return q*d/(1-q);
}

```

```
double determinant(double x1, double x2, vector<vector<function<double(double, double)>>>& matrix) {
    return matrix[0][0](x1, x2) * matrix[1][1](x1, x2) - matrix[0][1](x1, x2) * matrix[1][0](x1, x2);
}


tuple<double, double, int> newton(double start_value_1, double start_value_2, double EPS) {
    vector<vector<function<double(double, double)>>> J = {{df1x1, df1x2}, {df2x1, df2x2}};
    vector<vector<function<double(double, double)>>> A_1 = {{f1, df1x2}, {f2, df2x2}};
    vector<vector<function<double(double, double)>>> A_2 = {{df1x1, f1}, {df2x1, f2}};

    int counter = 0;
    double x_next_1, x_next_2, x_curr_1 = start_value_1, x_curr_2 = start_value_2;
    x_next_1 = start_value_1 - determinant(start_value_1, start_value_2, A_1)/determinant(start_value_1, start_value_2, J);
    x_next_2 = start_value_2 - determinant(start_value_1, start_value_2, A_2)/determinant(start_value_1, start_value_2, J);

    while (eps({x_curr_1, x_curr_2}, {x_next_1, x_next_2}, -1) >= EPS){
        counter += 1;

        x_curr_1 = x_next_1;
        x_curr_2 = x_next_2;

        x_next_1 = x_next_1 - determinant(x_next_1, x_next_2, A_1)/determinant(x_next_1, x_next_2, J);
        x_next_2 = x_next_2 - determinant(x_next_1, x_next_2, A_2)/determinant(x_next_1, x_next_2, J);
    }

    return {x_next_1, x_next_2, counter};
}


tuple<double, double, int> simple_iter(double start_value_1, double start_value_2, double q, double EPS) {
    int counter = 0;
    double x_next_1 = start_value_1, x_next_2 = start_value_2, x_curr_1 = start_value_1*5, x_curr_2 = start_value_2*5;

    while (eps({x_curr_1, x_curr_2}, {x_next_1, x_next_2}, q) >= EPS and counter < 300){
        counter += 1;

        x_curr_1 = x_next_1;
        x_curr_2 = x_next_2;
```

```cpp

        x_next_1 = fi1(x_next_1, x_next_2);
        x_next_2 = fi2(x_next_1, x_next_2);
    }

    return {x_next_1, x_next_2, counter};
}


int main(){
    double epsillon = 0.00001, res_1, res_2;
    int counter;

    tie(res_1, res_2, counter) = newton(-2.0, 0.0, epsillon);
    cout << endl << "Newton method" << endl << "x1 = " << res_1 << endl << "x2 = " <<
        res_2 << endl << "Iteration count = " << counter << endl << endl;

    tie(res_1, res_2, counter) = simple_iter(-2.0, 0.0, 0.7, epsillon);
    cout << "Simple iterations method" << endl << "x1 = " << res_1 << endl << "x2 = "
        << res_2 << endl << "Iteration count = " << counter << endl << endl;

    return 1;
}
```

```python
import math
import numpy as np
import matplotlib.pyplot as plt


figure, axes = plt.subplots(1)


x = np.linspace(0, 3, 500)
y = list(map(lambda i: (i+1)**2 + 8, x))
axes.plot(x, y)

x = np.linspace(0, 4, 500)
y = list(map(lambda i: 6*math.sqrt(i+3), x))
axes.plot(x, y)


# plt.xticks(np.arange(min(*x1, *x2)-1, max(*x1, *x2)+1, 1.0))
# plt.yticks(np.arange(min(*x1, *x2)-1, max(*x1, *x2)+1, 1.0))
# axes.set_aspect(1)
plt.grid()
plt.show()
```