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:	Ерофеева Е.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: 6

$y^x - 2x - 2 = 0$

2 График функции

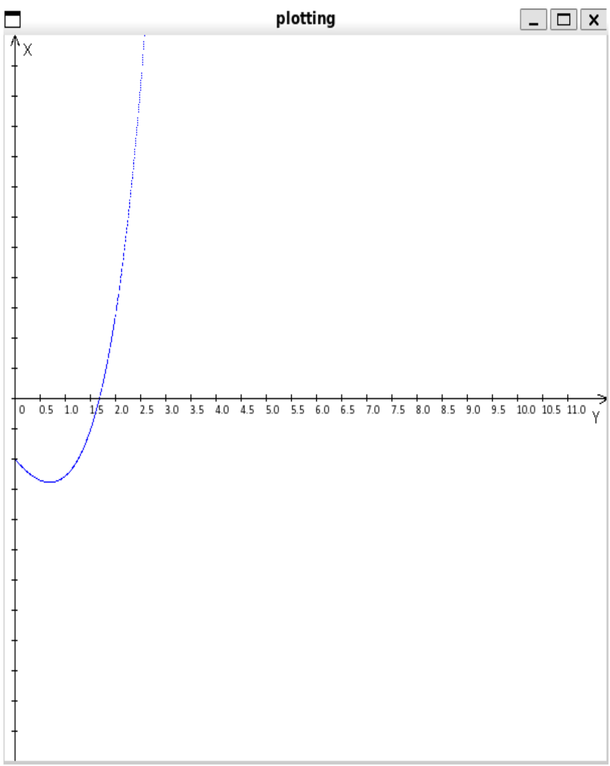

Рис. 1: Вывод программы в консоли

3 Результаты работы

```
Выполнено 3 итераций методом Ньютона
        Результат применения метода Ньютона:
1.68
Проверка: f(x) = 0
        Результат применения метода простых итераций:
1.68
Проверка: f(x) = 0
```

Рис. 2: Вывод программы в консоли

4 Исходный код

```
#include <iostream>
#include <cmath>

namespace nfunc{
    // f(x) = e^x - 2x - 2
    double f(double x){
        return exp(x) - 2*x - 2;
    }
    // f'(x) = e^x - 2
    double df_dx(double x){
        return exp(x) - 2;
    }
    // f''(x) = e^x
    double ddf_dxdx(double x){
        return exp(x);
    }
    double epsk(double xk, double xk_inc){
        return fabs(xk-xk_inc);
    }
}

namespace sifunc{
    // f(x) = e^x - 2x - 2
    double f(double x){
        return exp(x) - 2*x - 2;
    }
    // x = phi(x), phi(x) = ln(2(x+1))
    double phi(double x){
        return log(2*(x+1));
    }
    // phi'(x) = 1/(x+1)
    double dphi_dx(double x){
        return 1.0/(x+1);
    }
    double epsk(double xk, double xk_inc, double q){
        return q*fabs(xk - xk_inc)/(1-q);
    }
}


double newton(double a, double b, double eps){
    //    f(x_0)*f''(x_0) > 0
    //
    double x0 = a - eps;
    if(nfunc::f(a)*nfunc::ddf_dxdx(a) > 0){
        x0 = a;
    } else if(nfunc::f(b)*nfunc::ddf_dxdx(b) > 0){
```

```
        x0 = b;
    } else {
        return x0;
    }
    //
    int k = -1;
    double xk = x0;
    double xk_inc = x0 + 2*eps;
    while (nfunc::epsk(xk, xk_inc) >= eps){
        k += 1;
        xk = xk_inc;
        xk_inc = xk - nfunc::f(xk)/nfunc::df_dx(xk);
    }
    std::cout << " " << k << "    " << std::endl;
    return xk_inc;
}


double simple_iterations_method(double x0, double a, double b, double eps){
    //
   double q = fabs(sifunc::dphi_dx(x0));
    if ( q <= 0 || q>= 1){
        return a - eps;
    }
    //
    int k = -1;
    double xk = (a + b)/2;
    double xk_inc = x0 + 2*eps;
    while (sifunc::epsk(xk, xk_inc, q) >= eps){
        k += 1;
        xk_inc = xk;
        xk = sifunc::phi(xk_inc);
    }
    return xk_inc;
}


int main(){
    double eps = 0.001;

    double result = newton(1.5, 2, eps);
    std::cout << "\t   :" << std::endl;
    std::cout << round(result*100)/100.0 << std::endl;
    std::cout << ": f(x) = " << round(nfunc::f(result)*100)/100.0 << std::endl;

    result = simple_iterations_method(2, 1.5, 2, eps);
    std::cout << "\t    :" << std::endl;
    std::cout << round(result*100)/100.0 << std::endl;
    std::cout << ": f(x) = " << round(sifunc::f(result)*100)/100.0 << std::endl;
```

```

    return 0;
}
```

2.2 Методы простой итерации и Ньютона

5 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: 6

$$\begin{cases} x_1 - cos(x_2) = 0 \\ x_2 - lg(x_1 + 1) = 0 \end{cases}$$

6 Графики функций

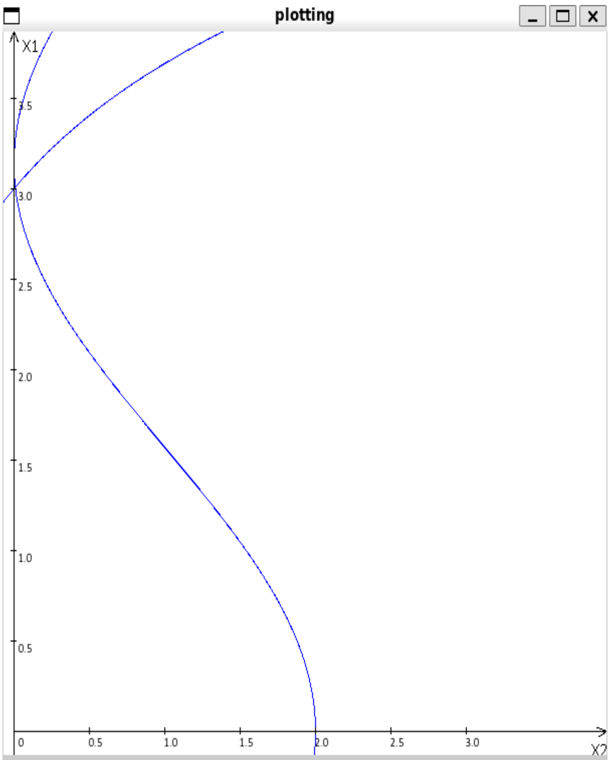

Рис. 3: Вывод программы в консоли

7 Результаты работы

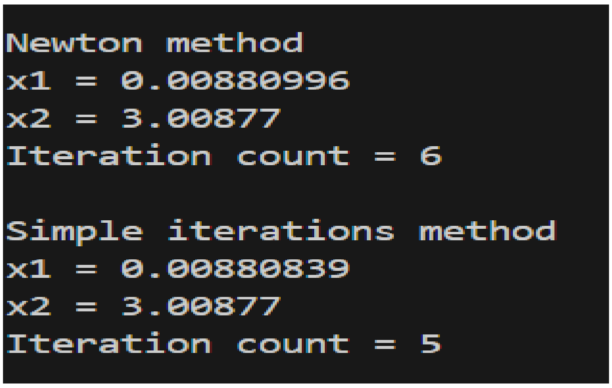

Рис. 4: Вывод программы в консоли

8 Исходный код

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

using namespace std;

namespace nfunc {
    double f1(double x1, double x2) {
        return x1 - cos(x2) - 1;
    }
    double f2(double x1, double x2) {
        return x2 - log(x1 + 1) - 3;
    }
    double df1_dx1(double x1, double x2){
        return 1;
    }
    double df1_dx2(double x1, double x2){
        return -sin(x2);
    }
    double df2_dx1(double x1, double x2){
        return 1/(x1 + 1);
    }
    double df2_dx2(double x1, double x2){
        return 1;
    }
}

namespace sifunc {
    double phi1(double x1, double x2) {
        return 1 + cos(x2);
    }
    double phi2(double x1, double x2) {
        return 3 + log(x1 + 1);
    }
    double dphi1_dx1(double x2) {
        return 0;
    }
    double dphi1_dx2(double x2) {
        return -sin(x2);
    }
    double dphi2_dx1(double x1) {
        return 1.0 / (x1 + 1);
    }
    double dphi2_dx2(double x1) {
        return 0;
    }
}

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 = {{nfunc::df1_dx1, nfunc::df1_dx2}, {nfunc::df2_dx1, nfunc::df2_dx2}};
    vector<vector<function<double(double, double)>>> A_1 = {{nfunc::f1, nfunc::df1_dx2}, {nfunc::f2, nfunc::df2_dx2}};
    vector<vector<function<double(double, double)>>> A_2 = {{nfunc::df1_dx1, nfunc::f1}, {nfunc::df2_dx1, nfunc::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){
        counter += 1;

        x_curr_1 = x_next_1;
        x_curr_2 = x_next_2;

        x_next_1 = sifunc::phi1(x_next_1, x_next_2);
        x_next_2 = sifunc::phi2(x_next_1, x_next_2);
    }
    return {x_next_1, x_next_2, counter};
}


double convergence(double max_dphi1, double max_dphi2) {
    double row1 = fabs(sifunc::dphi1_dx1(max_dphi1)) + fabs(sifunc::dphi1_dx2(max_dphi1));
    double row2 = fabs(sifunc::dphi2_dx1(max_dphi2)) + fabs(sifunc::dphi2_dx2(max_dphi2));
    return max(row1, row2);
}


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

    double q = convergence(0.5, 2.5);

    tie(res_1, res_2, counter) = newton(0.0, 3.0, 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(0.0, 3.0, 0.7, epsillon);
    cout << "Simple iterations method" << endl << "x1 = " << res_1 << endl << "x2 = " << res_2 << endl << "Iteration count = " << counter << endl << endl;

    return 1;
}
```