Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ
КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.1

НА ТЕМУ:

Численные методы решения нелинейных уравнений

Студент	ИУ5-14Б		Г.И. Корнеев
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсовой работы			М.И. Колосов
		(подпись, дата)	(И.О. Фамилия)

2025 г.

Цель: Изучить численные методы решения нелинейных уравнений

Задачи:

1. Найти корень уравнения по варианту методом половинного деления, методом простых итераций, методом Ньютона с погрешностью ε < 0.000001
2. Выполнить п.1 для ε < 0.00000000001
3. Результаты выполнения представить в виде таблицы

Ход работы:

Заголовок и функции представлены на рисунке 1

src > methods.cpp > ...

```cpp
#include "methods.h"

#include <cmath>       // fabs
#include <fstream>     // ofstream (CSV)
#include <iomanip>     // setprecision, fixed
#include <stdexcept>   // exceptions

/**********************
 * Исходная функция и производная
 **********************/

// f(x) = x^3 - 2
double f(double x) {
    return x*x*x - 2.0;
}

// f'(x) = 3x^2
double df(double x) {
    return 3.0 * x * x;
}
```

Рис.1-заголовок и функции

Методы

Метод половинного деления представлен на рисунке 2

```cpp
* 1) Метод половинного деления (бисекции)
***********************/
MethodResultBisection solve_bisection(double a, double b, double eps, int Nmax) {
    if (a >= b) throw std::invalid_argument("bisection: a >= b");
    double fa = f(x: a);
    double fb = f(x: b);
    if (fa * fb > 0.0) {
        // Если знаки одинаковые — не гарантируется наличие корня на отрезке
        throw std::invalid_argument("bisection: f(a) and f(b) must have opposite signs");
    }

    MethodResultBisection res{};
    res.eps = eps;

    for (int n = 0; n < Nmax; ++n) {
        double c  = 0.5 * (a + b);
        double fc = f(x: c);

        // Лог итерации: интервал, середина, значение функции
        res.rows.push_back({n, a, b, c, fc});

        // Критерии останова:
        // 1) |f(c)| < eps  или
        // 2) половина длины интервала < eps
        if (std::fabs(fc) < eps || 0.5 * (b - a) < eps) {
            res.root       = c;
            res.f_at_root  = fc;
            res.iterations = n + 1;
            return res;
        }

        // Выбор подынтервала в зависимости от знака
        if (fa * fc < 0.0) {
            b  = c;
            fb = fc;
        } else {
            a  = c;
            fa = fc;
        }
    }

    // Если вышли по Nmax — возвращаем текущее лучшее
    double c = 0.5 * (a + b);
    res.root       = c;
    res.f_at_root  = f(x: c);
    res.iterations = Nmax;
    return res;
}
```

Рис.2 - Метод половинного деления

Метод простых итераций представлен на рисунке 3

```cpp
* 2) Метод простых итераций (фиксированной точки)
***********************/
MethodResultIter solve_fixed_point(double x0, double eps, int Nmax) {
    // Контрактная функция: phi(x) = 0.5*(x + 2/x^2),
    // в корне |phi'(xi)| = 0.5 < 1 => локальная сходимость.
    auto phi: (lambda) = [](double x)-> double {
        return 0.5 * (x + 2.0 / (x * x));
    };

    MethodResultIter res{};
    res.eps = eps;

    double x = x0;
    for (int n = 0; n < Nmax; ++n) {
        // Страховка от деления на 0
        if (std::fabs(x) < 1e-14) x = 1e-6;

        double xn1   = phi(x);
        double delta = std::fabs(xn1 - x);
        double r     = std::fabs(f(xn1));

        res.rows.push_back({n, xn1, f(x: xn1), delta, r});

        if (delta < eps || r < eps) {
            res.root       = xn1;
            res.f_at_root  = f(x: xn1);
            res.iterations = n + 1;
            return res;
        }
        x = xn1;
    }

    res.root       = x;
    res.f_at_root  = f(x);
    res.iterations = Nmax;
    return res;
}
```

Рис.3 - Метод простых итераций

Метод Ньютона представлен на рисунке 4

```
 * 3) Метод Ньютона
 ***********************/
MethodResultIter solve_newton(double x0, double eps, int Nmax) {
    MethodResultIter res{};
    res.eps = eps;

    double x = x0;
    for (int n = 0; n < Nmax; ++n) {
        double y  = f(x);
        double dy = df(x);

        // Если производная слишком мала — остановим итерации
        if (std::fabs(dy) < 1e-14) {
            break;
        }

        double xn1   = x - y / dy;
        double delta = std::fabs(xn1 - x);
        double r     = std::fabs(f(xn1));

        res.rows.push_back({n, xn1, f(x: xn1), delta, r});

        if (delta < eps || r < eps) {
            res.root       = xn1;
            res.f_at_root  = f(x: xn1);
            res.iterations = n + 1;
            return res;
        }
        x = xn1;
    }

    res.root       = x;
    res.f_at_root  = f(x);
    res.iterations = (int)res.rows.size();
    return res;
}
```

Рис.4 - Метод Ньютона

Из трёх методов, самым быстрый является метод Ньютона, выполняется примерно в 3 шага, т.к. в этом методе используется больше информации о функции (значение + производная. Каждое приближение вдвое точнее предыдущего.

Результат вывода представлен на рисунке 5

```
[gregory@HP build]$ ./lab4
=== Results for eps = 1e-6 ===
Bisection: root = 1.259921073914, f(root) = 0.000000114382, iterations = 20
Fixed pt : root = 1.259920764248, f(root) = -0.000001360306, iterations = 17
Newton   : root = 1.259921049895, f(root) = 0.000000000002, iterations = 4

=== Results for eps = 1e-11 ===
Bisection: root = 1.259921049896, f(root) = 0.000000000004, iterations = 37
Fixed pt : root = 1.259921049897, f(root) = 0.000000000010, iterations = 34
Newton   : root = 1.259921049895, f(root) = 0.000000000002, iterations = 4

=== SUMMARY (by method and eps) ===
Method        eps          root                f(root)            iterations
---------------------------------------------------------------------------
Bisection     0.0000010000001.259921073914    0.000000114382    20
FixedPoint    0.0000010000001.259920764248    -0.000001360306   17
Newton        0.0000010000001.259921049895    0.000000000002    4
Bisection     0.0000000000101.259921049896    0.000000000004    37
FixedPoint    0.0000000000101.259921049897    0.000000000010    34
Newton        0.0000000000101.259921049895    0.000000000002    4
(also saved to results/summary.csv)

Подробные таблицы итераций — в ./results/ (относительно каталога запуска)
[gregory@HP build]$
```

Рисунок 5 – результат вывода

Таблица результатов представлена на рисунке 6

summary.csv

build > results > summary.csv

Method	Eps	Root	F(root)	Iterations
Bisection	0	1.26	0	20
FixedPoint	0	1.26	0	17
Newton	0	1.26	0	4
Bisection	0	1.26	0	37
FixedPoint	0	1.26	0	34
Newton	0	1.26	0	4

Рис.6 - Итоговая таблица

Работа с нейросетью

Обращение к нейросети представлено на рисунке 7

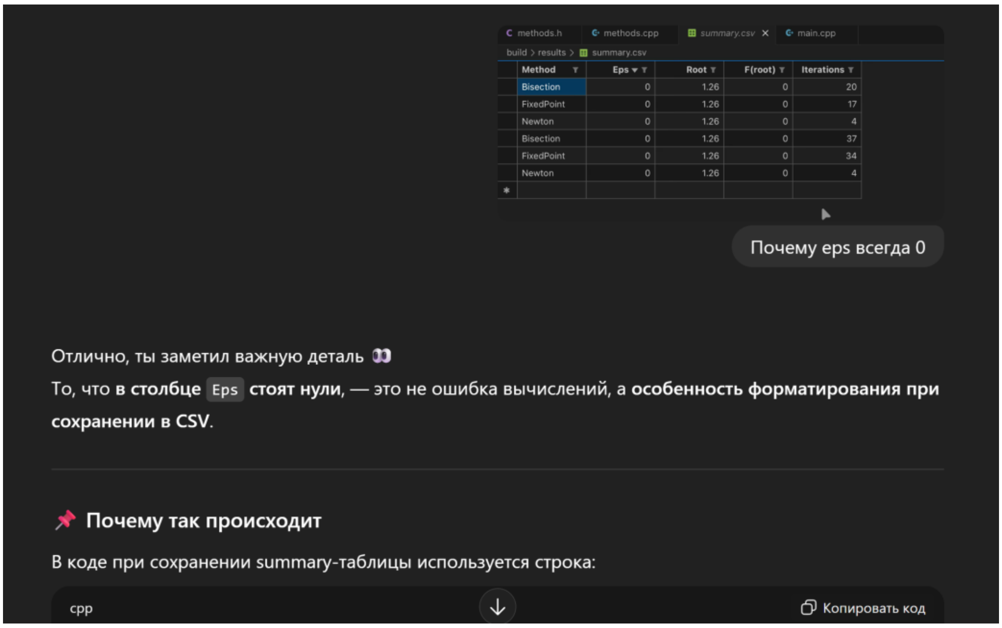

Рис.7 – взаимодействие с нейросетью

Вывод: В процессе лабораторной работы были исследованы три численных метода для решения нелинейных уравнений. Все алгоритмы продемонстрировали успешное нахождение корней с заданной точностью. Практическое использование показало преимущества и недостатки каждого подхода в зависимости от требований к точности и скорости вычислений.