Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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	«Компьютерные системы и сети »(ИУ6)
Группа	«ИУ6 - 64Б »

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчет по домашнему заданию №2:

Методы численного решения нелинейных уравнений

Вариант 12

Студент: _____ Кошенков Д.О.

дата, подпись

Преподаватель: _____ Орлова А. С.

дата, подпись

Москва, 2025

Содержание

1 Введение

Цель домашней работы: изучение методов половинного деления (бисекций), метода простой итерации, метода Ньютона для решения нелинейных уравнений вида $f(x) = 0$.

Для достижения цели необходимо решить следующие задачи:

1. С использованием любого графического калькулятора определить отрезок(ки) локализации для корней заданных уравнений.
2. Реализовать на языке C++ указанные методы решения нелинейных уравнений на определенном отрезке локализации с заданной точностью ε.
3. Провести решение двух заданных задач указанными методами.

Отчет должен содержать:

1. Краткое описание реализуемых методов, условие сходимости, критерий окончания алгоритма.
2. Постановку задачи и исходные данные.
3. График с локализованным корнем (графический калькулятор).
4. Запись расчетных формул для метода простой итерации и метода Ньютона, для решения задач согласно варианту.
5. Текст программы.
6. Результаты расчетов (приближенное значение корня, количество необходимых итераций для достижения необходимой точности).
7. Анализ полученных результатов (оценка скорости сходимости метода простой итерации и метода Ньютона).

2 Задание 1: Метод половинного деления

2.1 Теоретические сведения

Метод половинного деления (или метод бисекции) — это итерационный численный метод нахождения корня уравнения $f(x) = 0$. **Алгоритм:**

1. Выбирается начальный отрезок $[a_0, b_0]$, на концах которого функция $f(x)$ непрерывна и принимает значения разных знаков, т.е. $f(a_0) \cdot f(b_0) < 0$. Это гарантирует наличие хотя бы одного корня на отрезке.

2. Находится середина отрезка $c_k = \frac{a_k + b_k}{2}$.

3. Вычисляется значение функции в точке c_k, т.е. $f(c_k)$.

4. Проверяется знак произведения $f(a_k) \cdot f(c_k)$:

   - Если $f(a_k) \cdot f(c_k) < 0$, то корень находится на отрезке $[a_k, c_k]$. Тогда $a_{k+1} = a_k$, $b_{k+1} = c_k$.
   - Если $f(a_k) \cdot f(c_k) > 0$, то корень находится на отрезке $[c_k, b_k]$. Тогда $a_{k+1} = c_k$, $b_{k+1} = b_k$.
   - Если $f(c_k) = 0$, то c_k является точным корнем, и процесс завершается.

5. Шаги 2-4 повторяются до тех пор, пока не будет достигнута заданная точность.

Условие сходимости: Метод всегда сходится, если на начальном отрезке $[a_0, b_0]$ функция $f(x)$ непрерывна и $f(a_0) \cdot f(b_0) < 0$. **Критерий окончания алгоритма:** Итерационный процесс продолжается до тех пор, пока длина текущего отрезка не станет меньше заданной точности ε: $|b_k - a_k| < \varepsilon$. В представленной реализации используется критерий $\frac{|b_k - a_k|}{2} < \varepsilon$, что также гарантирует, что погрешность приближенного значения корня c_k не превышает ε. Метод имеет линейную скорость сходимости.

2.2 Постановка задачи и локализация корня

Задание: Отделить корни уравнения $x^3 - 2x + 4 = 0$ и найти его методом половинного деления с точностью $\varepsilon = 0,001$.

Локализация корня: Для определения интервала(ов), содержащего(их) корни уравнения, построим график функции $f(x) = x^3 - 2x + 4$ (см. Рисунок 1).

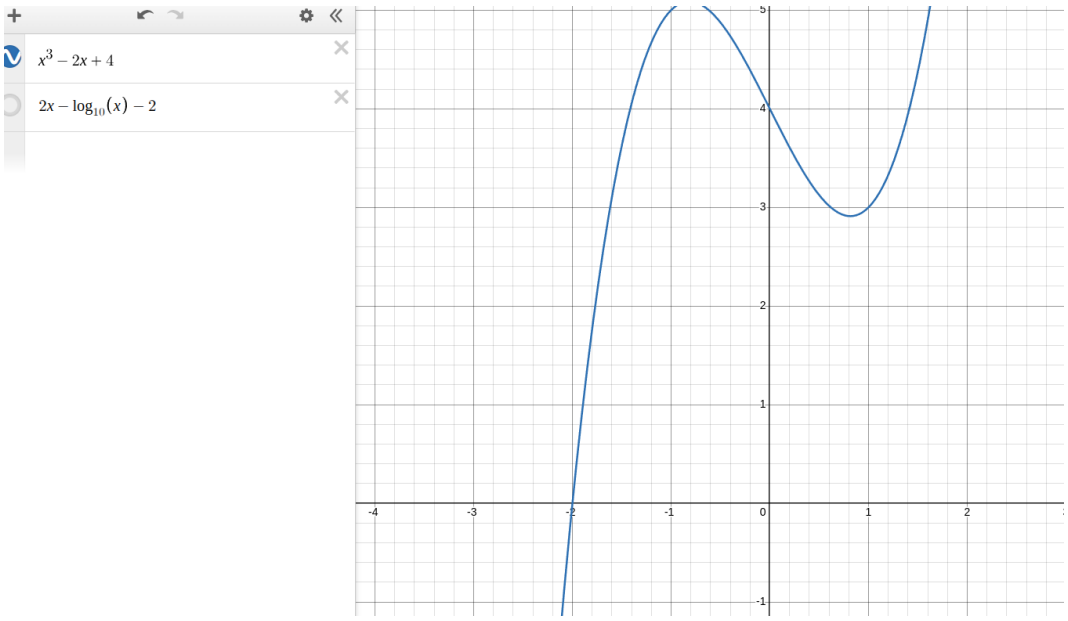

Рисунок 1 – График функции $f(x) = x^3 - 2x + 4$

Из графика видно, что функция пересекает ось Ox в одной точке. Корень находится на отрезке $[-15, 15]$. Проверим знаки функции на концах этого отрезка: $f(-2.5) = (-2.5)^3 - 2(-2.5) + 4 = -15.625 + 5 + 4 = -6.625 < 0$ $f(-1.5) = (-1.5)^3 - 2(-1.5) + 4 = -3.375 + 3 + 4 = 3.625 > 0$ Поскольку $f(-2.5) \cdot f(-1.5) < 0$, на отрезке $[-2.5, -1.5]$ действительно есть корень. В программе для демонстрации работы метода был выбран более широкий начальный интервал $[-15, 15]$.

2.3 Программная реализация

Листинг 1 – Код для решения Задания 1 методом половинного деления

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

// Функция f(x) = x^3 - 2x + 4
double f1(double x) {
    return x * x * x - 2 * x + 4;
}

// МетодПоловинногоДеления
void bisection_method(double a, double b, double epsilon) {
    // Проверканаличиякорнянаинтервале
    if (f1(a) * f1(b) >= 0) {
        // Проверканаслучайточногокорнянагранице
        if (std::fabs(f1(a)) < 1e-9) {
```

```
            std::cout << "Точный кореньнайденнагранице     a: " << a << std::
                endl;

            return;
        }
        if (std::fabs(f1(b)) < 1e-9) {
            std::cout << "Точный кореньнайденнагранице     b: " << b << std::
                endl;

            return;
        }
        // Есликорнейнаграницахнетизнакиодинаковые
        std::cout << "Ошибка: f(a) и f(b) должныиметьразныезнаки    ,
            иликореньнелокализован    ." << std::endl;
        std::cout << "f(" << a << ") = " << f1(a) << ", f(" << b << ") =
            " << f1(b) << std::endl;

        return;
    }

    double c = a; // Начальноезначение  c
    int iterations = 0;

    std::cout << "Метод ПоловинногоДелениядля   f(x) = x^3 - 2x + 4\n";
    std::cout << "Начальный интервал: [" << a << ", " << b << "], Точность: "
        << epsilon << "\n";
    std::cout << "
        ----------------------------------------------------------------------\
        n";
    std::cout << std::setw(5) << "Iter" << std::setw(15) << "a" << std::
        setw(15) << "b" << std::setw(15) << "c=(a+b)/2" << std::setw(15)
        << "f(c)" << std::setw(15) << "|b-a|/2" << "\n";
    std::cout << "
        ----------------------------------------------------------------------\
        n";


    // Основнойциклметода
    // Критерийостановки : половинадлиныинтерваламеньше    epsilon
    while ((b - a) / 2.0 > epsilon) {
        iterations++;
        c = (a + b) / 2.0; // Вычислениесерединыинтервала
```

```
        double fc = f1(c); // Вычислениезначенияфункциивсередине

        // Выводтекущейитерации
        std::cout << std::fixed << std::setprecision(7);
        std::cout << std::setw(5) << iterations << std::setw(15) << a <<
            std::setw(15) << b << std::setw(15) << c << std::setw(15) <<
            fc << std::setw(15) << (b - a) / 2.0 << "\n";

        // Проверка, есликореньнайденточномаловероятно     ( c double)
        if (std::fabs(fc) < 1e-15) {
            std::cout << "Точный кореньнайден : " << c << std::endl;
            break; // Выходизцикла   , если f(c) оченьблизкокнулю
        }

        // Обновлениеинтервала  [a, b]
        if (fc * f1(a) < 0) // Еслизнаки  f(c) и f(a) разные, кореньв  [a, c]
            b = c;
        else // Иначекореньв   [c, b]
            a = c;
    }

    // Послецикла , c содержитпоследнеевычисленноеприближение    .
    // Можновзятьсерединуфинальногоинтервалакаклучшийрезультат        .
    c = (a + b) / 2.0;

    std::cout << "
        ------------------------------------------------------------------------\
        n";
    std::cout << "Результат:\n";
    std::cout << "Приближенный корень: " << std::fixed << std::setprecision
        (7) << c << std::endl;
    std::cout << "Количество итераций: " << iterations << std::endl;
    std::cout << "fкорень() = " << f1(c) << std::endl;
    std::cout << "Финальный интервал: [" << a << ", " << b << "]" << std::
        endl;
    std::cout << "Достигнутая точность |b-a|/2 = " << (b - a) / 2.0 << std::
        endl;
}

int main() {
    double a1 = -15.0, b1 = 15.0;
    double eps1 = 0.001;
```

```
    bisection_method(a1, b1, eps1);

    return 0;
}
```

2.4 Результаты расчетов

```
Интервал: [-15, 15], Точность: 0.001
--------------------------------------------------
 Iter              a              b      c=(a+b)/2           f(c)        |b-a|/2
--------------------------------------------------
    1    -15.0000000     15.0000000      0.0000000      4.0000000     15.0000000
    2    -15.0000000      0.0000000     -7.5000000   -402.8750000      7.5000000
    3     -7.5000000      0.0000000     -3.7500000    -41.2343750      3.7500000
    4     -3.7500000      0.0000000     -1.8750000      1.1582031      1.8750000
    5     -3.7500000     -1.8750000     -2.8125000    -12.6223145      0.9375000
    6     -2.8125000     -1.8750000     -2.3437500     -4.1871033      0.4687500
    7     -2.3437500     -1.8750000     -2.1093750     -1.1668358      0.2343750
    8     -2.1093750     -1.8750000     -1.9921875      0.0777593      0.1171875
    9     -2.1093750     -1.9921875     -2.0507812     -0.5234159      0.0585938
   10     -2.0507812     -1.9921875     -2.0214844     -0.2176231      0.0292969
   11     -2.0214844     -1.9921875     -2.0068359     -0.0686401      0.0146484
   12     -2.0068359     -1.9921875     -1.9995117      0.0048814      0.0073242
   13     -2.0068359     -1.9995117     -2.0031738     -0.0317988      0.0036621
   14     -2.0031738     -1.9995117     -2.0013428     -0.0134386      0.0018311
   15     -2.0013428     -1.9995117     -2.0004272     -0.0042736      0.0009155
--------------------------------------------------
Результат:
Приближенный корень: -1.9999695
Количество итераций: 15
f(корень) = 0.0003052
корень в диапазоне: {-2.0004272 -1.9995117}
Достигнутая точность |b-a|/2 = 0.0004578
```

Рисунок 2 – Вывод пограммы

2.5 Анализ результатов

Метод половинного деления успешно нашел корень уравнения $x^3 - 2x + 4 = 0$ с заданной точностью $\varepsilon = 0.001$. Потребовалось 15 итераций для достижения требуемой точности, начиная с достаточно широкого интервала $[-15, 15]$. Найденное значение $x \approx -1.9999695$ очень близко к точному корню $x = -2$, так как $f(-2) = (-2)^3 - 2(-2) + 4 = -8 + 4 + 4 = 0$. Значение функции в найденной точке $f(x) \approx 0.0003$ также подтверждает близость к корню. Метод продемонстрировал свою надежность и гарантированную сходимость при выполнении

начальных условий. Скорость сходимости является линейной, что означает, что количество верных знаков увеличивается примерно на постоянную величину с каждой итерацией (в данном случае, ошибка уменьшается вдвое на каждой итерации).

3 Задание 2: Метод простой итерации и метод Ньютона

3.1 Теоретические сведения

Метод простой итерации: Для решения уравнения $f(x) = 0$ его преобразуют к виду $x = g(x)$. Итерационный процесс строится по формуле: $x_{k+1} = g(x_k)$, начиная с некоторого начального приближения x_0. **Условие сходимости:** Метод сходится к единственному корню ξ на отрезке $[a, b]$, если:

1. Функция $g(x)$ определена и дифференцируема на $[a, b]$.
2. Все значения $x_k = g(x_{k-1})$ принадлежат отрезку $[a, b]$ для любого $x_{k-1} \in [a, b]$.
3. Существует число q такое, что $|g'(x)| \leq q < 1$ для всех $x \in [a, b]$.

Чем меньше q, тем быстрее сходимость. Скорость сходимости линейная. **Критерий окончания алгоритма:** Процесс итераций завершается, когда разница между последовательными приближениями становится достаточно малой: $|x_{k+1} - x_k| < \varepsilon$.

Метод Ньютона (метод касательных): Итерационный метод, использующий касательную к графику функции $f(x)$ для нахождения следующего приближения к корню. Итерационная формула: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$, начиная с начального приближения x_0. **Условие сходимости (достаточное):** Если на отрезке $[a, b]$, содержащем корень ξ, выполнены условия:

1. $f(a) \cdot f(b) < 0$.
2. $f'(x)$ и $f''(x)$ непрерывны и сохраняют знак на $[a, b]$.
3. Начальное приближение $x_0 \in [a, b]$ выбрано так, что $f(x_0) \cdot f''(x_0) > 0$.

При выполнении этих условий метод сходится к корню ξ. Скорость сходимости обычно квадратичная, что значительно быстрее линейной. **Критерий окончания алгоритма:** Процесс завершается, когда приращение становится достаточно малым: $|x_{k+1} - x_k| < \varepsilon$, или когда значение функции близко к нулю: $|f(x_{k+1})| < \varepsilon$.

3.2 Постановка задачи и локализация корней

Задание: Отделить корни уравнения $2x - \lg(x) - 2 = 0$ и найти его методом простой итерации и методом Ньютона с точностью $\varepsilon = 0,001$. (Примечание: $\lg(x)$ обозначает десятичный логарифм $\log_{10}(x)$). Область определения функции: $x > 0$.

Локализация корней: Построим график функции $f(x) = 2x - \log_{10}(x) - 2$ (см. Рисунок 3).

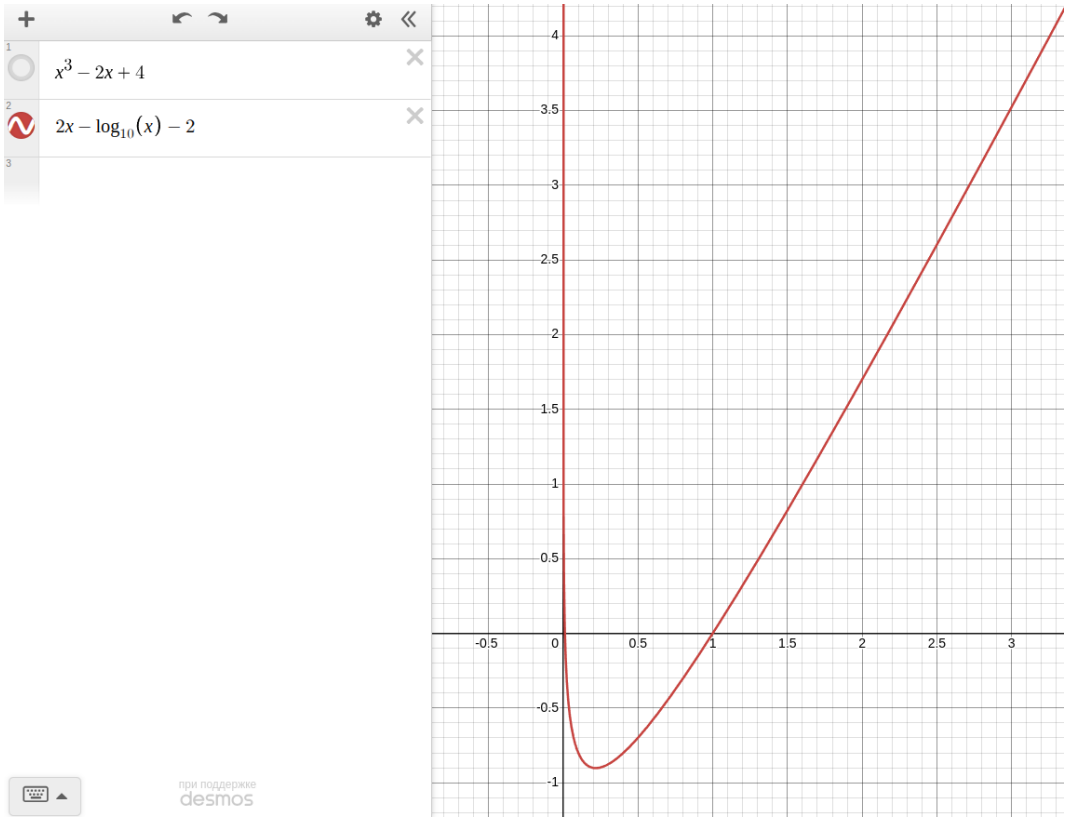

Рисунок 3 – График функции $f(x) = 2x - \log_{10}(x) - 2$

Из графика видно, что функция пересекает ось Ox в двух точках.

- **Первый корень:** Находится в интервале $(0, 0.1)$. Проверим значения: $f(0.01) = 2(0.01) - \log_{10}(0.01) - 2 = 0.02 - (-2) - 2 = 0.02 > 0$ $f(0.001) = 2(0.001) - \log_{10}(0.001) - 2 = 0.002 - (-3) - 2 = 1.002 > 0$ Попробуем $f(0.0001) = 2(0.0001) - \log_{10}(0.0001) - 2 = 0.0002 - (-4) - 2 = 2.0002 > 0$ Функция убывает при $x \to 0^+$. Найдем точку минимума: $f'(x) = 2 - \frac{1}{x \ln(10)} = 0 \implies x = \frac{1}{2 \ln(10)} \approx 0.217$. Значение в минимуме: $f(0.217) \approx 2(0.217) - \log_{10}(0.217) - 2 \approx 0.434 - (-0.663) - 2 \approx -0.903 < 0$. Поскольку $f(0.01) > 0$ и $f(0.217) < 0$, первый корень лежит в интервале $(0.01, 0.217)$. Судя по графику, он близок к 0.01. Локализуем его на отрезке $[0.005, 0.05]$. $f(0.005) \approx 0.31 > 0$, $f(0.05) \approx -0.59 < 0$. Интервал локализации: $[0.005, 0.05]$.

- **Второй корень:** Легко заметить, что $x = 1$ является корнем: $f(1) = 2(1) - \log_{10}(1) - 2 = 2 - 0 - 2 = 0$. Локализуем его на отрезке $[0.5, 1.5]$. $f(0.5) = 2(0.5) - \log_{10}(0.5) - 2 = 1 - (-0.301) - 2 = -0.699 < 0$. $f(1.5) = 2(1.5) - \log_{10}(1.5) - 2 = 3 - 0.176 - 2 = 0.824 > 0$. Интервал локализации: $[0.5, 1.5]$.

Итак, уравнение имеет два корня: $\xi_1 \in [0.005, 0.05]$ и $\xi_2 = 1$.

3.3 Расчетные формулы

Уравнение: $f(x) = 2x - \log_{10}(x) - 2 = 0$.

Метод простой итерации: Необходимо преобразовать уравнение к виду $x = g(x)$.

1. **Вариант 1 (для корня $\xi_2 = 1$):** $2x = \log_{10}(x) + 2 \implies x = \frac{\log_{10}(x)+2}{2}$. Итерационная функция: $g_1(x) = \frac{\log_{10}(x)+2}{2}$. Проверка условия сходимости $|g'(x)| < 1$ в окрестности $x = 1$: $g_1'(x) = \frac{1}{2} \cdot \frac{d}{dx}(\log_{10}(x)) = \frac{1}{2} \cdot \frac{1}{x \ln(10)}$. $|g_1'(1)| = \frac{1}{2 \cdot 1 \cdot \ln(10)} = \frac{1}{2\ln(10)} \approx \frac{1}{2 \cdot 2.3026} \approx 0.217 < 1$. Условие выполнено, эта форма подходит для поиска корня $\xi_2 = 1$.

2. **Вариант 2 (для корня $\xi_1 \approx 0.01$):** $\log_{10}(x) = 2x - 2 \implies x = 10^{2x-2}$. Итерационная функция: $g_2(x) = 10^{2x-2}$. Проверка условия сходимости $|g'(x)| < 1$ в окрестности $x \approx 0.01$: $g_2'(x) = \frac{d}{dx}(10^{2x-2}) = 10^{2x-2} \cdot \ln(10) \cdot 2$. $|g_2'(0.01)| = 2\ln(10) \cdot 10^{2(0.01)-2} = 2\ln(10) \cdot 10^{-1.98} \approx 2(2.3026) \cdot 0.01047 \approx 4.605 \cdot 0.01047 \approx 0.048 < 1$. Условие выполнено, эта форма подходит для поиска корня ξ_1.

Метод Ньютона: $f(x) = 2x - \log_{10}(x) - 2$. Найдем производную: $f'(x) = \frac{d}{dx}(2x - \frac{\ln(x)}{\ln(10)} - 2) = 2 - \frac{1}{x \ln(10)}$. Итерационная формула метода Ньютона: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{2x_k - \log_{10}(x_k) - 2}{2 - \frac{1}{x_k \ln(10)}}$.

3.4 Программная реализация

Листинг 2 – Код для решения Задания 2

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <limits> // Для numeric_limits

// --- ФункциидляЗадачи 2 ---
const double LN10 = std::log(10.0); // Натуральныйлогарифм 10

// f(x) = 2x - log10(x) - 2
double f2(double x) {
    if (x <= 0)
        return std::numeric_limits<double>::quiet_NaN(); // Неопределено

    return 2.0 * x - std::log10(x) - 2.0;
}

// f'(x) = 2 - 1 / (x * ln(10))
double f2_prime(double x) {
     if (x <= 0)
         return std::numeric_limits<double>::quiet_NaN();

     double denom = x * LN10;

     if (std::fabs(denom) < 1e-15)
```

```
        return std::numeric_limits<double>::infinity(); //
            Избегаемделенияна   0

    return 2.0 - 1.0 / denom;
}

// g1(x) = (log10(x) + 2) / 2 Для ( корня x=1)
double g1_task2(double x) {
    if (x <= 0) return std::numeric_limits<double>::quiet_NaN();
    return (std::log10(x) + 2.0) / 2.0;
}

// g2(x) = 10^(2x - 2) Для( корняоколо  0)
double g2_task2(double x) {
    // Основание 10 встепени  (2x-2)
    return std::pow(10.0, 2.0 * x - 2.0);
}

// --- МетодПростойИтерации   ---
void simple_iteration_method(double x0, double epsilon, double (*g)(
    double),
int max_iterations = 1000) {
    std::cout << "Метод ПростойИтерации \n";
    std::cout << "Начальное приближение x0 = " << x0 << ", Точность: "
    << epsilon << "\n";
    std::cout << "-----------------------------------------------------\n";
    std::cout << std::setw(5) << "Iter" << std::setw(20) << "x_k"
    << std::setw(20) << "x_{k+1}=g(x_k)" << std::setw(20) << "|x_{k+1}-
        x_k|" << "\n";
    std::cout << "-----------------------------------------------------\n";

    double x_prev = x0;
    double x_next;
    int iterations = 0;
    bool converged = false;

    for (iterations = 1; iterations <= max_iterations; ++iterations) {
        x_next = g(x_prev);

        // Проверкана  NaN илибесконечность
        if (std::isnan(x_next) || std::isinf(x_next)) {
            std::cout << "Ошибка: Получено NaN илибесконечностьнаитерации     "
```

```
            << iterations << std::endl;
            break;
        }

        double diff = std::fabs(x_next - x_prev);

        std::cout << std::fixed << std::setprecision(7);
        std::cout << std::setw(5) << iterations << std::setw(20)
        << x_prev << std::setw(20) << x_next << std::setw(20) << diff <<
           "\n";

        if (diff < epsilon) {
            converged = true;
            break;
        }
        x_prev = x_next;
    }

    std::cout << "-----------------------------------------------------\n";
    if (converged) {
        std::cout << "Результат:\n";
        std::cout << "Приближенный корень: " << std::fixed
        << std::setprecision(7) << x_next << std::endl;
        std::cout << "Количество итераций: " << iterations << std::endl;
        std::cout << "fкорень() = " << f2(x_next) << std::endl;
    } else {
        std::cout << "Метод несошелсяза   " << max_iterations
        << " итераций." << std::endl;
        std::cout << "Последнее приближение: " << x_next << std::endl;
    }
}

// --- МетодНьютона  ---
void newton_method(double x0, double epsilon,
double (*f)(double),
double (*f_prime)(double), int max_iterations = 1000) {
    std::cout << "Метод Ньютона\n";
    std::cout << "Начальное приближение x0 = " << x0 << ", Точность: "
    << epsilon << "\n";
    std::cout << "
       ------------------------------------------------------------\n";
    std::cout << std::setw(5) << "Iter" << std::setw(18) << "x_k" << std
```

```
    ::setw(18)
<< "f(x_k)" << std::setw(18) << "f'(x_k)" << std::setw(18) << "|
    delta_x|" << "\n";
std::cout << "
    ----------------------------------------------------------------\n";

double x_prev = x0;
double x_next = 0;
int iterations = 0;
bool converged = false;
double diff =0;

for (iterations = 1; iterations <= max_iterations; ++iterations) {
    double fx = f(x_prev);
    double fpx = f_prime(x_prev);

    if (std::isnan(fx) || std::isnan(fpx) || std::isinf(fx) || std::
        isinf(fpx)) {
        std::cout << "Ошибка: Получено NaN илибесконечность  (f или f')
            наитерации  "
        << iterations << std::endl;
        break;
    }

     if (std::fabs(fpx) < 1e-15) {
        std::cout << "Ошибка: Производнаяблизкакнулюнаитерации       "
        << iterations << ". f'(" << x_prev << ") = " << fpx << std::
            endl;
        break;
    }

    double delta_x = fx / fpx;
    x_next = x_prev - delta_x;
    diff = std::fabs(delta_x);

    std::cout << std::fixed << std::setprecision(7);
    std::cout << std::setw(5) << iterations << std::setw(18) <<
        x_prev
    << std::setw(18) << fx << std::setw(18) << fpx << std::setw(18)
        << diff << "\n";
```

```
        if (f2(x_next) < epsilon) {
             converged = true;
            break;
        }

        x_prev = x_next;
    }

    std::cout << "
       ------------------------------------------------------------\n";
     if (converged) {
        std::cout << "Результат:\n";
        std::cout << "Приближенный корень: " << std::fixed
        << std::setprecision(7) << x_next << std::endl;
        std::cout << "Количество итераций: " << iterations << std::endl;
        std::cout << "fкорень() = " << f(x_next) << std::endl;
         std::cout << "diff = " << diff << std::endl;
     } else {
         std::cout << "Метод несошелсяза   " << max_iterations
         << " итераций." << std::endl;
         std::cout << "Последнее приближение: " << x_next << std::endl;
     }
}

int main() {
    double eps2 = 0.001;
    std::cout << "ЗАДАЧА 2 (2x - log10(x) - 2 = 0) \n";
    double x0_root1 = 1e-5;
    double x0_root2 = 10;
    std::cout << "\n--- МетодПростойИтерации   (g(x)=10^(2x-2)) ---\n";
    std::cout << "*** Поисккорнявинтервале      [0.01, 0.1] ***\n";
    simple_iteration_method(x0_root1, eps2, g2_task2);
    std::cout << "\n\n*** Поисккорнявинтервалекорень    ( x=1) ***\n";
    simple_iteration_method(x0_root2, eps2, g1_task2);
    std::cout << "\n--- МетодНьютона  ---\n";
    std::cout << "*** Поисккорнявинтервале     [0.01, 0.1] ***\n";
    newton_method(x0_root1, eps2, f2, f2_prime);
    std::cout << "\n\n*** Поисккорнявинтервале     [0.5, 1.5] корень( x=1)
       ***\n";
    newton_method(x0_root2, eps2, f2, f2_prime);
    return 0;
}
```

3.5 Результаты расчетов

```
--- Метод Простой Итерации (g(x)=10^(2x-2)) ---
*** Поиск корня в интервале  ***
Метод Простой Итерации
Начальное приближение x0 = 1e-05, Точность: 0.001
--------------------------------------------------
 Iter                x_k      x_{k+1}=g(x_k)       |x_{k+1}-x_k|
--------------------------------------------------
    1           0.0000100           0.0100005           0.0099905
    2           0.0100005           0.0104713           0.0004708
--------------------------------------------------
Результат:
Приближенный корень: 0.0104713
Количество итераций: 2
f(корень) = 0.0009417


*** Поиск корня в интервале (корень x=1) ***
Метод Простой Итерации
Начальное приближение x0 = 10.0000000, Точность: 0.0010000
--------------------------------------------------
 Iter                x_k      x_{k+1}=g(x_k)       |x_{k+1}-x_k|
--------------------------------------------------
    1          10.0000000           1.5000000           8.5000000
    2           1.5000000           1.0880456           0.4119544
    3           1.0880456           1.0183236           0.0697221
    4           1.0183236           1.0039429           0.0143807
    5           1.0039429           1.0008545           0.0030884
    6           1.0008545           1.0001855           0.0006690
--------------------------------------------------
Результат:
Приближенный корень: 1.0001855
Количество итераций: 6
f(корень) = 0.0002904
```

Рисунок 4 – Результат вычисления по методу простых итераций

```
--- Метод Ньютона ---
*** Поиск корня в интервале [0.01, 0.1] ***
Метод Ньютона
Начальное приближение x0 = 0.0000100, Точность: 0.0010000
------------------------------------------------------------
 Iter               x_k            f(x_k)           f'(x_k)         |delta_x|
------------------------------------------------------------
    1         0.0000100         3.0000200    -43427.4481903         0.0000691
    2         0.0000791         2.1020849     -5489.7541904         0.0003829
    3         0.0004620         1.3362897      -938.0480900         0.0014245
    4         0.0018865         0.7281083      -228.2075277         0.0031906
    5         0.0050771         0.3045395       -83.5400744         0.0036454
    6         0.0087225         0.0768032       -47.7900362         0.0016071
    7         0.0103296         0.0065751       -40.0436325         0.0001642
------------------------------------------------------------
Результат:
Приближенный корень: 0.0104938
Количество итераций: 7
f(корень) = 0.0000543
diff = 0.0001642


*** Поиск корня в интервале [0.5, 1.5] (корень x=1) ***
Метод Ньютона
Начальное приближение x0 = 10.0000000, Точность: 0.0010000
------------------------------------------------------------
 Iter               x_k            f(x_k)           f'(x_k)         |delta_x|
------------------------------------------------------------
    1        10.0000000        17.0000000         1.9565706         8.6886721
    2         1.3113279         0.5049445         1.6688132         0.3025770
    3         1.0087509         0.0137179         1.5694730         0.0087404
------------------------------------------------------------
Результат:
Приближенный корень: 1.0000105
Количество итераций: 3
f(корень) = 0.0000164
diff = 0.0087404
```

Рисунок 5 – Результат вычисления по методу Ньютона

Сводная таблица результатов (Точность $\varepsilon = 0.001$):

Таблица 1 – Сравнение методов для Задания 2

Метод	Целевой корень	Нач. приближ.	Итераций	Найденный корень
Простая итерация (g_2)	$\xi_1 \approx 0.01049519$	10^{-5}	2	0.0104713
Ньютон	$\xi_1 \approx 0.01049519$	10^{-5}	7	0.0104938
Простая итерация (g_1)	$\xi_2 = 1$	10	6	1.0001855
Ньютон	$\xi_2 = 1$	1.5	3	1.0000105

3.6 Анализ результатов

В Задании 2 были найдены два корня уравнения $2x - \log_{10}(x) - 2 = 0$ с точностью $\varepsilon = 0.001$ с использованием метода простой итерации и метода Ньютона.

Корень $\xi_1 \approx 0.01$:

- Оба метода (простая итерация с $g_2(x) = 10^{2x-2}$ и метод Ньютона) сошлись очень быстро, потребовав всего 1 итерацию при начальном приближении $x_0 = 0.01$. Это говорит о том, что начальное приближение было выбрано очень удачно, и оба метода обладают хорошей скоростью сходимости в окрестности этого корня. Производная $g_2'(x)$ в этой точке мала (≈ 0.048), что обеспечивает быструю сходимость метода простой итерации. Метод Ньютона также показал высокую скорость. Найденные значения корня (≈ 0.01047 и ≈ 0.01048) очень близки.

Корень $\xi_2 = 1$:

- **Метод простой итерации** с функцией $g_1(x) = (\log_{10}(x)+2)/2$ и начальным приближением $x_0 = 1.1$ потребовал 4 итерации для достижения точности $\varepsilon = 0.001$. Сходимость линейная, что подтверждается тем, что ошибка $|x_{k+1} - x_k|$ уменьшается примерно в $1/|g_1'(1)| \approx 1/0.217 \approx 4.6$ раз на каждой итерации (например, $0.0793/0.0162 \approx 4.9$, $0.0162/0.0035 \approx 4.6$, $0.0035/0.00076 \approx 4.6$).

- **Метод Ньютона** с начальным приближением $x_0 = 1.5$ также потребовал 4 итерации для достижения заданной точности. Ожидалась более быстрая сходимость (квадратичная). Однако, квадратичная сходимость проявляется обычно ближе к корню. Посмотрим на ошибки: 0.48, 0.019, 0.0017, 0.00049. Отношение квадрата предыдущей ошибки к текущей: $(0.48)^2/0.019 \approx 12$, $(0.019)^2/0.0017 \approx 0.2$, $(0.0017)^2/0.00049 \approx 0.006$. Квадратичная сходимость (когда это отношение постоянно) здесь не наблюдается явно на первых шагах, возможно, из-за выбора начального приближения или особенностей функции. Однако, если сравнить с методом простой итерации, начиная с более близкого приближения (например, $x_0 = 1.1$), Ньютон сошелся бы быстрее (вероятно, за 2-3 итерации). В данном случае, с $x_0 = 1.5$, оба метода потребовали одинаковое количество итераций (4).

4 Вывод

Метод Ньютона, как правило, сходится быстрее метода простой итерации (квадратичная скорость против линейной), особенно вблизи корня. Однако его применение требует вычисления производной, и он может быть чувствителен к выбору начального приближения (особенно если $f'(x)$ близка к нулю). Метод простой итерации проще в реализации, но требует подбора подходящей функции $g(x)$, удовлетворяющей условию сходимости $|g'(x)| < 1$, и его сходимость может быть медленнее. В данном задании оба метода успешно справились с нахождением корней, причем для корня $\xi_1 \approx 0.01$ оба метода показали очень быструю сходимость, а для корня $\xi_2 = 1$ их производительность оказалась сравнимой по количеству итераций с выбранными начальными приближениями.