Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»(ИТАС)

**О Т Ч Е Т**

**о лабораторной работе № 2**

**Дисциплина: «Основы алгоритмизации и программирования»**

**Тема: «Численные методы решения уравнений»**

Выполнил студент гр. РИС-24-3б

Суяргулов Артём Олегович

(Фамилия, Имя, Отчество)

(подпись)

Проверила:

доцент

кафедры ИТАС

Полякова О.А.

(оценка) (подпись)

(дата)

Пермь 2024

**Постановка задач:**

1. Изучение численных методов решения уравнений.
2. Разработка алгоритмов для решения уравнения.
3. Составление блок-схем для каждого случая.
4. Реализация программ на языке c++.
5. Проверка работы программ, проверка ответа.
6. Построение графика, с отображением решения.

**Метод половинного деления:**

**Код программы:**#include <iostream>

#include <cmath>

using namespace std;

// Функция, корень которой мы ищем

double f(double x) {

return 0.25 \* pow(x, 3) + x - 1.2502;

}

int main() {

double a = 0, b = 2, c; // Границы отрезка [a, b], на котором ищется корень, и переменная для середины отрезка

double epsilon = pow(10, -6); // Задаем точность вычислений

int iterations = 0; // Счетчик количества итераций

// Проверка условия существования корня на отрезке [a, b]

if (f(a) \* f(b) >= 0) {

cout << "Метод половинного деления неприменим на этом интервале." << endl;

return 1; // Завершаем программу с ошибкой, если условие не выполняется

}

// Цикл метода половинного деления

while (abs(b - a) / 2 > epsilon) { // Цикл продолжается пока ширина интервала больше заданной точности

c = (a + b) / 2; // Находим середину отрезка

if (f(c) \* f(a) < 0) { // Если функция меняет знак на отрезке [a, c]

b = c; // То новый отрезок [a, c]

}

else {

a = c; // Новый отрезок [c, b]

}

iterations++; // Увеличиваем счетчик итераций

}

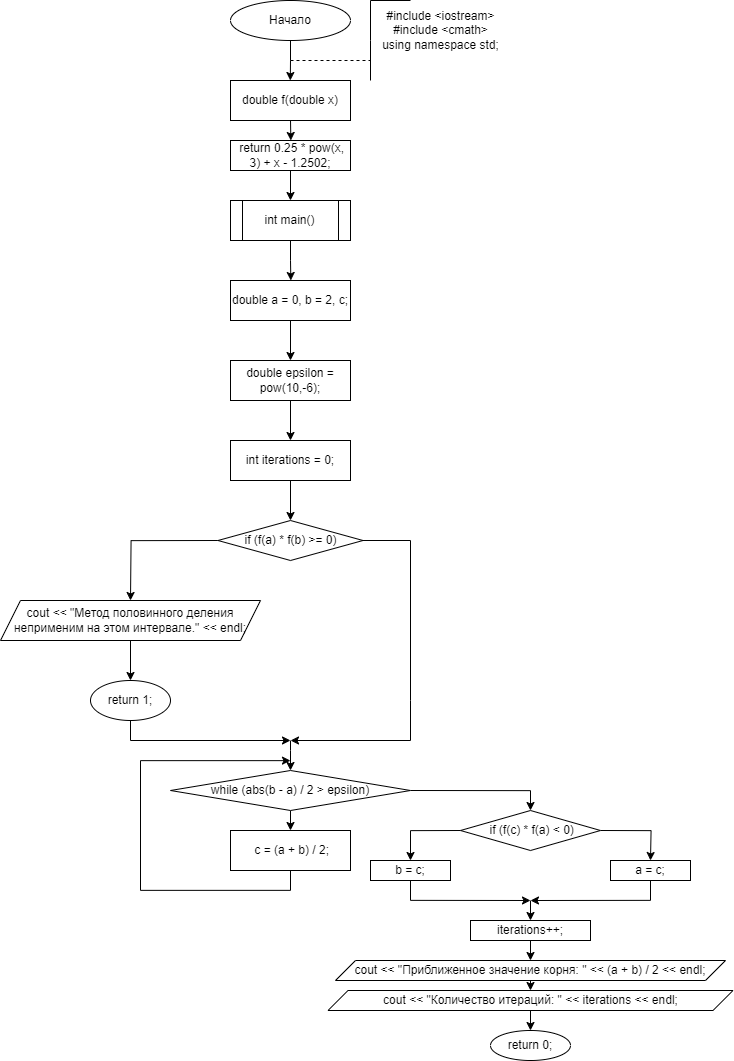
cout << "Приближенное значение корня: " << (a + b) / 2 << endl; // Выводим приближенное значение корня

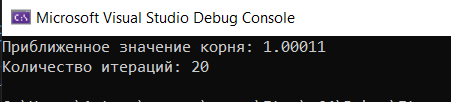
cout << "Количество итераций: " << iterations << endl; // Выводим количество итераций

return 0;

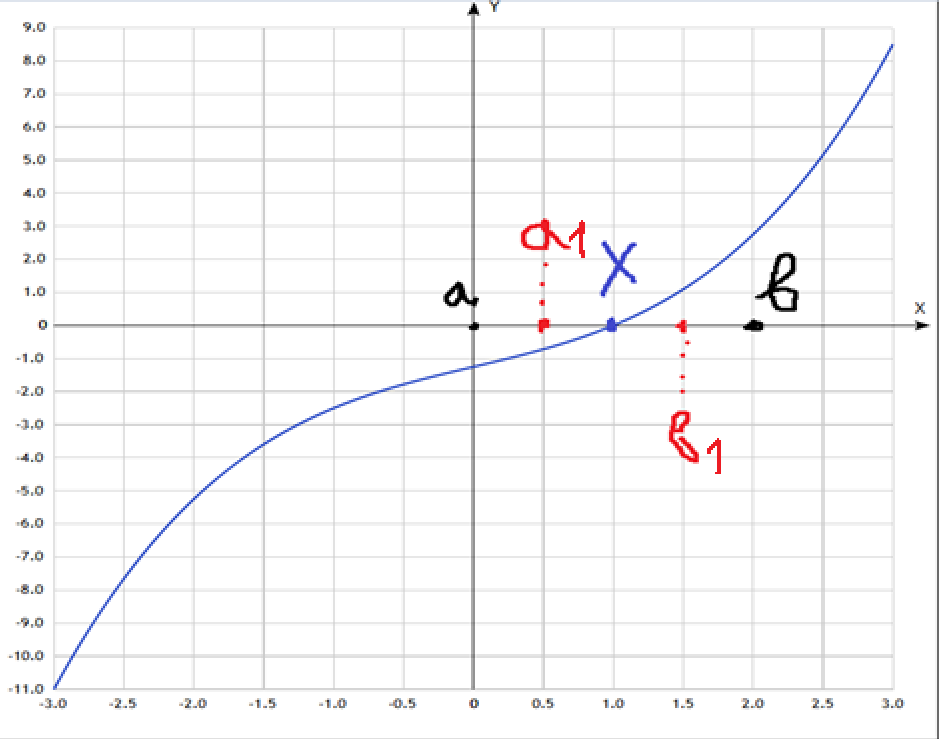
}

**Блок-схема:**



**Результаты:  
**

**График:**

**Метод Итерации:**

**Код программы:**

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

// Функция итераций для метода простой итерации. Переписано уравнение для нахождения x

double g(double x) {

return 1.2502 / (1 + 0.25 \* x \* x); // Функция g(x) преобразованное уравнение 0.25x³ + x - 1.2502 = 0 в виде x = g(x)

}

int main() {

double x = 1.0; // Начальное приближение к корню (x0)

double tol = 1e-6; // Объявляем точность для последущей остановки

double x\_next = 0; // Переменная для хранения следующего приближения к корню

// Итерационный процесс

do {

x = x\_next; // Обновляем текущее приближение значением из предыдущей итерации

x\_next = g(x); // Вычисляем следующее приближение используя функцию g(x)

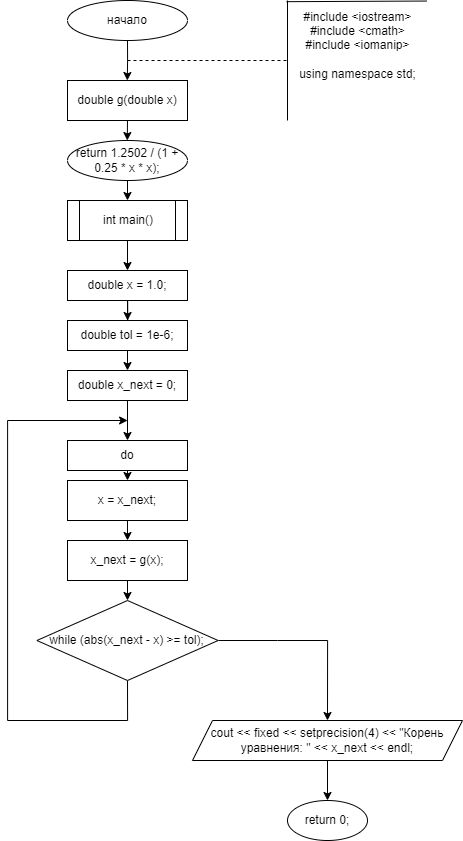
} while (abs(x\_next - x) >= tol); // Продолжаем итерации пока разница между текущим и следующим приближением больше допустимой погрешности

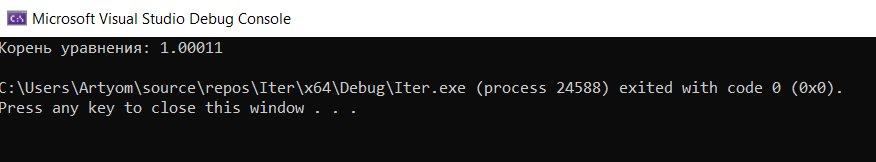
cout << fixed << setprecision(4) << "Корень уравнения: " << x\_next << endl; // Выводим результат с точностью до 4 знаков после запятой

return 0;

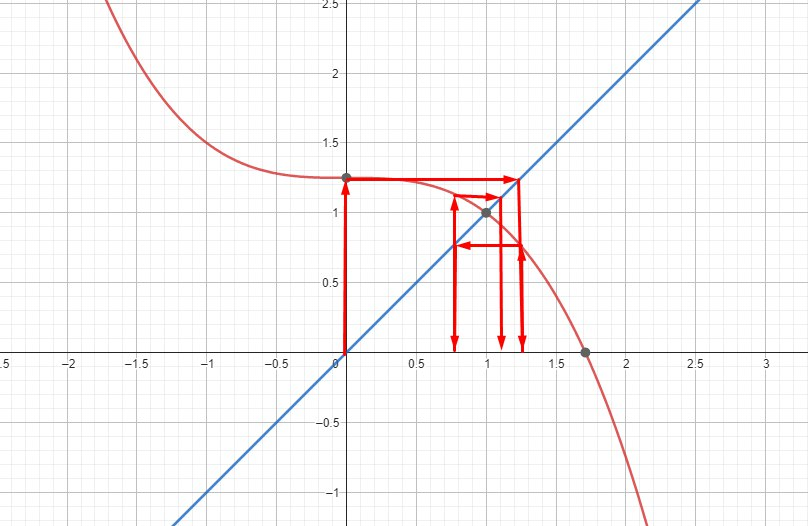
}

**Блок-схема:**

****

**Результаты:  
**

**График:**



**Метод Ньютона:**

**Код программы:**#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

// Функция, которую нужно решить

double f(double x) {

return 0.25 \* pow(x, 3) + x - 1.2502; // Определяем функцию f(x) = 0.25x³ + x - 1.2502

}

// Производная функции f(x)

double df(double x) {

return 0.75 \* pow(x, 2) + 1; // Производная f'(x) = 0.75x² + 1

}

int main() {

double x0 = 1.0; // Начальное приближение корня

double x1 = x0; // Значение x на следующей итерации

double tolerance = 1e-6; // Точность

cout << fixed << setprecision(5); // Устанавливаем формат вывода с 5 знаками после запятой

// Цикл итераций метода Ньютона

do {

double pr\_xo = df(x0); // Вычисляем значение производной в точке x0

//Проверка на деление на ноль или очень малое значение производной

if (abs(pr\_xo) < 1e-10) {

cout << "Производная близка к нулю. Метод Ньютона не сходится." << endl;

return 1; // Завершаем программу с ошибкой

}

x1 = x0 - f(x0) / pr\_xo; // Формула метода Ньютона: x\_(n+1) = x\_n - f(x\_n) / f'(x\_n)

x0 = x1; // Обновляем значение x0 для следующей итерации

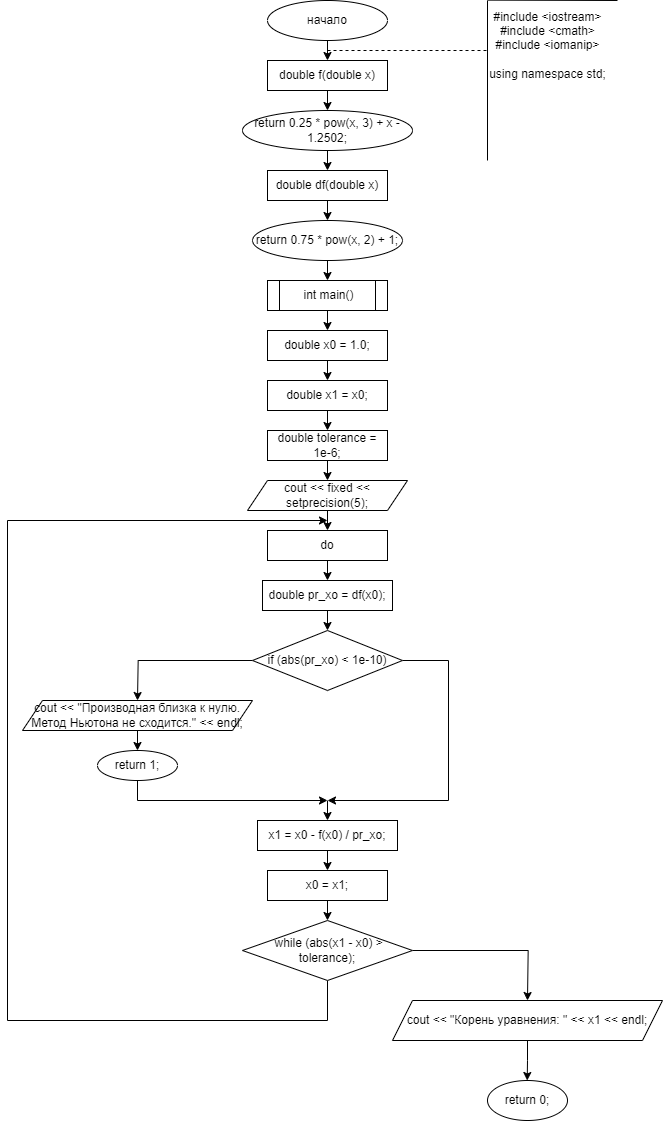
} while (abs(x1 - x0) > tolerance); // Продолжаем итерации, пока погрешность больше допустимой

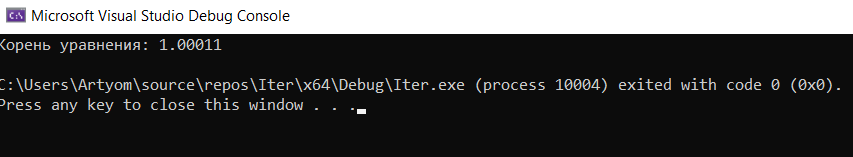
cout << "Корень уравнения: " << x1 << endl; // Выводим найденный корень

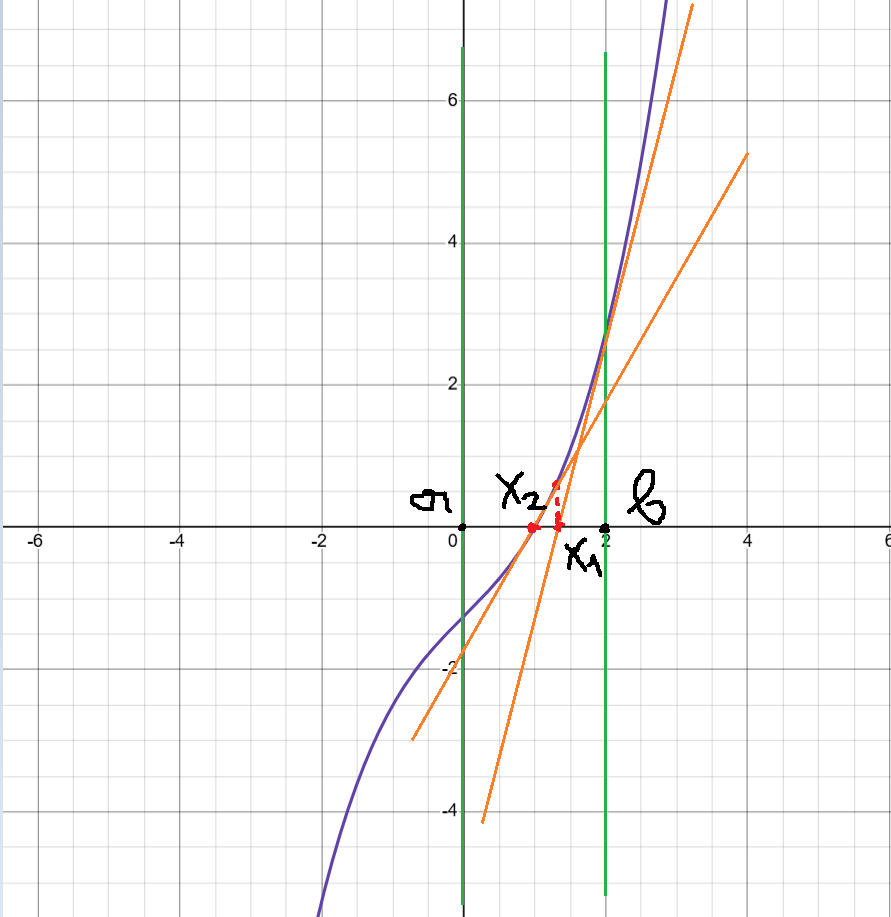
return 0;

}

**Блок-схема:**



**Результаты:  
**

**График:  
**

**Ссылки на GitHub:**

<https://github.com/Drazda1488/Laba_2_CPP>

