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»

направление подготовки: 09.04.01- «Программная инженерия»

Лабораторная работа №2

«Решение нелинейных уравнений»

Выполнил студент гр. РИС-24-3б

Ушаков Арсений Анатольевич____

Проверил:

Доц. Каф. ИТАС_______________

Ольга Андреевна Полякова______

г. Пермь, 2024

Постановка задачи.

Разработать алгоритм и написать код на языке C++ для решения данного уравнения: $x - \frac{1}{3+sin(3.6x)} = 0$.

Вариант 9.

Задано нелинейное уравнение $x - \frac{1}{3+sin(3.6x)} = 0$, отрезок [0;0.85], содержащий корень 0.2624. Точность вычислений eps = 10^{-6}.

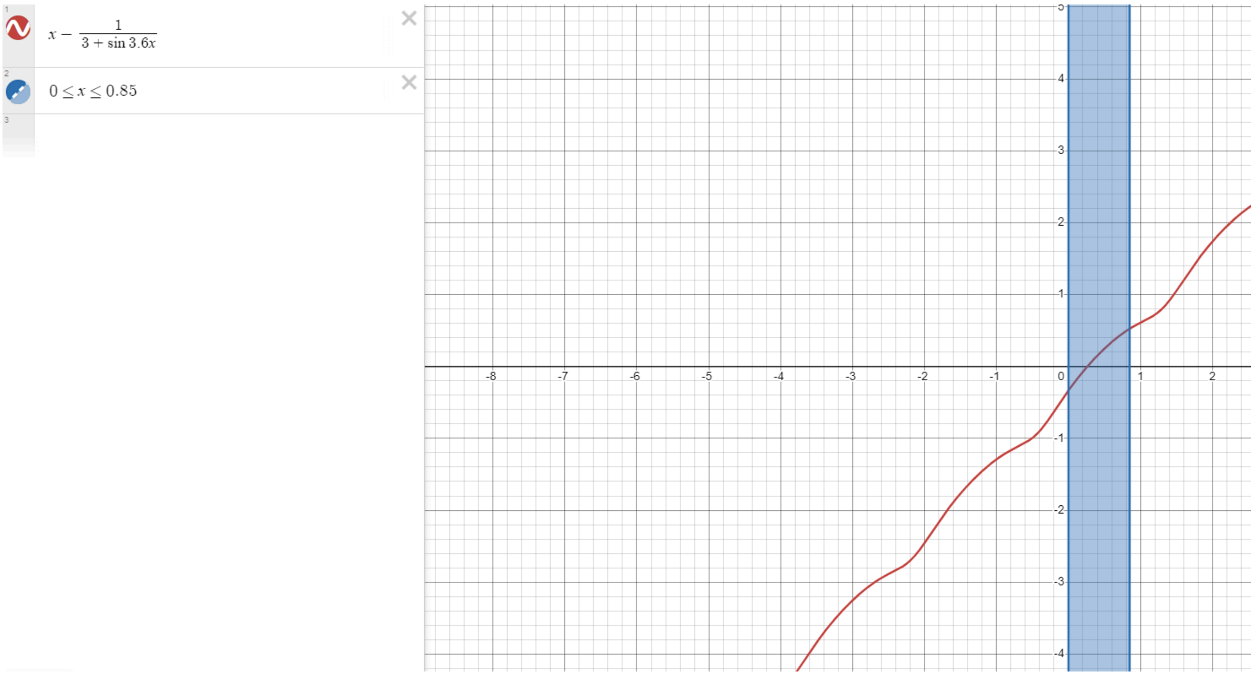

Рис. 1 - График уравнения

Метод Ньютона

1. Обозначим функцию $f(x) = x - \frac{1}{3+sin(3.6x)}$

2. Найдем первую производную от функции: $f^{'}(x) = 1 - \frac{18cos(\frac{18x}{5})}{5(3+sin(\frac{18x}{5}))^{2}}$

3. Найдем вторую производную от функции:

$$f''(x)= \frac{-3240sin(\frac{18x}{5})-1944-324sin(\frac{18x}{5})cos(\frac{18x}{5})^{2}}{25(3+sin(\frac{18x}{5}))^{4}}$$

Если для интервала [a; b] выполняется $f(a) * f''(a) > 0$ или $f(b) * f''(b) > 0$, то функция монотонна и непрерывна, и корень на интервале существует, иначе корня на интервале не существует.

Примем x0 = b, через точку (x0; f(x0)) проведем касательную к графику функции. Приближенным значением корня x1 будет пересечение касательной с осью Ox. Следующее значение вычисляется по формуле: $x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$

.

Пока eps <= |x0-xi|, проводим новые касательные и получаем новые приближенные значения корня. Когда это условие выполнено - найдено точное решение уравнения.

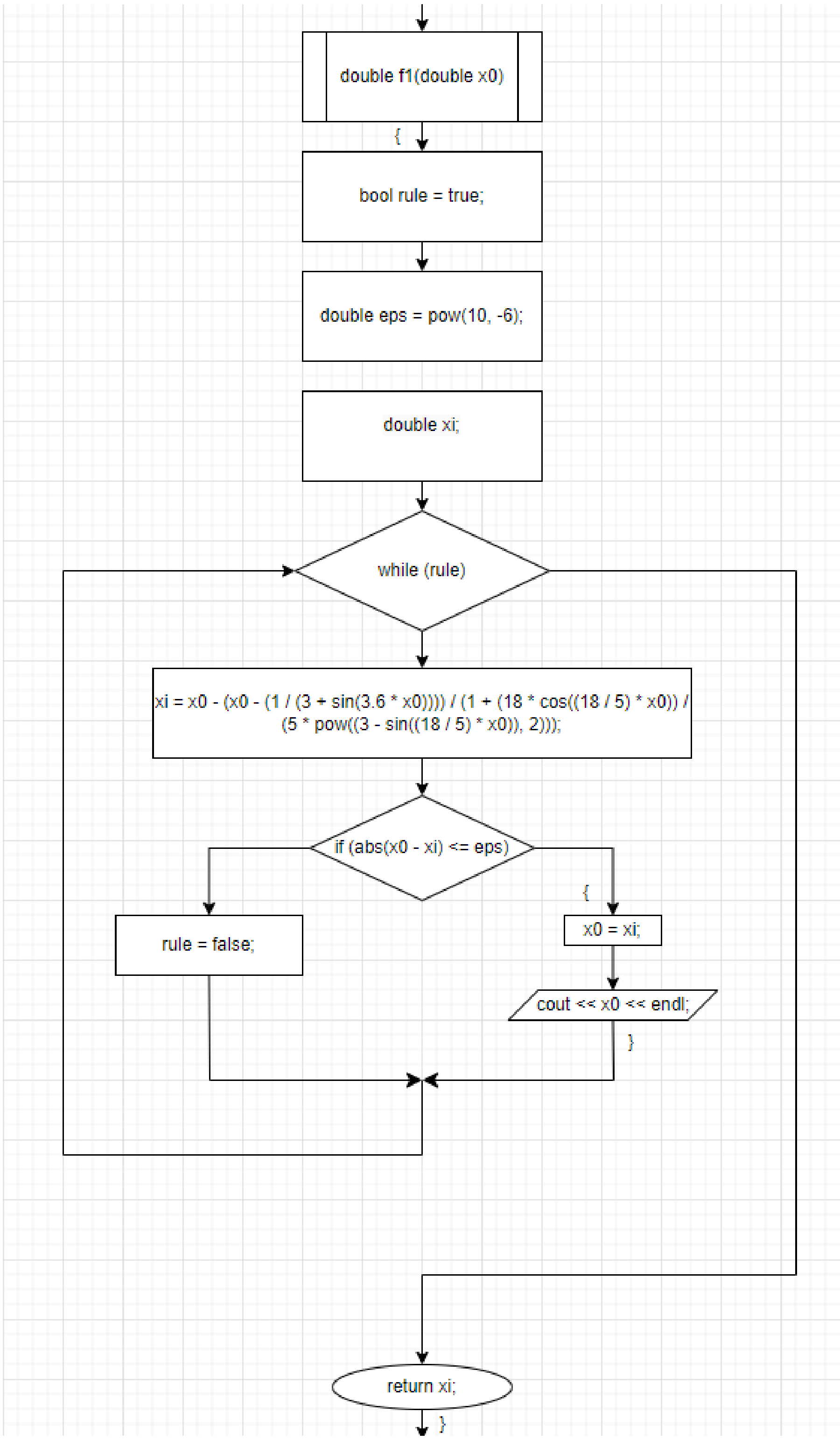

Рис. 2 - Блок-схема дополнительной функции. Метод Ньютона

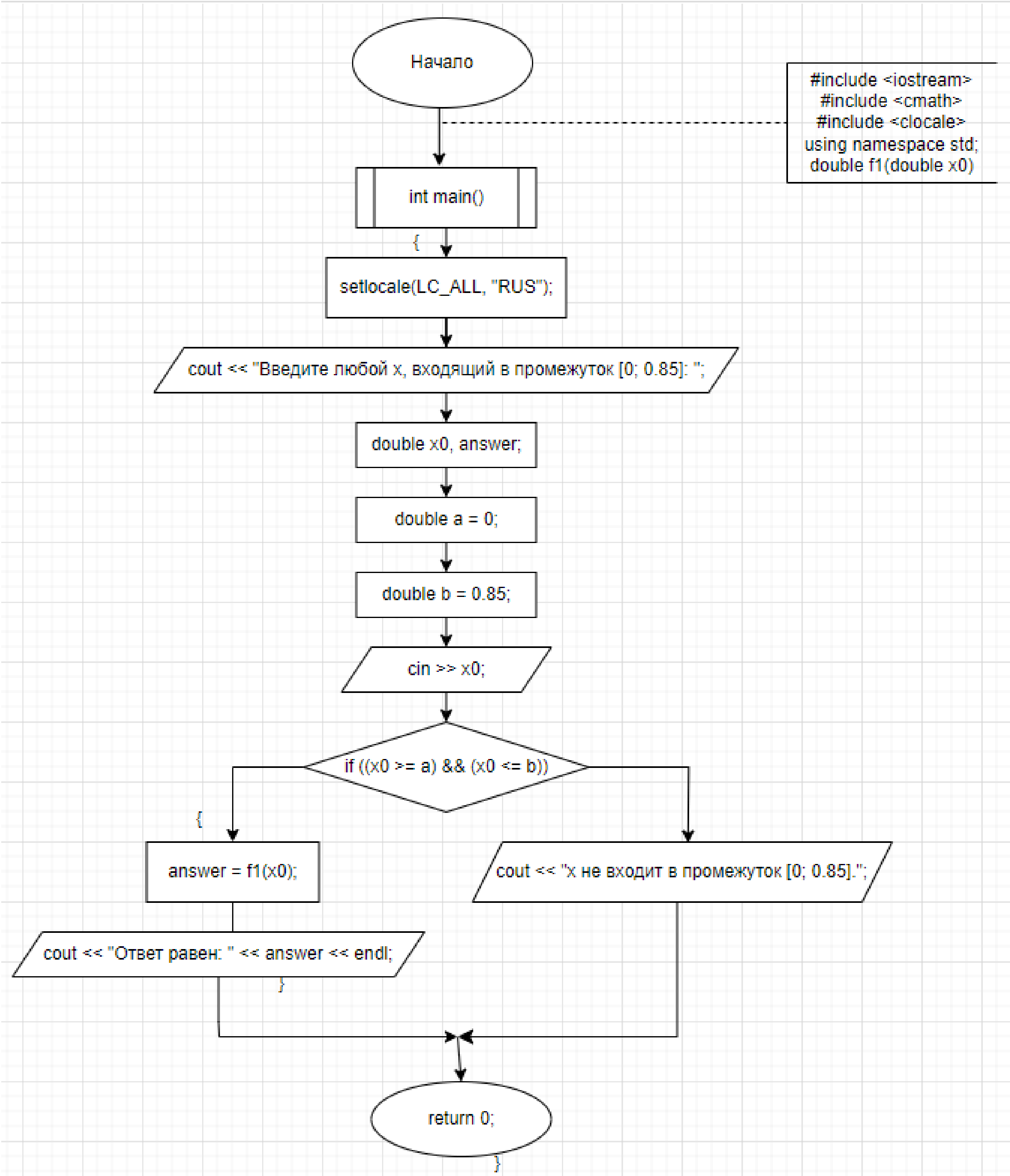

Рис. 3 - блок-схема функции main. Метод Ньютона

NEWTON (Глобальная область)

```cpp
#include <iostream>
#include <cmath>
#include <clocale>
using namespace std;
double f1(double x0)
{
    bool rule = true;
    double eps = pow(10, -6); //Точность вычисления
    double xi;
    while (rule) {
        xi = x0 - (x0 - (1 / (3 + sin(3.6 * x0)))) / (1 + (18 * cos((18 / 5) * x0)) / (5 * pow((3 - sin((18 / 5) * x0)), 2))); //Касательная к графику
        if (abs(x0 - xi) <= eps) //Проверка промежуточного корня
        {
            rule = false; //Цикл завершается
        }
        else {
            x0 = xi; //Присваиваем новое значение, чтобы найти следующее приблизительное значение корня
            cout << "Промежуточное значение = " << x0 << endl; //Вывож промежуточного значения
        }
    }
    return xi;
}
int main()
{
    setlocale(LC_ALL, "RUS");
    cout << "Введите любой x, входящий в промежуток [0; 0.85]: ";
    double x0, answer;
    double a = 0;
    double b = 0.85;
    cin >> x0;
    if ((x0 >= a) && (x0 <= b)) { //Проверяем входит ли x в промежуток
        answer = f1(x0);
        cout << "Ответ равен: " << answer << endl;
    }
    else {
        cout << "x не входит в промежуток [0; 0.85].";
    }
    return 0;
}
```

Рис. 4 - Программная реализация метода Ньютона

Консоль отладки Microsoft Visual Studio

```
Введите любой x, входящий в промежуток [0; 0.85]: 0.5
Промежуточное значение = 0.266475
Промежуточное значение = 0.263362
Промежуточное значение = 0.262651
Промежуточное значение = 0.262489
Промежуточное значение = 0.262452
Промежуточное значение = 0.262444
Промежуточное значение = 0.262442
Ответ равен: 0.262442
```

Рис. 5 - Результат выполнения программы

Найденное значение корня равно 0.2624.

Метод половинчатого деления.

Если выполняется условие $f(a) * f(b) < 0$, то график функции пересекает ось Ox в интервале [a; b]. Делим интервал пополам, полученная на половине точка x0 считается приближенным значением корня.

Отбрасываем половину, в которой не содержится корня. Если выполняется условие $f(a) * f(x0) < 0$, то правая граница интервала переносится в точку x0, иначе левая граница интервала переносится в точку x0.

Продолжаем делить интервал и отсекать ненужную половину, пока не выполнится условие $|a - b| < eps$, тогда приближенным значением корня будет являться любая граница интервала.

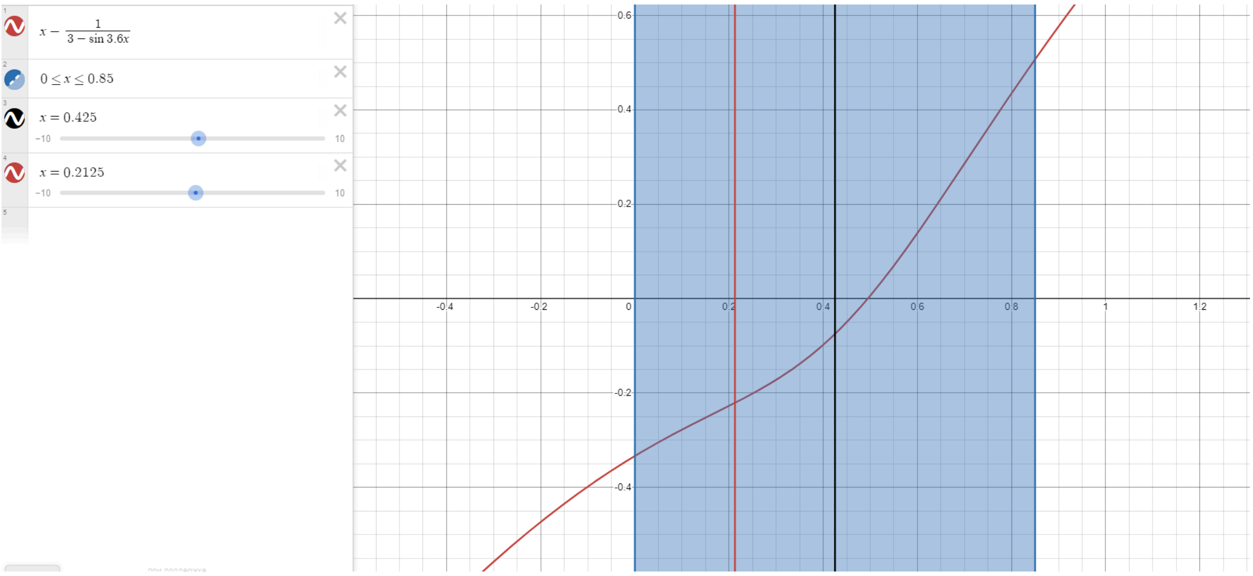

Рис. 6 - Геометрическая интерпретация метода половинчатого деления

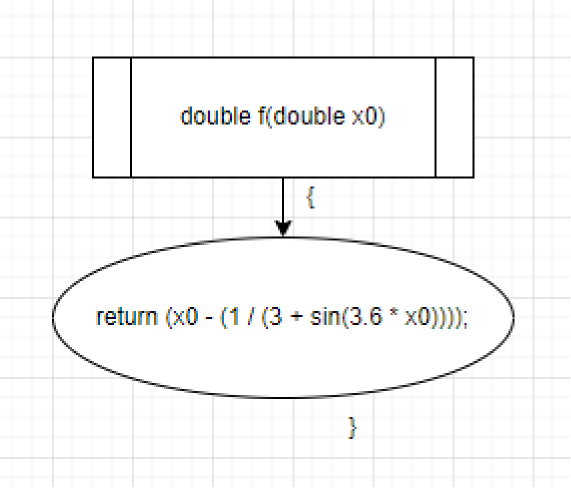

Рис. 7 - Блок-схема первой дополнительной функции.

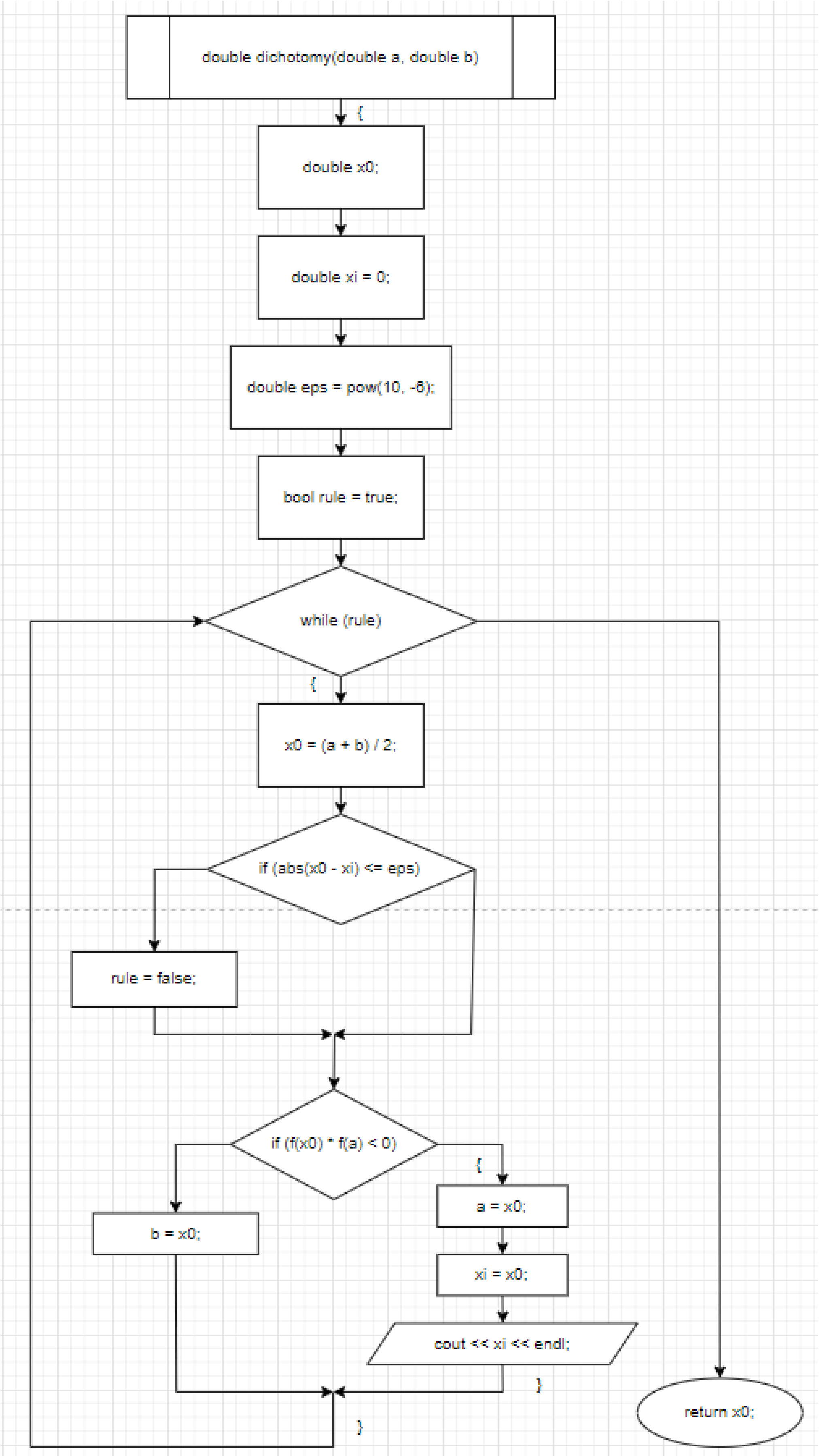

Рис. 8 - Блок-схема второй дополнительной функции.

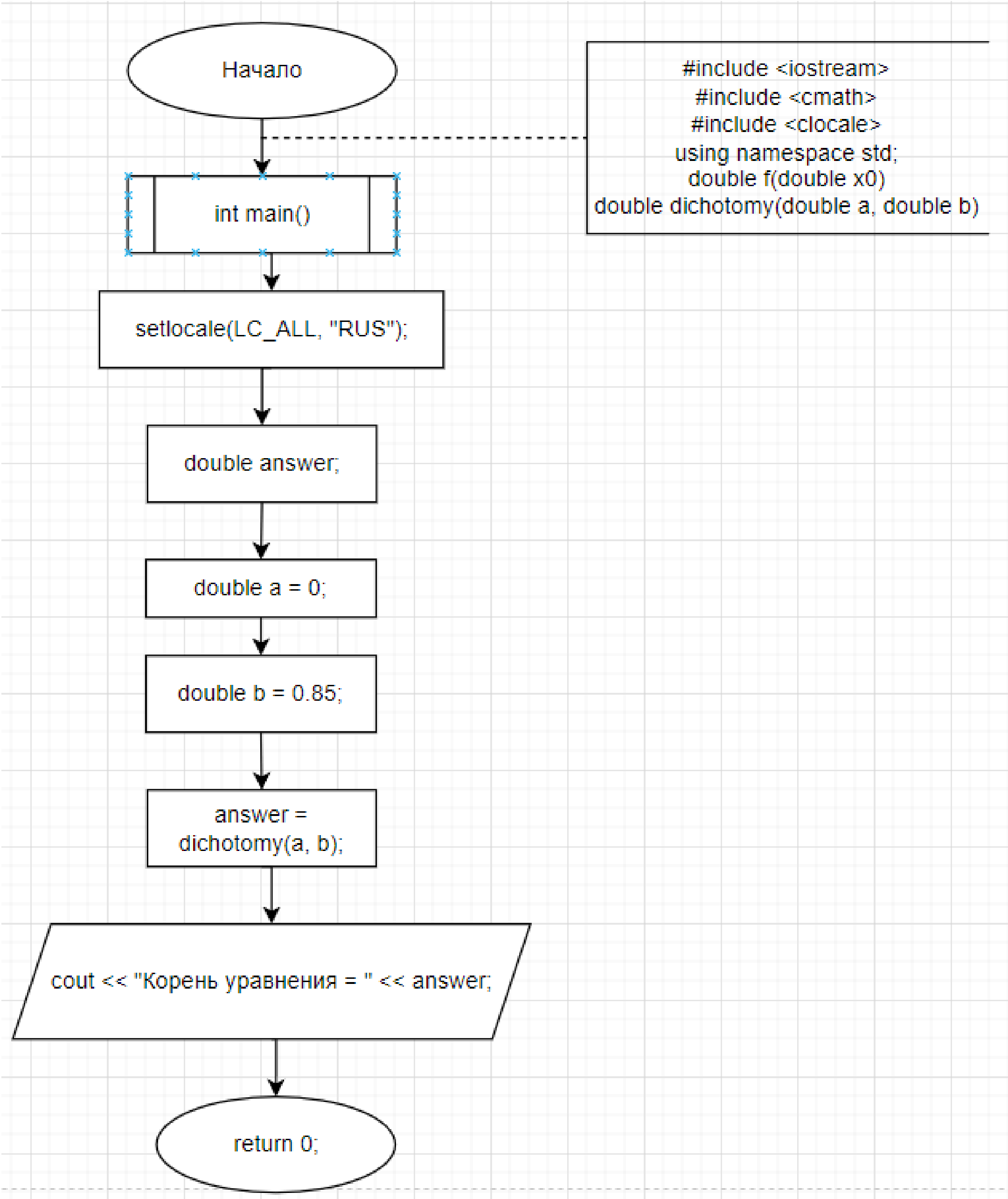

Рис. 9 - Блок-схема функции main.

```cpp
#include <iostream>
#include <cmath>
#include <clocale>
using namespace std;

double f(double x0) //Функция нелинейного уравнения
{
    return (x0 - (1 / (3 + sin(3.6 * x0))));
}
double dichotomy(double a, double b)
{
    double x0;
    double xi = 0;
    double eps = pow(10, -6); //Точность решения
    bool rule = true;
    while (rule)
    {
        x0 = (a + b) / 2;
        if (abs(x0 - xi) <= eps) //Проверка промежуточного значения корня
        {
            rule = false; //Завершаем цикл половинчатого деления
        }
        if (f(x0) * f(a) < 0) //Проверка на какой половине есть корень
        {
            b = x0;
        }
        else
        {
            a = x0;
            xi = x0;
            cout << "Промежуточное значение = " << xi << endl; //Вывод промежуточного значения
        }
    }
    return x0;
}
int main()
{
    setlocale(LC_ALL, "RUS");
    double answer;
    double a = 0;
    double b = 0.85;
    answer = dichotomy(a, b);
    cout << "Корень уравнения = " << answer;
    return 0;
}
```

Рис. 10 - Программная реализация метода половинчатого деления.

Консоль отладки Microsoft Visual Studio

```
Промежуточное значение = 0.2125
Промежуточное значение = 0.239063
Промежуточное значение = 0.252344
Промежуточное значение = 0.258984
Промежуточное значение = 0.262305
Промежуточное значение = 0.262408
Промежуточное значение = 0.262434
Промежуточное значение = 0.262441
Корень уравнения = 0.262442
```

Рис. 11 - Результат выполнения программы.

Найдено значение корня 0.2624.

Метод итераций.

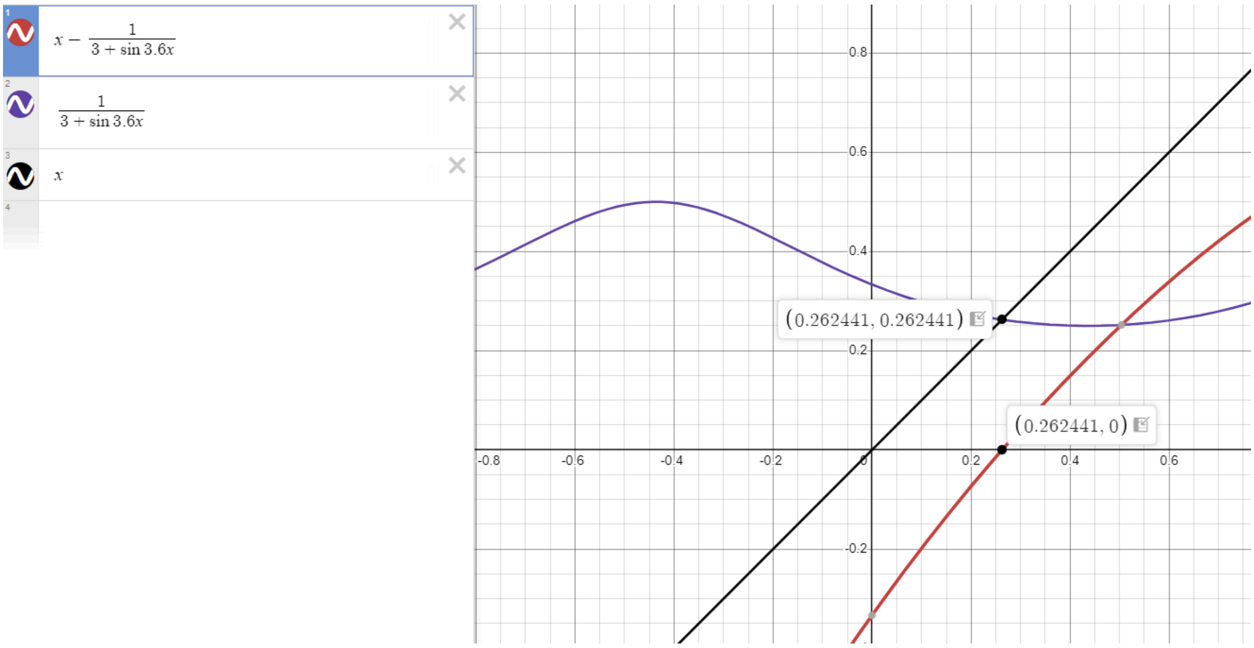

Рис. 12 - Графики метода итераций

Выражаем вспомогательную функцию x=φ(x), $x = \frac{1}{3+sin(3.6x)}$ (график фиолетового цвета на рисунке 12).

Находим производную от вспомогательной функции

$\varphi\,'x = \frac{-3.6 * cos(3.6x)}{(3 - sin(3.6x))^{2}}$ и проверим условие сходимости

|φ'(0)|<1. $\varphi'(0) = |\frac{-3.6 * cos(3.6 * 0)}{(3 - sin(3.6 * 0))^{2}}| = |\frac{-3.6}{9}| < 1$

Следовательно, эта вспомогательная функция подходит.

Примем за начальное значение x0 левую границу интервала 0. Следующее значение x1 =φ(0). Вычисляем следующие значения x по формуле xi=φ(xi-1) до тех пор, пока модуль разности двух соседних значений x не будет меньше eps.

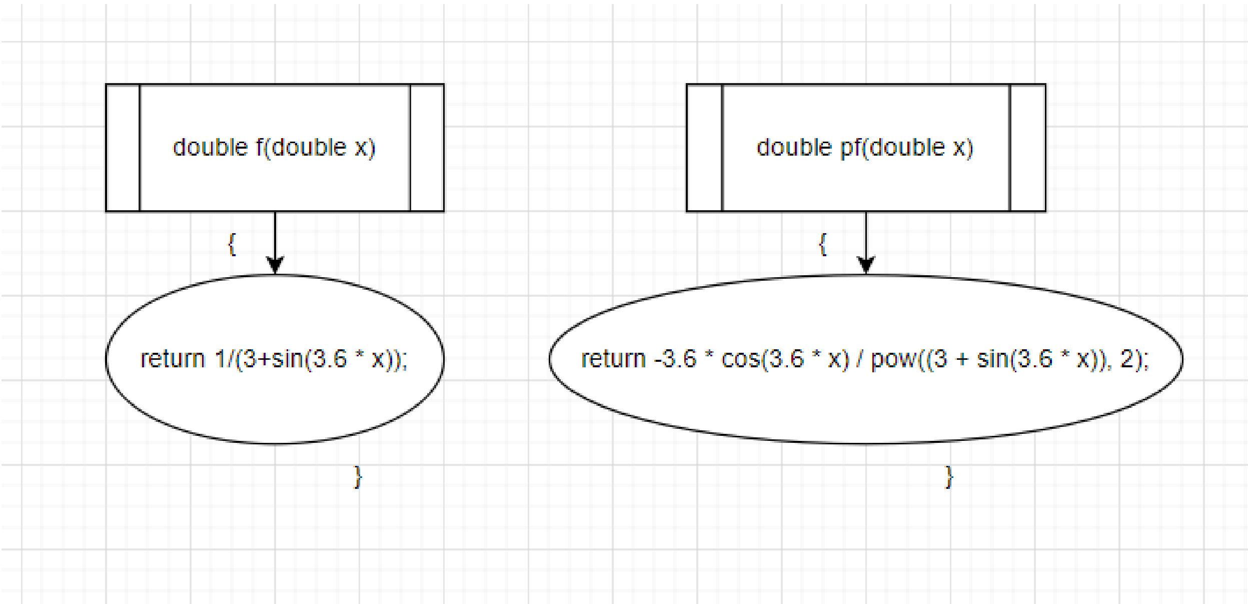

Рис. 13 - Блок-схемы дополнительных функций

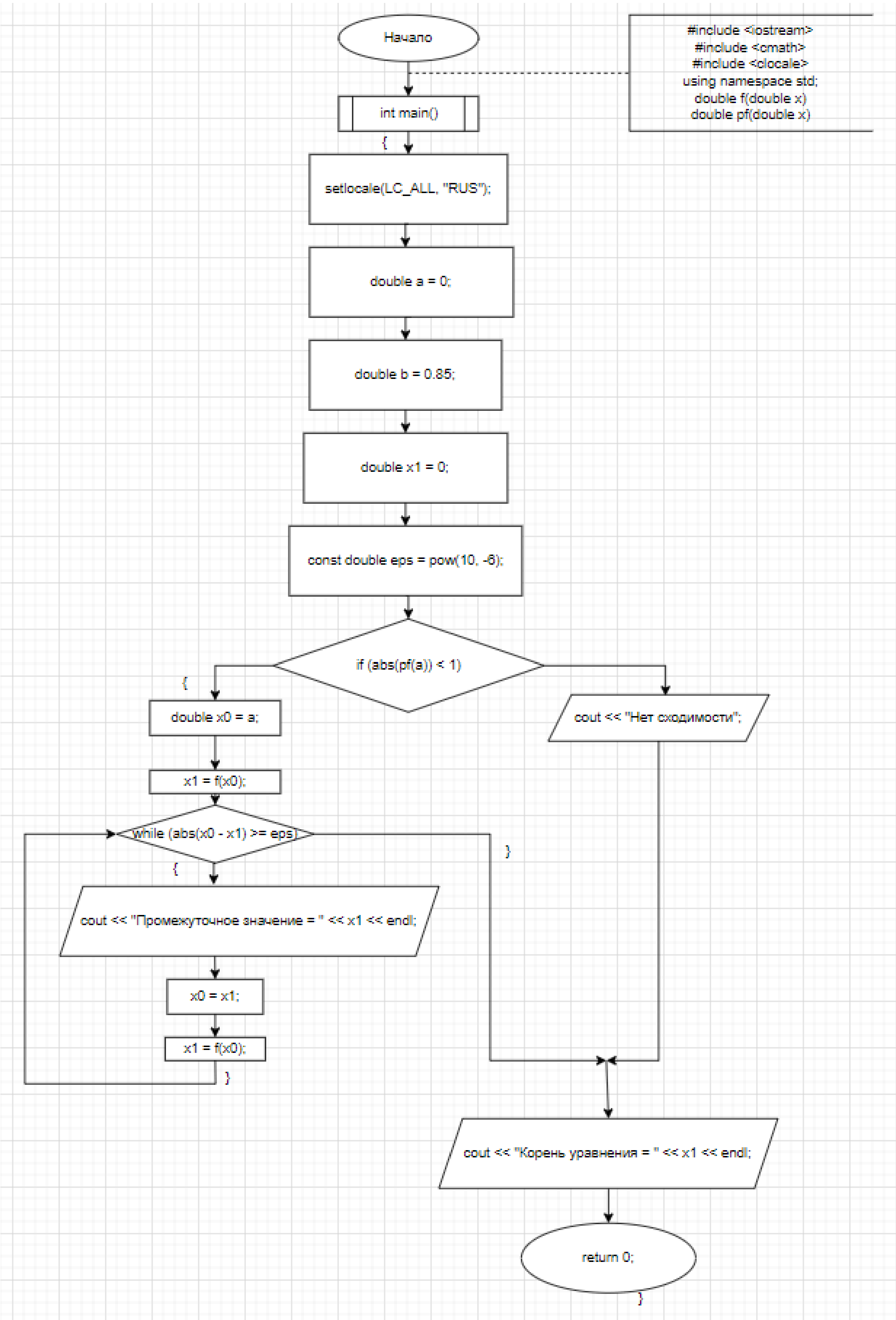

Рис. 14 - Блок-схема функции main

iteracii (Глобальная область)

```cpp
#include <iostream>
#include <cmath>
#include <clocale>
using namespace std;
double f(double x)
{
    return 1 / (3 + sin(3.6 * x)); //Вспомогательная функция фи(x)=x
}
double pf(double x)
{
    return -3.6 * cos(3.6 * x) / pow((3 + sin(3.6 * x)), 2); //Производная вспомогательной функции
}
int main()
{
    setlocale(LC_ALL, "RUS");
    double a = 0;
    double b = 0.85;
    double x1 = 0;
    const double eps = pow(10, -6);
    if (abs(pf(a)) < 1) //Проверка сходимости
    {
        double x0 = a;
        x1 = f(x0);
        while (abs(x0 - x1) >= eps) //Проверка на точность
        {
            cout << "Промежуточное значение = " << x1 << endl;
            x0 = x1;
            x1 = f(x0);
        }
    }
    else {
        cout << "Нет сходимости";
    }
    cout << "Корень уравнения = " << x1 << endl;
    return 0;
}
```

Рис. 15 - Программная реализация метода итераций

Консоль отладки Microsoft Visual Studio

```
Промежуточное значение = 0.333333
Промежуточное значение = 0.254321
Промежуточное значение = 0.26365
Промежуточное значение = 0.262266
Промежуточное значение = 0.262467
Промежуточное значение = 0.262438
Промежуточное значение = 0.262442
Корень уравнения = 0.262441
```

Рис. 16 - Результат выполнения программы

Ссылка на github: https://github.com/ArseniyUshakov/laboratories