**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №4

«**Нахождение корней нелинейного уравнения**»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил | |  | Принял | |
| ФИО: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | ФИО: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Группа: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Должность: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2022 г.

Постановка задачи

**1.** Найти корень уравнения ***x - cos(x) = 0*** простой итерацией, половинным делением и методом Ньютона с погрешностью eps<0.000001 и для каждого из трех методов определить количество шагов алгоритма.

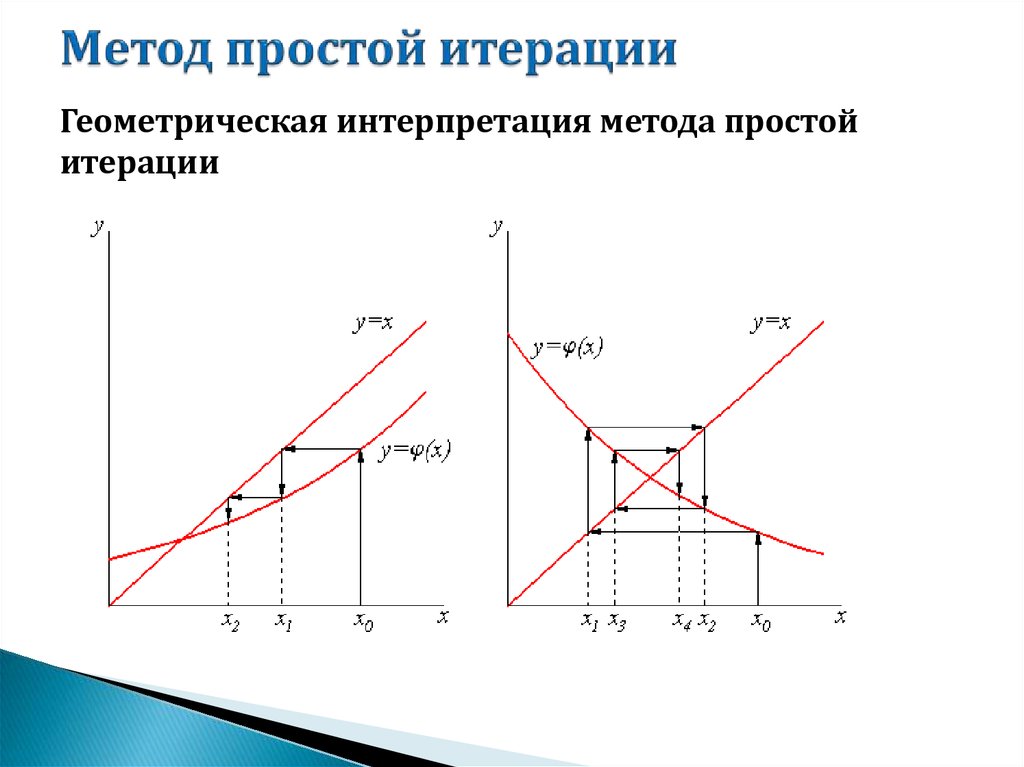
**2.** Выполнить п.1 для *eps < 0.00000001*.

**3.** Выполнить п.1 для уравнения ***x – 10cos(x) = 0*** и объяснить результаты.

Разработка алгоритма

Метод итерации

**Принцип работы:**

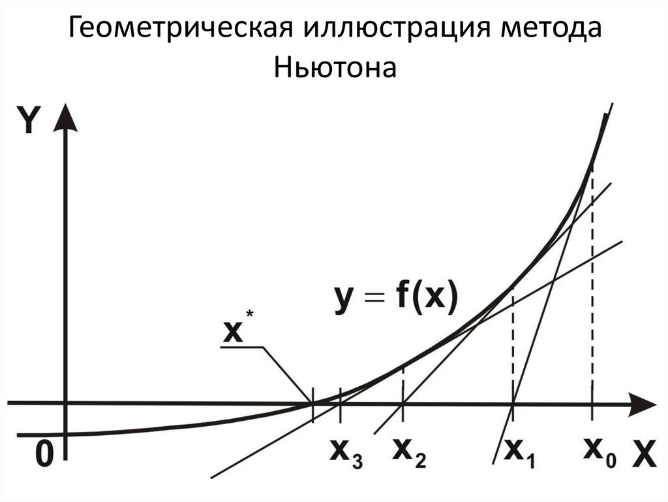
****Для метода справедливо, что Xj+1=Xj – f(Xj), так как, исходя из его геометрической интерпретации, таким образом мы будем бесконечно приближаться к корню нелинейного уравнения.

**Описание входных и выходных данных:**

* double x – инициализирующая координата для x
* double eps – погрешность
* double f(x) – функция
* size\_t i – количество итераций
* double c – коэффициент
* double f\_result – корень уровнения f(x)

Метод Ньютона

**Принцип работы:**

****Для данного метода справедливо, что Хj+1 = Xj - f(Xj)/ f ′(Xj), равенство получается при записи производной в качестве тангенса угла наклона графика: f ‘(Xj) = f(Xj)/(Xj-Xj+1).

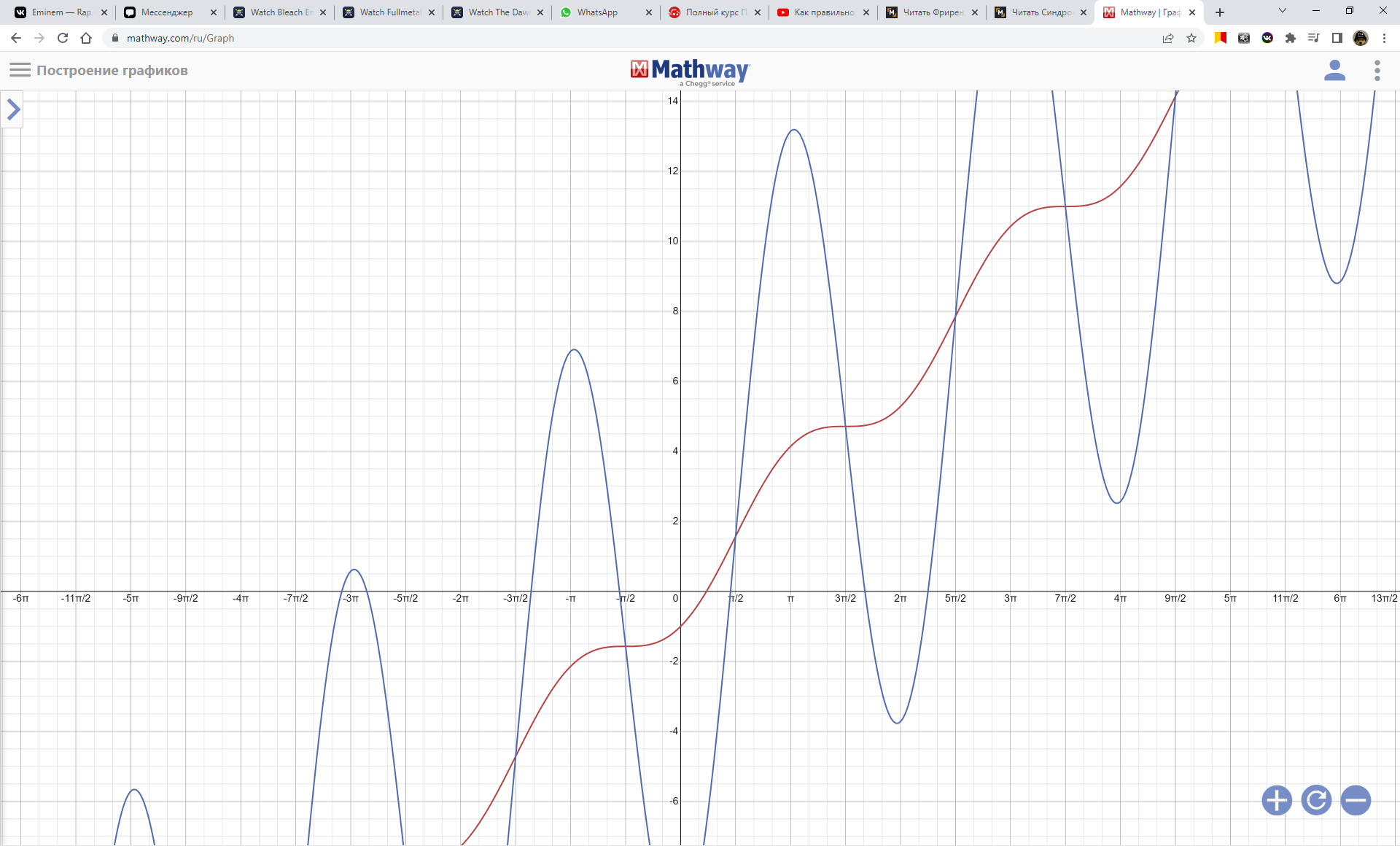
**Описание входных и выходных данных:**

* double x1 – инициализирующая координата для x
* double eps – погрешность
* double f(x) – функция
* double df(x) – производная функции
* size\_t i – количество итераций
* double c – коэффициент
* double f\_result – корень уровнения f(x)

Метод половинного деления

**Принцип работы:**

Пусть left – левая граница поиска, right – правая. Если f(left) и f(right) различаются по знаку, тогда между ними лежит корень, т.е. график функции пересекает ось x. За счет многократного нахождения середины mid отрезка между границами и сравнения знаков f(mid) с f(right) и f(left) и в соответствии с результатом присвоения значения mid правой или левой границе мы после множества итераций достигнем корня с погрешностью в eps.



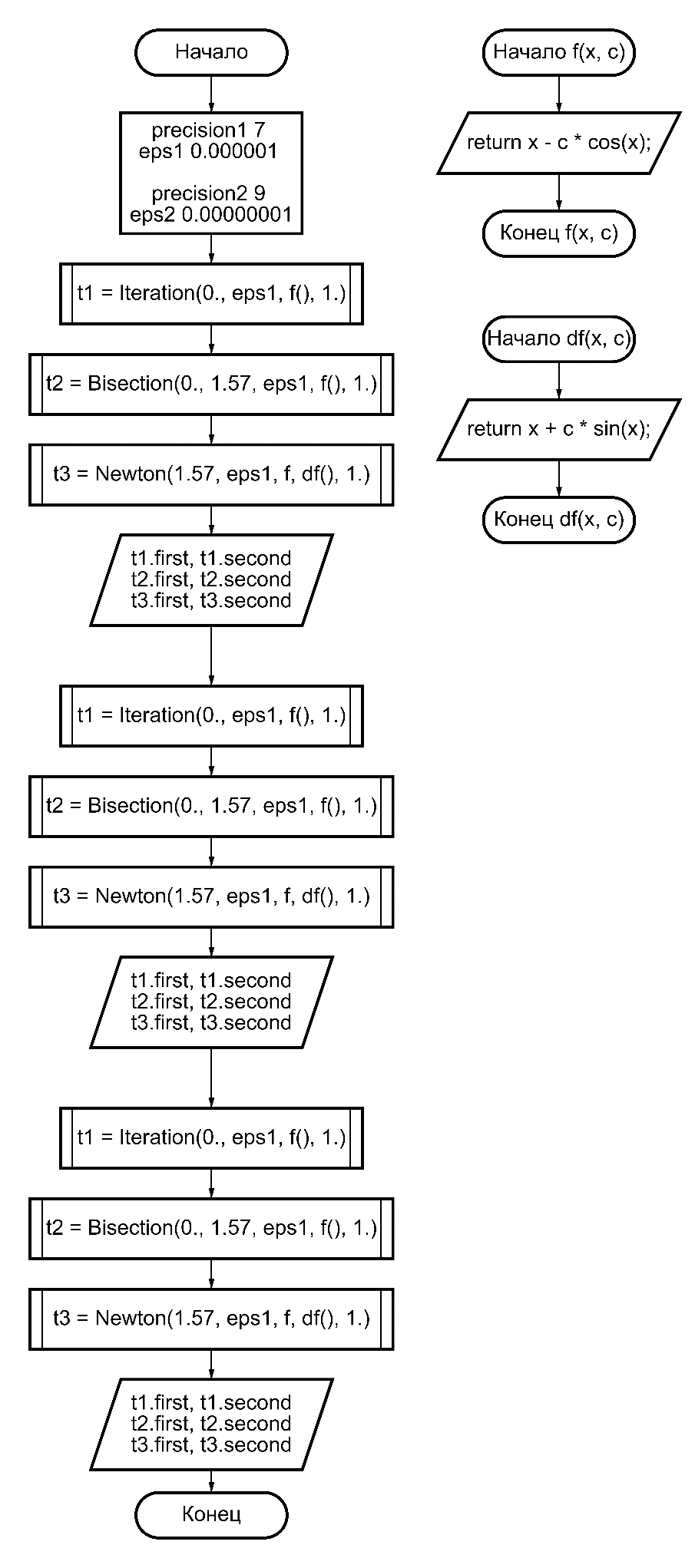
**Описание входных и выходных данных:**

* double x0 – левая граница\*
* double x1 – правая граница\*
* eps – погрешность
* double f(x) – функция
* int iter – количество итераций
* double x – корень уравнения f(x)

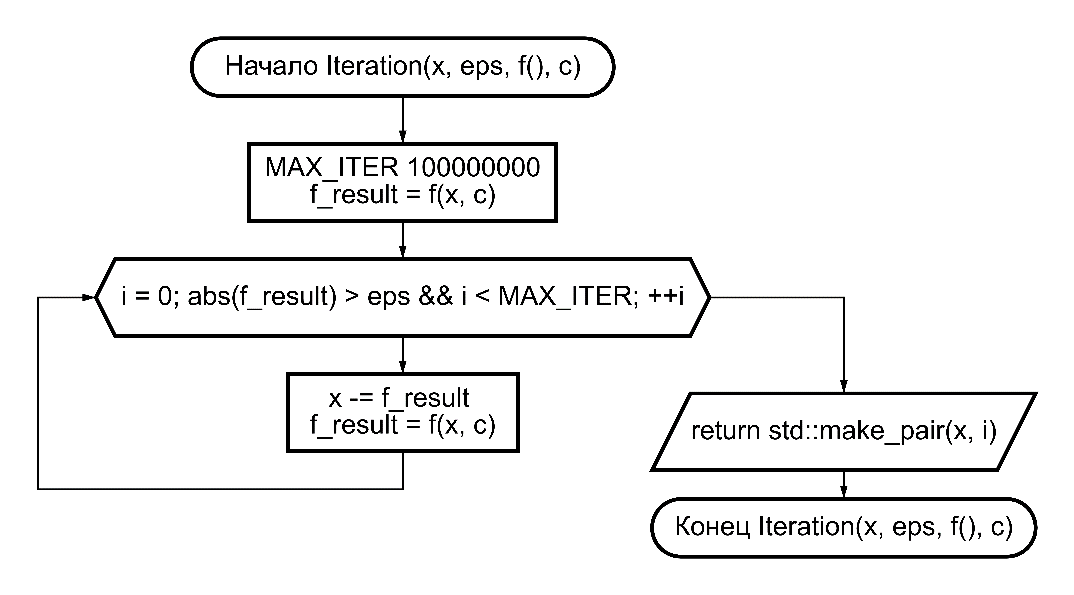
\*Возьмем за границы 0.0 и 1.57, так как из-за специфики функции, ее периодичности, один корень гарантированно будет лежать между 0 и pi/2 (графики п.1 и п.3)

**Схема алгоритма**

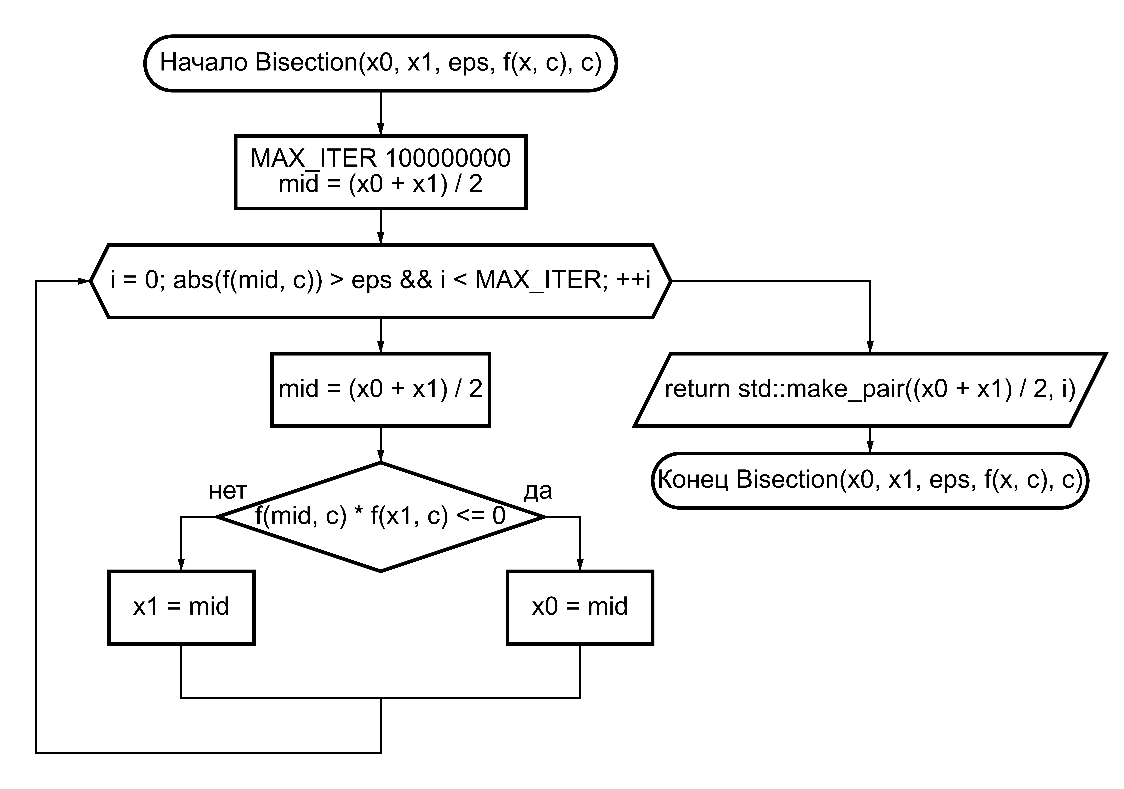
Main.cpp



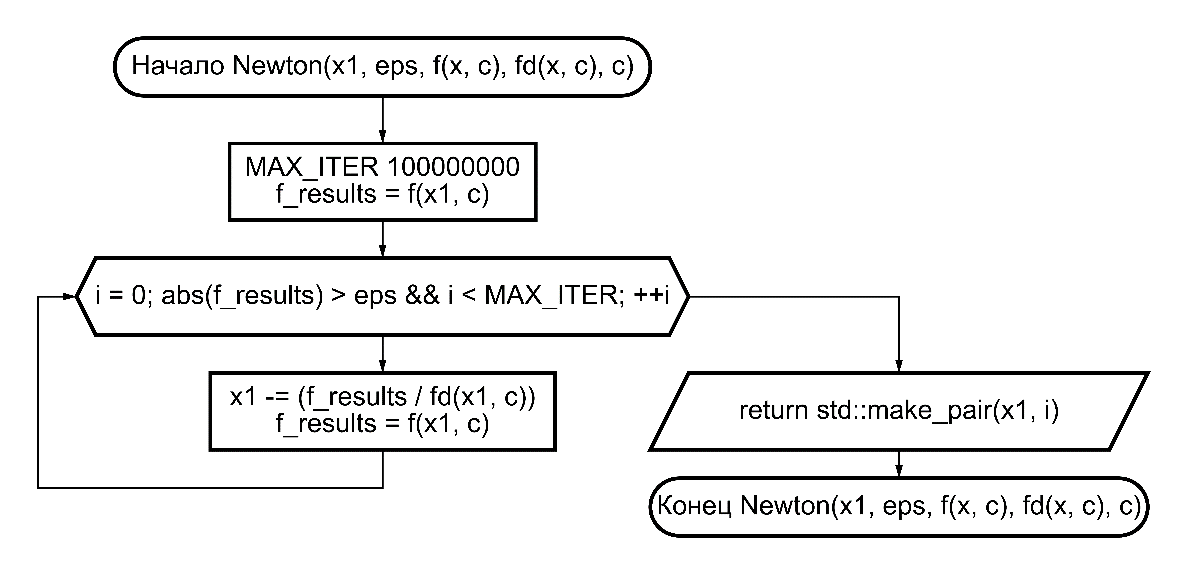
Iteration.cpp



Bisection.cpp



Newton.cpp



**Текст программы**

Main.cpp

#include "../include/lab04.hpp"

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

#define precision1 7

#define eps1 0.000001

#define precision2 9

#define eps2 0.00000001

double f(double x, double c) {

return x - c \* cos(x);

}

double df(double x, double c) {

return 1. + c \* sin(x);

}

int main() {

cout << "--- ь1 ---\n";

pair<double, size\_t> t1 = Iteration(0., eps1, f, 1.);

pair<double, size\_t> t2 = Bisection(0., 1.57, eps1, f, 1.);

pair<double, size\_t> t3 = Newton(1.57, eps1, f, df, 1.);

cout << fixed << setprecision(precision1) << "x = " << t1.first << ", iterations - " << t1.second << endl;

cout << setprecision(precision1) << "x = " << t2.first << ", iterations - " << t2.second << endl;

cout << setprecision(precision1) << "x = " << t3.first << ", iterations - " << t3.second << endl;

cout << "\n\n";

cout << "--- ь2 ---\n";

t1 = Iteration(0., eps2, f, 1.);

t2 = Bisection(0., 1.57, eps2, f, 1.);

t3 = Newton(1.57, eps2, f, df, 1.);

cout << fixed << setprecision(precision2) << "x = " << t1.first << ", iterations - " << t1.second << endl;

cout << setprecision(precision2) << "x = " << t2.first << ", iterations - " << t2.second << endl;

cout << setprecision(precision2) << "x = " << t3.first << ", iterations - " << t3.second << endl;

cout << "\n\n";

cout << "--- ь3 ---\n";

t1 = Iteration(0., eps1, f, 10.);

t2 = Bisection(0., 1.57, eps1, f, 10.);

t3 = Newton(1.57, eps1, f, df, 10.);

cout << fixed << setprecision(precision1) << "x = " << t1.first << ", iterations - " << t1.second << endl;

cout << setprecision(precision1) << "x = " << t2.first << ", iterations - " << t2.second << endl;

cout << setprecision(precision1) << "x = " << t3.first << ", iterations - " << t3.second << endl;

return 0;

}

Iteration.cpp

#include <cmath>

using namespace std;

#define MAX\_ITER 100000000

pair<double, std::size\_t> Iteration(double x, double eps, double(f(double x, double c)), double c) {

double f\_result = f(x, c);

size\_t i;

for (i = 0; abs(f\_result) > eps && i < MAX\_ITER; ++i) {

x -= f\_result;

f\_result = f(x, c);

}

return std::make\_pair(x, i);

}

Bisection.cpp

#include <iostream>

using namespace std;

#define MAX\_ITER 100000000

pair<double, std::size\_t> Bisection(double x0, double x1, double eps, double(f(double x, double c)), double c) {

double mid = (x0 + x1) / 2;

size\_t i;

for (i = 0; abs(f(mid, c)) > eps && i < MAX\_ITER; ++i) {

mid = (x0 + x1) / 2;

if (f(mid, c) \* f(x1, c) <= 0) { x0 = mid; }

else { x1 = mid; }

}

return std::make\_pair((x0 + x1) / 2, i);

}

Newton.cpp

#include <iostream>

using namespace std;

#define MAX\_ITER 100000000

pair<double, std::size\_t>

Newton(double x1, double eps, double(f(double x, double c)), double(fd(double x, double c)), double c) {

double f\_results = f(x1, c);

size\_t i;

for (i = 0; abs(f\_results) > eps && i < MAX\_ITER; ++i) {

x1 -= (f\_results / fd(x1, c));

f\_results = f(x1, c);

}

return std::make\_pair(x1, i);

}

Lab04.hpp

#ifndef INF\_LAB\_4\_LAB04\_H

#define INF\_LAB\_4\_LAB04\_H

#include <iostream>

//Iteration

std::pair<double, size\_t> Iteration(double x, double eps, double(f(double x, double c)), double c);

//Bisection

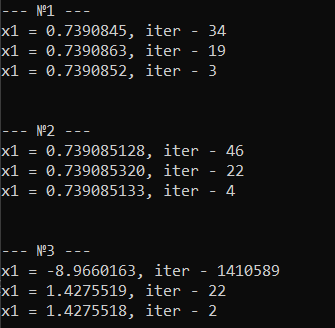
std::pair<double, std::size\_t> Bisection(double x0, double x1, double eps, double(f(double x, double c)), double c);

//Newton

std::pair<double, std::size\_t>

Newton(double x, double eps, double(f(double x, double c)), double(fd(double x, double c)), double c);

#endif //INF\_LAB\_4\_LAB04\_H

**Анализ результатов**

\*В задаче 3 имеем отличный от остального результат в методе простых итераций, так как из-за огромной амплитуды функции f(x)=10cos(x) шаг каждой итерации получается очень большим. Алгоритм приходит к оптимальному шагу только через некоторое количество итераций, причем попадает в наименьший корень нелинейного уравнения.

**Вывод**

Я научился использовать функции, изучил математические методы нахождения корней нелинейных уравнений, научился использовать внешнее связывание единиц трансляции.