Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторної роботи №1

На тему:
“Розв'язування нелінійних рівнянь методом дихотомії та методом хорд”

Лектор:
доц. каф. ПЗ
Мельник Н.Б.
Виконав:
ст. гр. ПЗ-18
Юшкевич А.І.
Прийняв:
проф. каф. ПЗ
Гавриш В.І.
« ___ » ___ 2023 р.

∑ = _____

Львів – 2023

Тема: розв'язування нелінійних рівнянь за методом дихотомії та за методом хорд.

Мета: ознайомлення на практиці з методами відокремлення дійсних ізольованих коренів нелінійних рівнянь. Вивчення методів уточнення коренів - методу дихотомії та методу хорд.

Завдання

Методами дихотомії та хорд визначити корінь рівняння з точністю 10^{-3}.
Код до програми див. у розділі Код реалізації розв'язання нелінійного рівняння подано у додатку.

Рівняння: $\arctan(x) - \sin(x) = 0$

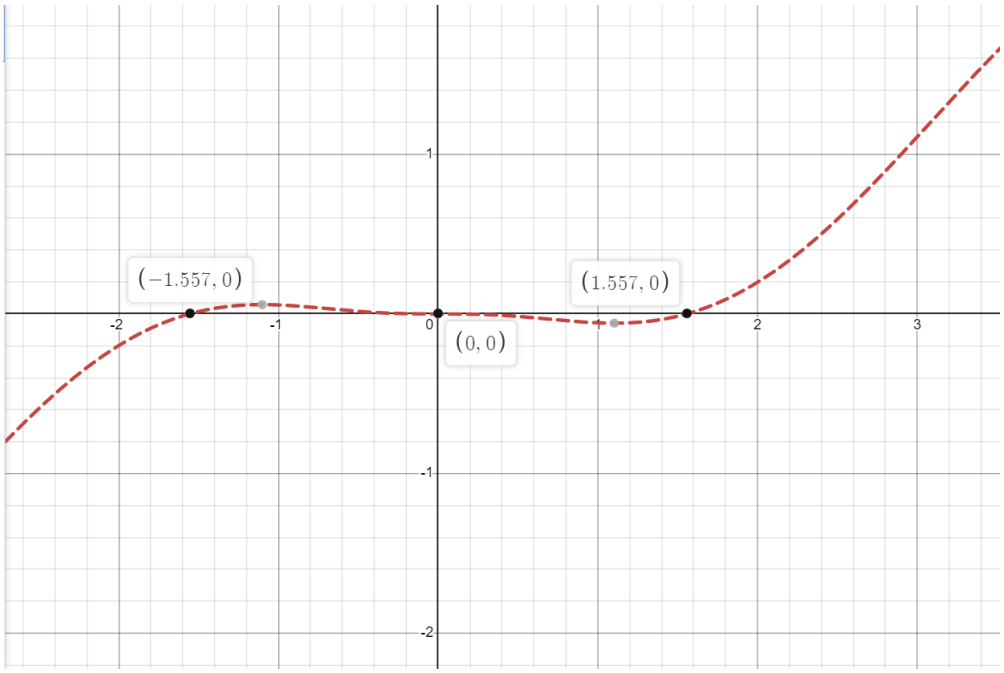

Рис. 1. Графік функції $f(x) = \arctan(x) - \sin(x)$

Метод дихотомії

Наведемо основні етапи методу дихотомії:

1) локалізуємо дійсний корінь на відрізку;
2) якщо на даному відрізку дійсний корінь існує, то визначаємо середину відрізка, а саме точку c;
3) перевіряємо знаки значень функції в точках a та c і c та b. Якщо f(a) * f(c) > 0 - це означає, що відрізок [a;c] не містить кореня рівняння, а отже він знаходиться на відрізку [c;b], тому повинно виконуватися наступне f(c) * f(b) < 0;
4) вибираємо відрізок [c;b], який містить корінь;
5) виконуємо послідовно кроки, наведені в пунктах 2, 3, 4 до тих пір, поки не отримаємо задану точність.

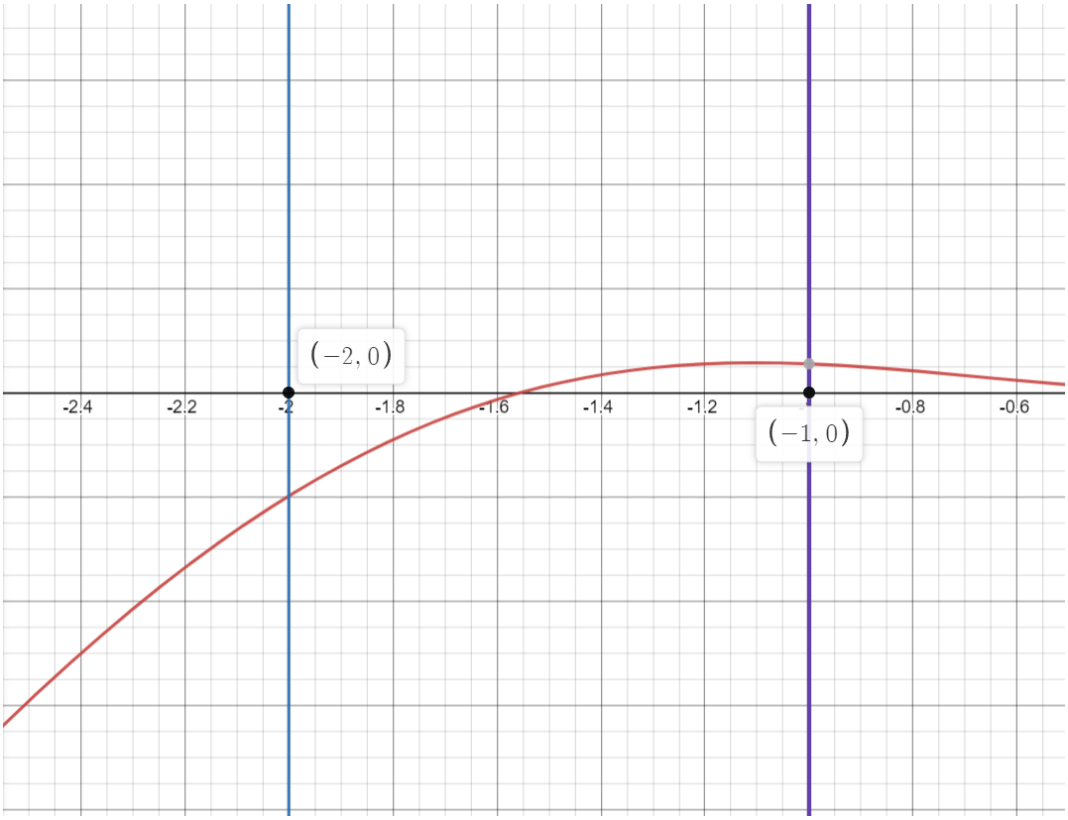

Рис. 2. Метод дихотомії. Локалізовані корені.

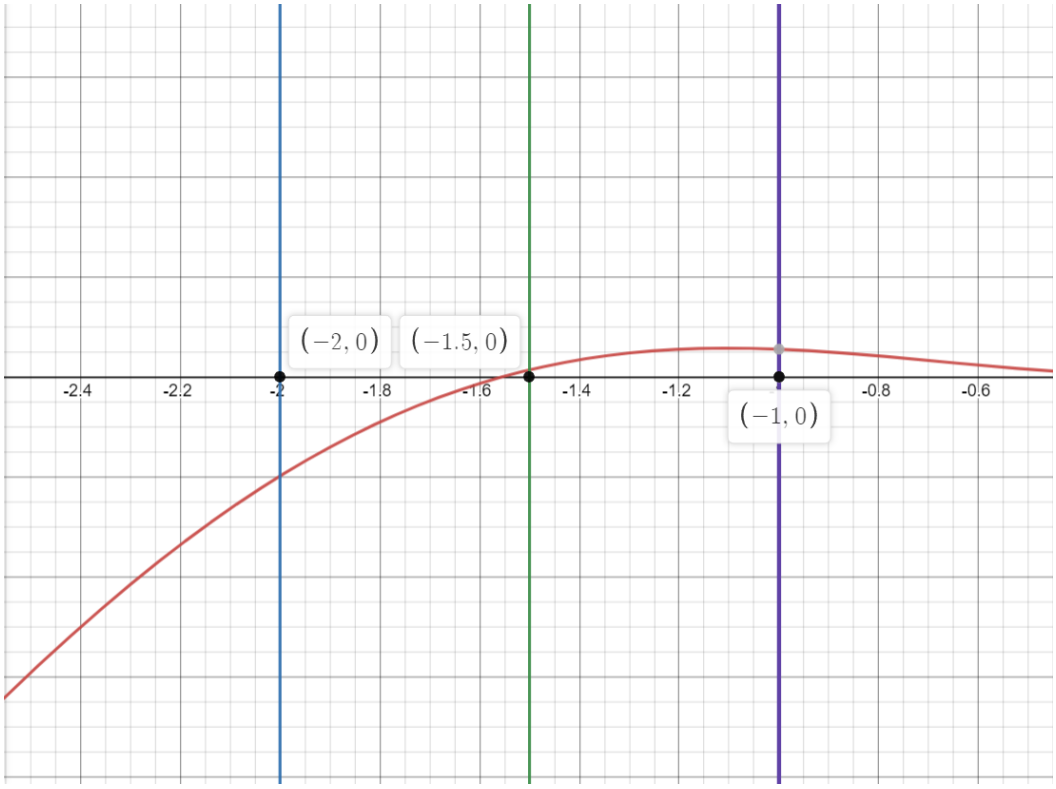

Рис. 3. Метод дихотомії. Перша ітерація.

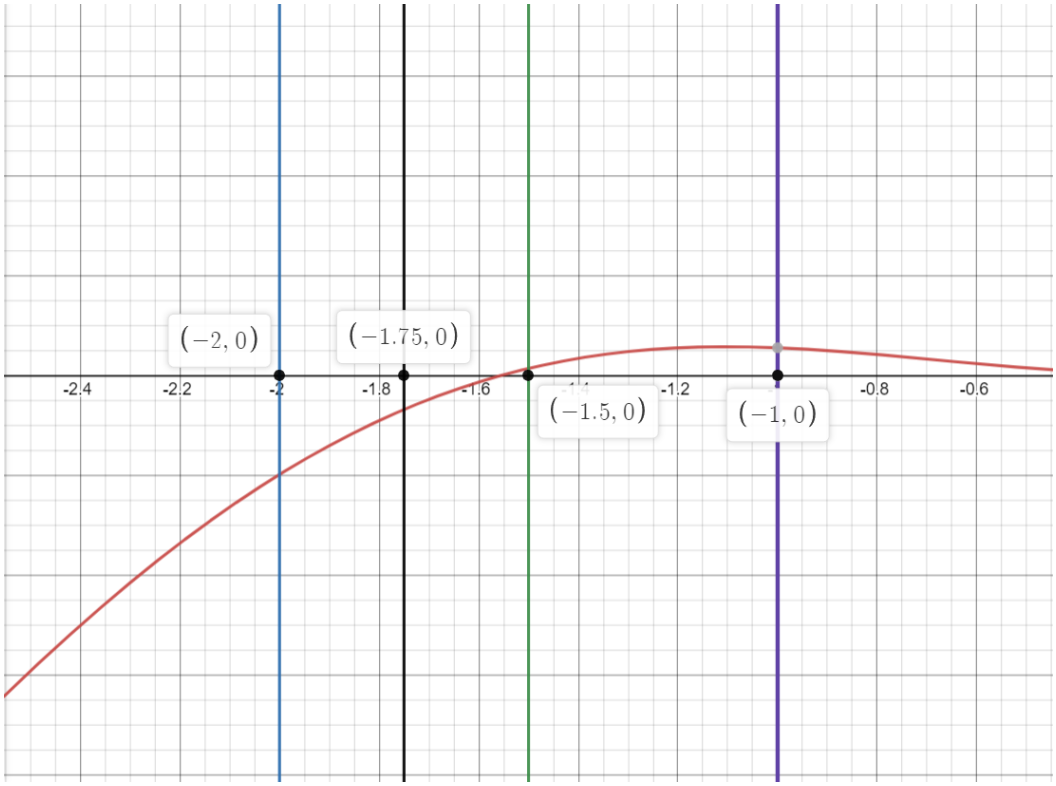

Рис. 5. Метод дихотомії. Друга ітерація.

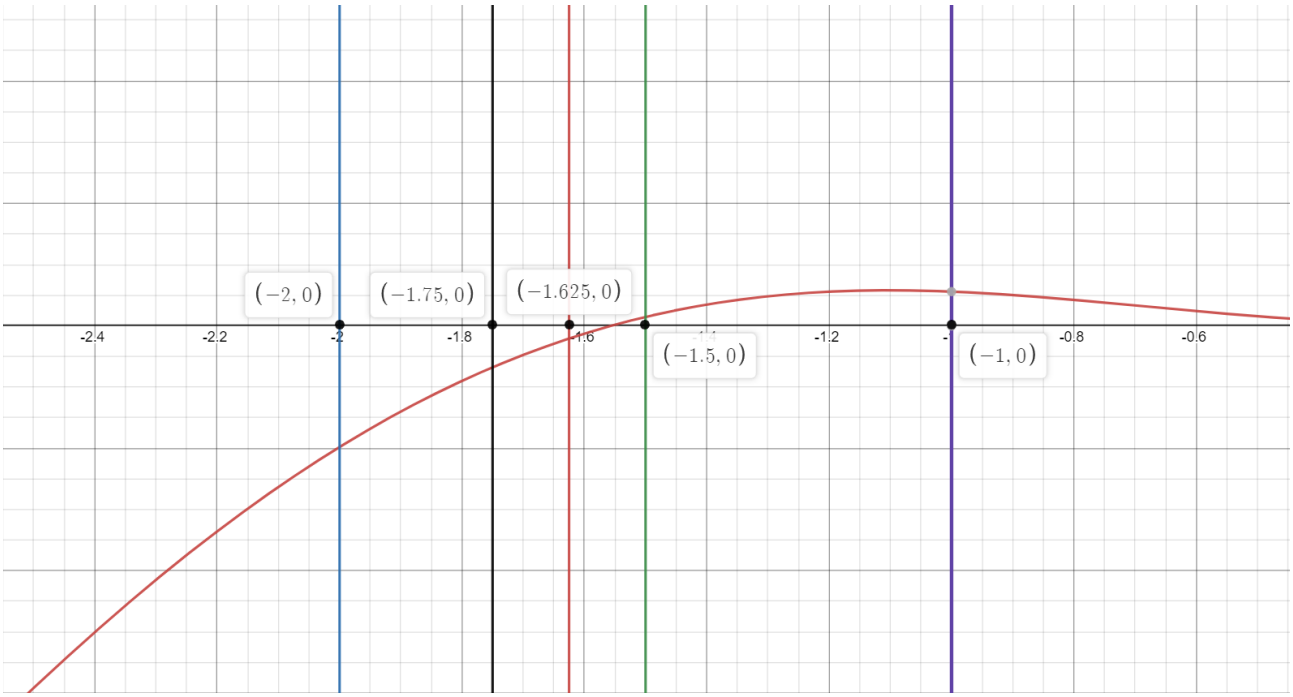

Рис. 6. Метод дихотомії. Третя ітерація.

Основні етапи обчислювального алгоритму, який був реалізований у програмному продукті на мові C++ :

1) введення даних користувачем для ініціалізації змінних(leftLim, rightLim, eps), що відповідають за початковий проміжок знаходження кореня;

2) створюємо екземпляр класу Dichotomy – findDich та параметризуємо його отриманими в попередньому пункті значеннями.
3) викликаємо на об ‘єкті findDich функцію Find() (Рис.7.);
4) в функції Find() викликається функція SetLimits()(Рис.8.), що перевіряє вхідні значення на правильність (leftLim < rightLim, міняє їх місцями, якщо leftLim > rightLim) та локалізує корінь на заданому проміжку.
5) якщо корінь локалізовано повертаємося у функцію Find(), яка описує алгоритм методу дихотомії(див. вище). Функція повертає структуру, що містить кількість ітерацій та результат.
6) виводимо результат виконання програми на екран (Рис.9.).

```
SResult Find() {
    x = FindSecant(a, b);

    bool resultIsFound = false;

    resultIsFound = SetLimits(a, b, eps);
    if (resultIsFound)
        x = a;

    ldouble staticPoint = 0;
    ldouble xPrev = x;
    while (!resultIsFound) {
        result.iterations++;

        if (f(x) * f(a) <= 0)
            staticPoint = a;
        else
            staticPoint = b;

        xPrev = x;
        x = FindSecant(x, staticPoint);

        if (fabs(x - xPrev) <= eps)
            resultIsFound = true;
    }
    result.result = x;

    return result;
}
```

Рис. 7. Метод Find об’єкту класу Secant

```cpp
bool SetLimits(ldouble& leftLim, ldouble& rightLim, ldouble eps) {
    bool onGoing = true, resultIsFound = false;
    ldouble step = (fabs(leftLim) + fabs(rightLim)) / 10, fX = 0, fLeft = 0;

    if (leftLim > rightLim)
        Swap(leftLim, rightLim);

    if (step >= 1)
        step = 1;

    fLeft = f(leftLim);
    if (fLeft == 0) {
        resultIsFound = true;
        rightLim = fLeft;
        onGoing = false;
    }

    while (onGoing) {

        for (ldouble x = leftLim + step; x < rightLim; x += step) {
            fX = f(x);

            if (fabs(fX) <= eps) {
                leftLim = x;
                rightLim = x;
                resultIsFound = true;
                onGoing = false;
                break;
            }

            if (fLeft * fX <= 0) {
                rightLim = x;
                leftLim = x -= step;
                onGoing = false;
                break;
            }

            fLeft = fX;
        }
        step /= 10;
    }

    return resultIsFound;
}
```

Рис. 8. Функція SetLimits()

```
Microsoft Visual Studio Debu
Enter left lim: 54
Enter right lim: -35
Enter epsilon: 0.001


RESULT

Dichotomy:
Iterations: 10
Result: -1.55713


C:\Disk D\Fall\Numerical_Methods\Lab_01\Lab_01_DM\x64\Debug\Lab_01_NM.exe (process 24264) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Рис. 9. Протокол виконання програми

Метод хорд

Перед тим, як розпочати роботу розглянемо алгоритм методу хорд:

1) локалізуємо дійсний корінь на відрізку;
2) якщо відрізок з коренем існує, то складаємо рівняння прямої через дві точки, а саме через кінці відрізка на якому знаходиться корінь;
3) знаходимо перетин цієї прямої з віссю абсцис, G = a + $\frac{(y-f(a))*(a-b)}{f(a)-f(b)}$, де G - це x координата точки перетину з віссю абсцис, a - лівий край відрізку, b - правий край відрізку;
4) вибираємо точку через яку проводити наступну пряму, для цього перевіряємо знаки значень функції в точках a та c і c та b, де c це точка перетину прямої через кінці з віссю абсцис. Якщо f(a) * f(c) > 0 - це означає, що корінь рівняння не знаходиться на відрізку [a;c] і відповідно буде знаходитися на відрізку [c;b], тому виконується наступне f(c) * f(b) < 0;
5) проводимо пряму через точки, визначені в кроці 4;
6) виконуємо послідовні кроки, наведені в пунктах 3, 4 та 5 доти, поки отримаємо задану точність.

Основні етапи обчислювального алгоритму, який був реалізований у програмному продукті на мові C++ :

1) введення даних користувачем для ініціалізації змінних(leftLim, rightLim, eps), що відповідають за початковий проміжок знаходження кореня;
2) створюємо екземпляр класу Secant – findSec та параметризуємо його отриманими в попередньому пункті значеннями.
3) викликаємо на об ‘єкті findSec функцію Find() (Рис.10.);
4) в функції Find() викликається функція SetLimits()(Рис.7.), що перевіряє вхідні значення на правильність (leftLim < rightLim, міняє їх місцями, якщо leftLim > rightLim) та локалізує корінь на заданому проміжку.
5) якщо корінь локалізовано повертаємося у функцію Find(), яка описує алгоритм методу дихотомії(див. вище). Функція повертає структуру, що містить кількість ітерацій та результат.
6) виводимо результат виконання програми на екран (Рис.11.).

```cpp
SResult Find()
{
    ldouble midLim = 0;
    bool BordersAreSetting = true, resultIsFound = false;

    resultIsFound = SetLimits(leftLim, rightLim, eps);

    if (resultIsFound)
        midLim = leftLim;
    else
        midLim = (leftLim + rightLim) / 2;
    while (!resultIsFound) {
        result.iterations++;

        if (f(leftLim) * f(midLim) <= 0) {
            rightLim = midLim;
        }
        else {
            leftLim = midLim;
        }

        midLim = (leftLim + rightLim) / 2;
        if (fabs(leftLim - rightLim) < eps)
            resultIsFound = true;

    }
    result.result = midLim;

    return result;
}
```

Рис. 10. Метод Find об’єкту класу Secant

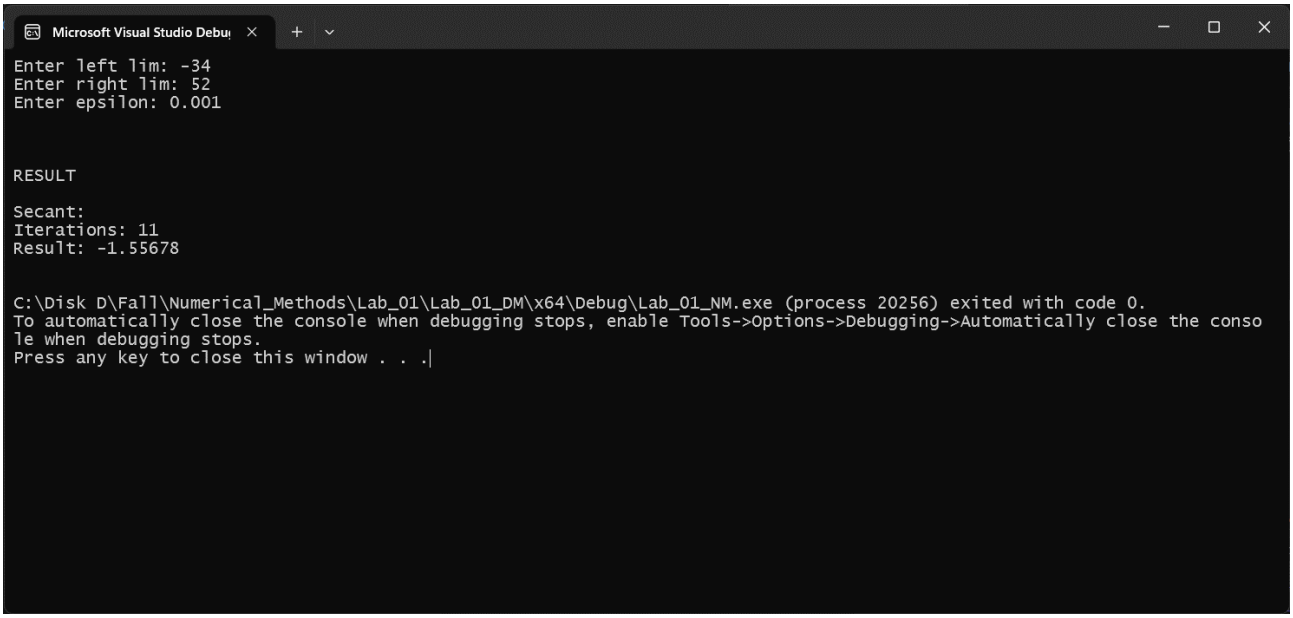

Рис. 11. Протокол виконання програми

Висновки

У результаті виконання лабораторної роботи, визначено дійсний корінь нелінійного алгебраїчного рівняння $arctan(x) - sin(x) = 0$ з точністю 10^{-3}методом дихотомії та методом хорд. Задана точність отримана методом дихотомії за 10 ітерацій, а за методом хорд за 11 ітерації.

Додаток

Header.h:

```
#pragma once

#include <iostream>
#include <cmath>

typedef long double ldouble;

using namespace std;

typedef struct {
	int iterations;
	ldouble result;
} SResult;

ldouble f(ldouble x) {
   return atan(x) - sin(x);
}

void Swap(ldouble& left, ldouble& right) {
   ldouble temp = left;
   left = right;
   right = temp;
}

bool SetLimits(ldouble& leftLim, ldouble& rightLim, ldouble eps) {
   bool onGoing = true, resultIsFound = false;
   ldouble step = (fabs(leftLim) + fabs(rightLim)) / 10, fX = 0, fLeft = 0;

   if (leftLim > rightLim)
      Swap(leftLim, rightLim);

   if (step >= 1)
      step = 1;
```

```
    fLeft = f(leftLim);
    if (fLeft == 0) {
        resultIsFound = true;
        rightLim = fLeft;
        onGoing = false;
    }

    while (onGoing) {

        for (ldouble x = leftLim + step; x < rightLim; x += step) {
            fX = f(x);

            if (fabs(fX) <= eps) {
                leftLim = x;
                rightLim = x;
                resultIsFound = true;
                onGoing = false;
                break;
            }

            if (fLeft * fX <= 0) {
                rightLim = x;
                leftLim = x -= step;
                onGoing = false;
                break;
            }

            fLeft = fX;
        }
        step /= 10;
    }

    return resultIsFound;
}

class Secant {

private:
        ldouble leftLim, rightLim, x, eps;
        SResult result;

        ldouble FindSecant(ldouble a, ldouble b) {

                return a - (f(a) * (b - a)) / (f(b) - f(a));
        }
```

```
public:
        Secant(ldouble a, ldouble b, ldouble eps) {
      this->leftLim = a;
      this->rightLim = b;
      this->eps = eps;
      result.iterations = 0;
      result.result = NAN;
      x = 0;

        }

        SResult Find() {
              x = FindSecant(leftLim, rightLim);

      bool resultIsFound = false;

      resultIsFound = SetLimits(leftLim, rightLim, eps);
      if (resultIsFound)
         x = leftLim;

      ldouble staticPoint = 0;
              ldouble xPrev = x;
      while (!resultIsFound) {
         result.iterations++;

         if (f(x) * f(leftLim) <= 0)
            staticPoint = leftLim;
         else
            staticPoint = rightLim;

         xPrev = x;
         x = FindSecant(x, staticPoint);

         if (fabs(x - xPrev) <= eps)
            resultIsFound = true;
      }
              result.result = x;

              return result;
        }

};

class Dichotomy {
private:
   ldouble leftLim, rightLim, eps;
   SResult result;
```

```
public:
    Dichotomy(ldouble leftLim, ldouble rightLim, ldouble eps) {
        this->leftLim = leftLim;
        this->rightLim = rightLim;
        this->eps = eps;
        result.iterations = 0;
        result.result = NAN;
    }

    SResult Find()
    {
        ldouble midLim = 0;
        bool BordersAreSetting = true, resultIsFound = false;

        resultIsFound = SetLimits(leftLim, rightLim, eps);

        if (resultIsFound)
            midLim = leftLim;
        else
            midLim = (leftLim + rightLim) / 2;
        while (!resultIsFound) {
            result.iterations++;

            if (f(leftLim) * f(midLim) <= 0) {
                rightLim = midLim;
            }
            else {
                leftLim = midLim;
            }

            midLim = (leftLim + rightLim) / 2;
            if (fabs(leftLim - rightLim) < eps)
                resultIsFound = true;

        }
        result.result = midLim;

        return result;
    }
};
```

Main.cpp:

```
#include "Header.h"
```

```
int main() {
	ldouble left, right, eps;

	cout << "Enter left lim: ";
	cin >> left;
	cout << "Enter right lim: ";
	cin >> right;
	cout << "Enter epsilon: ";
	cin >> eps;
	cout << endl << endl << endl << "RESULT" << endl << endl;

	Secant findSec(left, right, eps);
	Dichotomy findDich(left, right, eps);

	SResult dichRes = findDich.Find();
	SResult secRes = findSec.Find();

	cout << "Dichotomy: " << endl << "Iterations: " << dichRes.iterations << endl
<< "Result: " << dichRes.result << endl << endl;
	cout << "Secant: " << endl << "Iterations: " << secRes.iterations << endl <<
"Result: " << secRes.result << endl << endl;

	return 0;
}
```