**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

**Інститут комп’ютерних наук та інформаційних технологій**

**Кафедра програмного забезпечення**



**Звіт**

Про виконання лабораторної роботи №2

**На тему:**

“ознайомлення на практиці з методом дотичних та методом послідовних наближень для розв’язування нелінійних рівнянь”

**Лектор:**

доц. каф. ПЗ

Мельник Н.Б.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

асис. каф. ПЗ

Гавриш В.І.

« \_\_\_ » \_\_\_ 2022 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема**: розв’язування нелінійних рівнянь за методом дотичних та за методом послідовних наближень.

**Мета**: ознайомлення на практиці з методами відокремлення дійсних ізольованих коренів нелінійних рівнянь. Вивчення методів уточнення коренів - методу дотичних та методу послідовних наближень.

# Завдання

Методами дотичних та простих ітерацій визначити корінь рівняння з точністю . Код до програми див. у розділі Код реалізації розв’язання нелінійного рівняння подано у додатку.

Варіант 22:

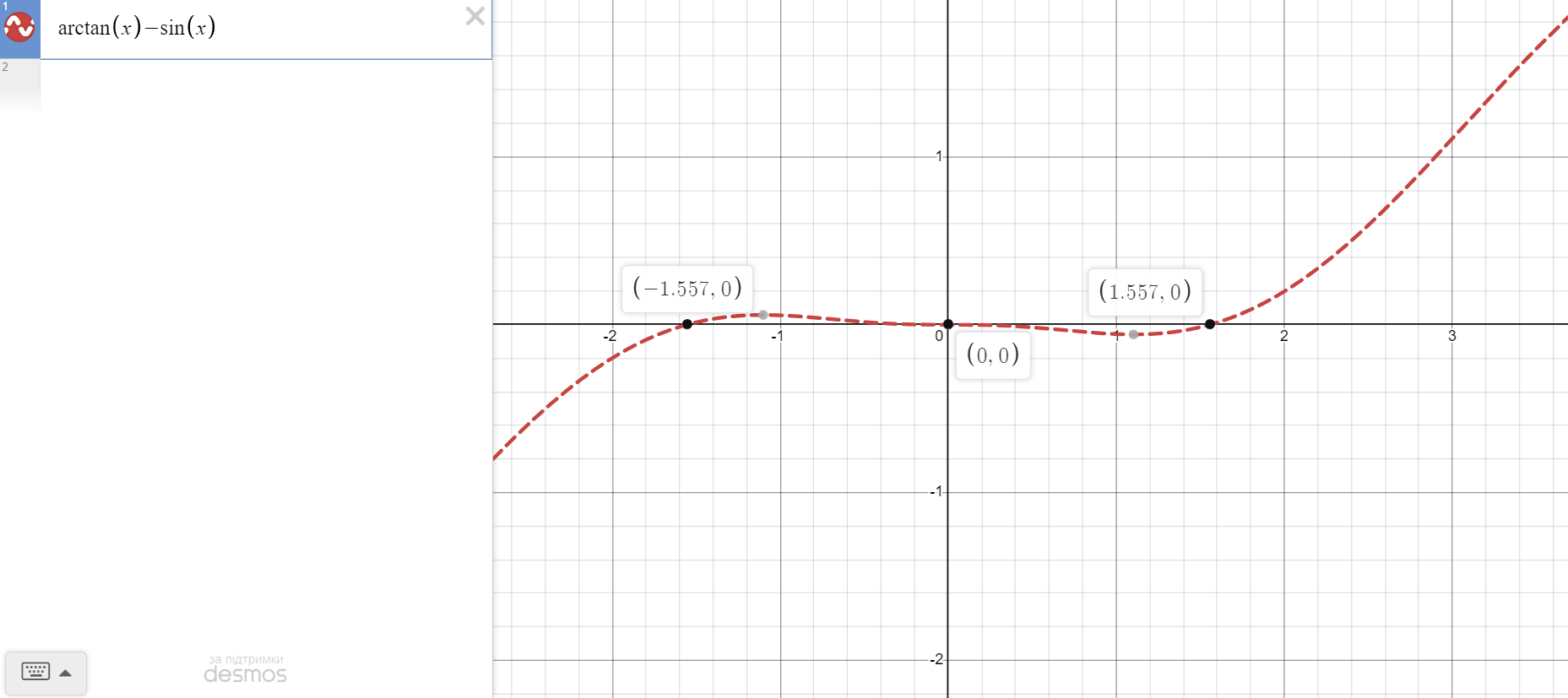


Рис 1. Графік функції

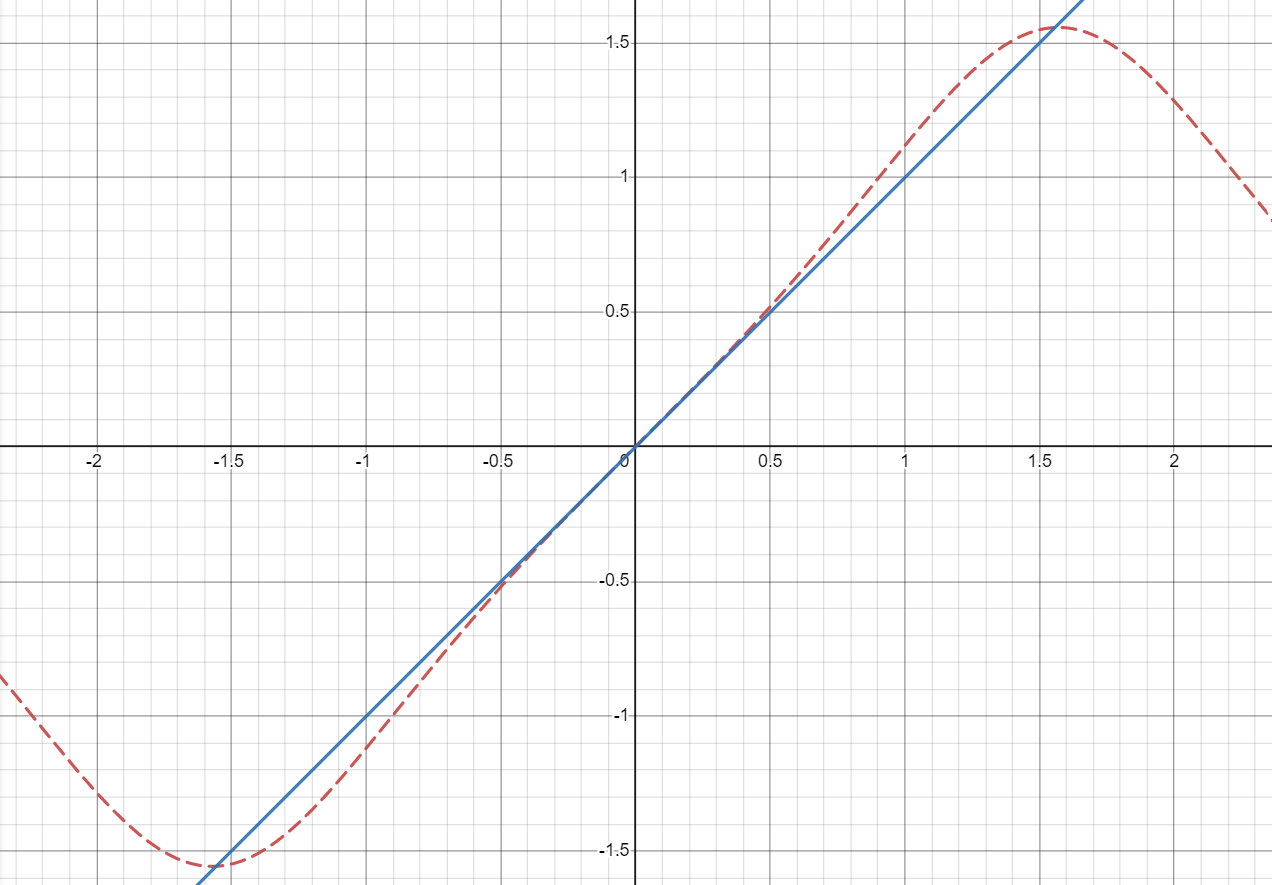


Рис 2. Графік перетину функцій та

## **Метод дотичних**

Наведемо основні етапи методу дотичних:

1. Задати початкове наближення  так, щоб виконувалася умова  . Задати точність . Покласти .
2. Обчислити  за формулою ,
3. Якщо  , процес закінчити і покласти .
4. Якщо , покласти  і перейти до п.2.

**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 Newton – findNewt та параметризуємо його отриманими в попередньому пункті значеннями.
3. викликаємо на об‘єкті findNewt функцію Find() (Рис.3.);
4. в функції Find() викликається функція SetLimits()(Рис.4.), 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 дотичних(див. вище). Функція повертає структуру, що містить кількість ітерацій та результат.
6. виводимо результат виконання програми на екран (Рис.5.).

****

Рис.3. Функція Find()

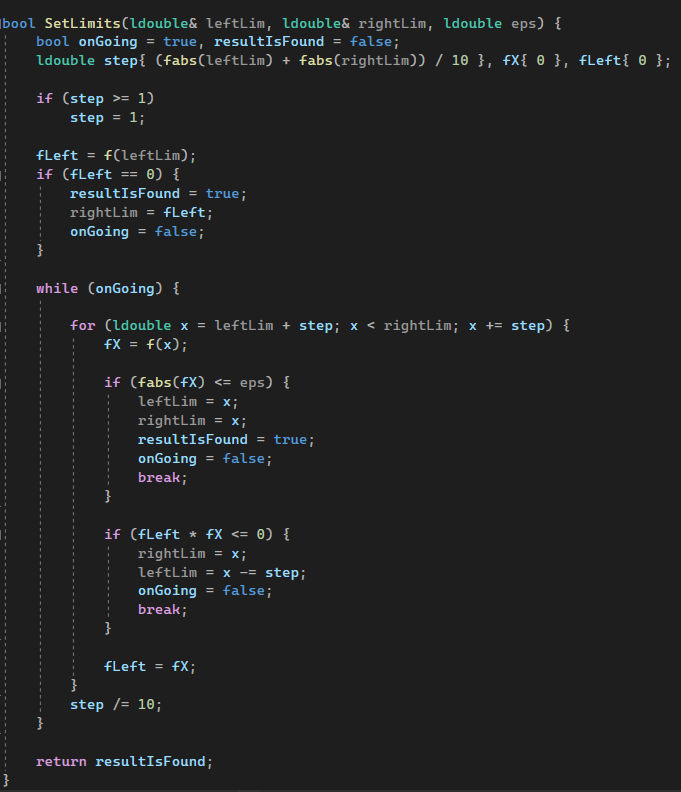


Рис.4. Функція SetLimits()

**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);

}

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 (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;

}

ldouble Get1D(ldouble x) {

return (1 / (x \* x + 1) - cos(x));

}

class Newton {

private:

ldouble leftLim;

ldouble rightLim;

ldouble eps;

SResult result;

ldouble Get2D(ldouble x) {

return sin(x) - 2 \* x / ((x \* x + 1) \* (x \* x + 1));

}

ldouble GetX(ldouble x) {

return x - (f(x) / Get1D(x));

}

public:

Newton(ldouble leftLim, ldouble rightLim, ldouble eps) {

this->leftLim = leftLim;

this->rightLim = rightLim;

this->eps = eps;

result = { 0, NAN };

}

SResult Find() {

bool resultIsFound = false;

resultIsFound = SetLimits(leftLim, rightLim, eps);

if (resultIsFound)

result.result = leftLim;

while ((Get1D(leftLim) \* Get1D(rightLim) <= 0 || Get2D(leftLim) \* Get2D(rightLim) <= 0)) {

resultIsFound = SetLimits(leftLim, rightLim, eps);

if (resultIsFound) {

result.result = leftLim;

break;

}

}

if (!resultIsFound) {

ldouble runLim = 0, prevLim = 0;

if (Get2D(leftLim) > 0 && Get1D(leftLim) > 0) {

runLim = rightLim;

}

else if (Get2D(leftLim) > 0 && Get1D(leftLim) < 0)

runLim = leftLim;

else if (Get2D(leftLim) < 0 && Get1D(leftLim) > 0)

runLim = leftLim;

else

runLim = rightLim;

while (fabs(runLim - prevLim) > eps) {

result.iterations++;

prevLim = runLim;

runLim = GetX(runLim);

}

result.result = runLim;

}

return result;

}

};

class SimpleIterations {

private:

ldouble leftLim;

ldouble rightLim;

ldouble eps;

SResult result;

ldouble GetCloser(ldouble x, ldouble k) {

return x - f(x)/k;

}

public:

SimpleIterations(ldouble leftLim, ldouble rightLim, ldouble eps) {

this->leftLim = leftLim;

this->rightLim = rightLim;

this->eps = eps;

result = { 0, NAN };

}

SResult Find() {

bool resultIsFound = false;

resultIsFound = SetLimits(leftLim, rightLim, eps);

result.result = leftLim;

if (!resultIsFound){

ldouble maxDerivative = 0, currentDerivative = 0, prevRes = leftLim;

for (ldouble n = leftLim; n < rightLim; n += fabs(leftLim + rightLim) / 10)

if (fabs(currentDerivative = Get1D(n)) > maxDerivative)

maxDerivative = currentDerivative;

result.result = GetCloser(result.result, maxDerivative);

while (!(fabs(result.result - prevRes) <= eps)) {

result.iterations++;

prevRes = result.result;

result.result = GetCloser(result.result, maxDerivative);

}

}

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;

Newton findNewt(left, right, eps);

SimpleIterations findSimpIt(left, right, eps);

SResult newtRes = findNewt.Find();

SResult SimpItRes = findSimpIt.Find();

cout << "Newton: " << endl << "Iterations: " << newtRes.iterations << endl << "Result: " << newtRes.result << endl << endl;

cout << "Simple Iterations: " << endl << "Iterations: " << SimpItRes.iterations << endl << "Result: " << SimpItRes.result << endl << endl;

return 0;

}