 Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

До лабораторної роботи №2

**На тему:**  «Розв’язування нелінійних рівнянь методом дотичних та методом послідовних наближень»

**З дисципліни:** “Чисельні методи”

**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 методом дотичних та 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 послідовних наближень визначити корінь рівняння з точністю 10-3. Рівняння:  . Код програмної реалізації розв’язання нелінійного рівняння подано у додатку.

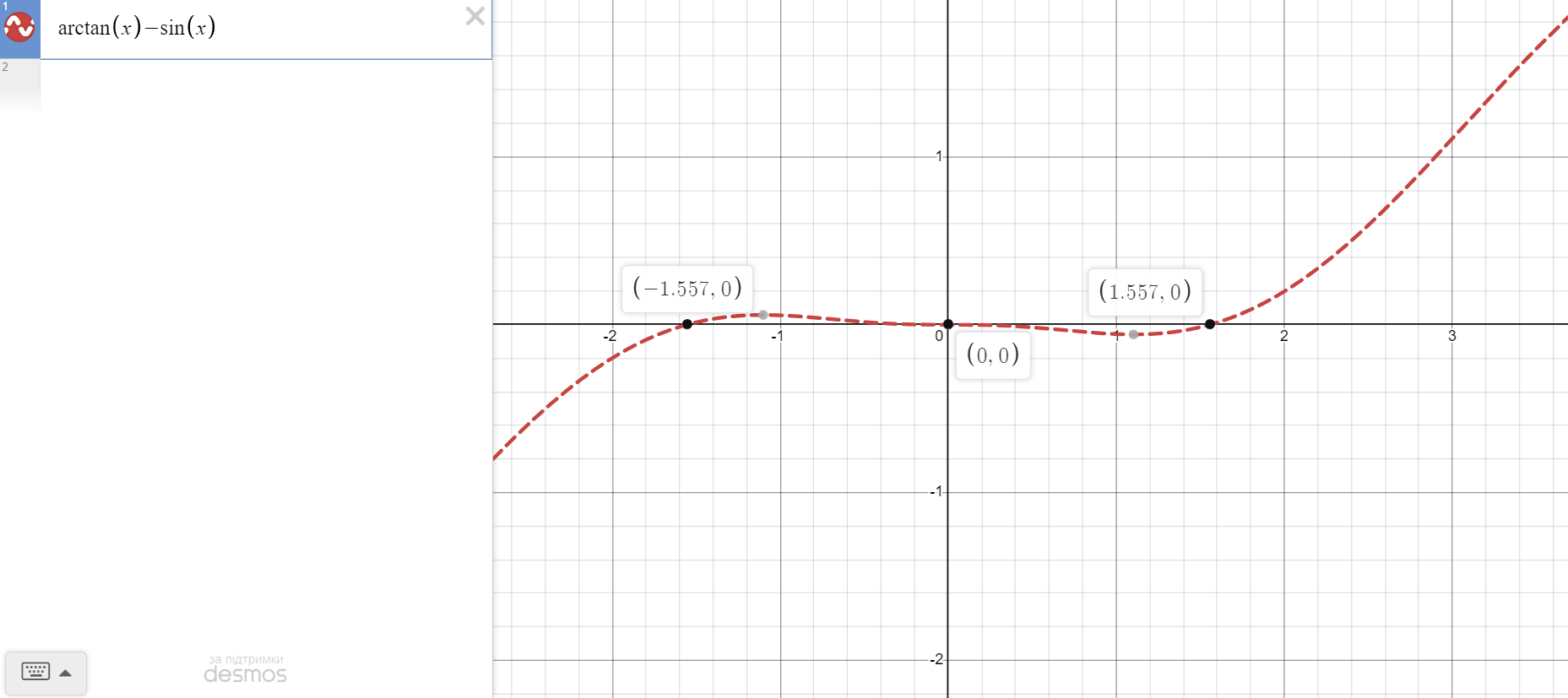


Рис. 1. Графік функції

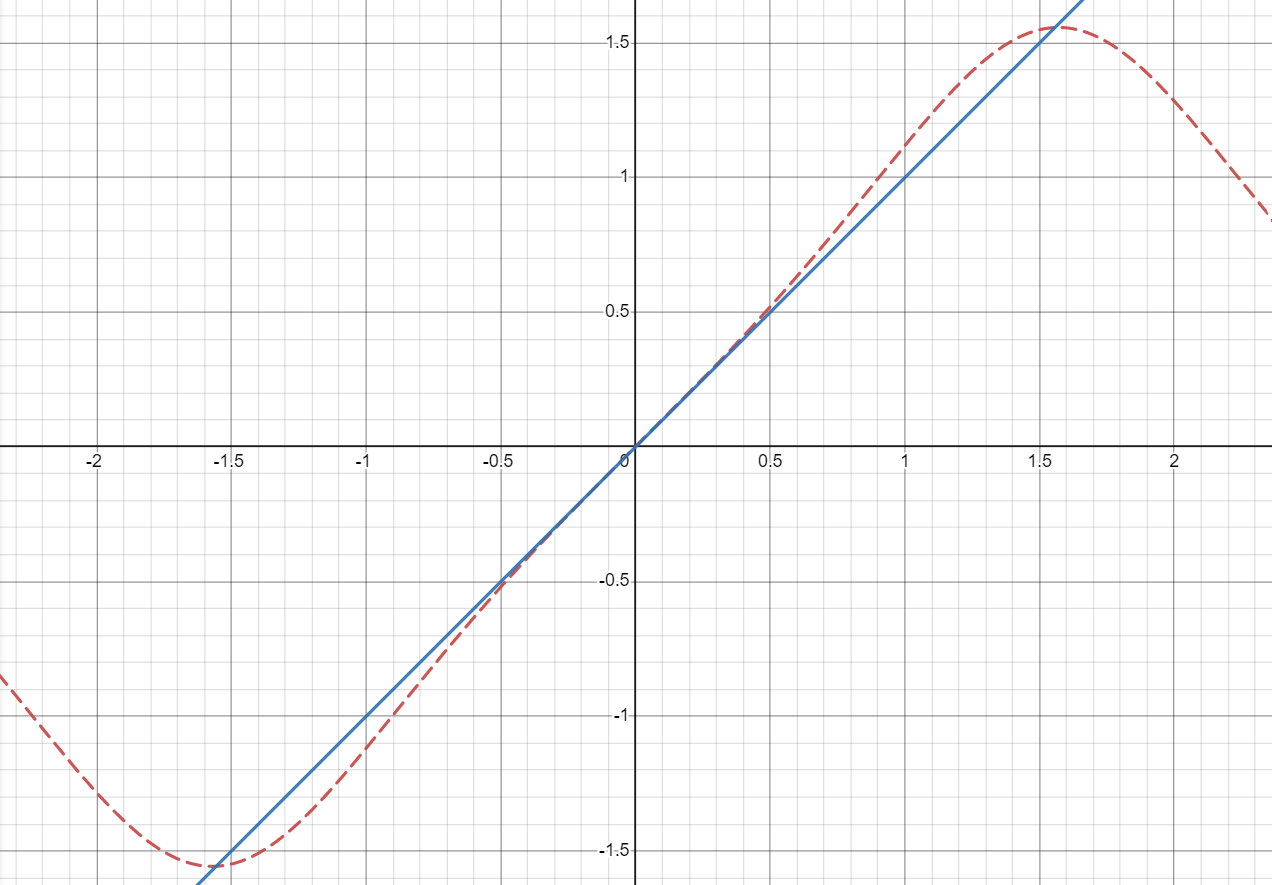


Рис. 2. Графік перетину функцій та

**Метод послідовних наближень**

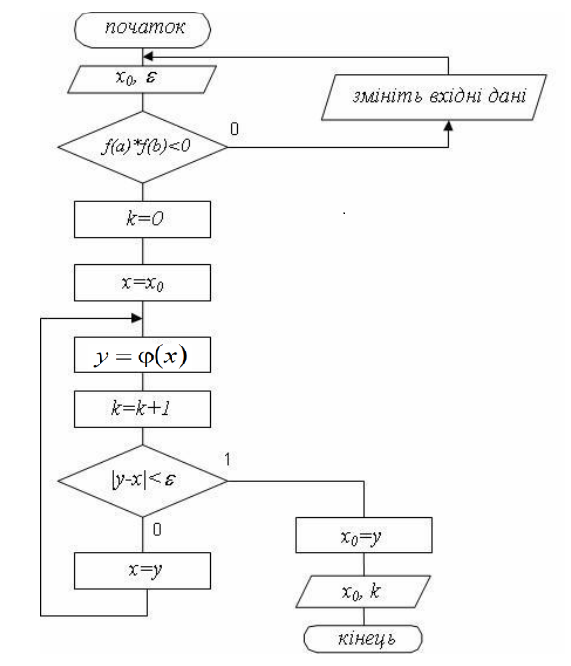


Рис. 3. Блок-схема пошуку коренів ітераційним методом

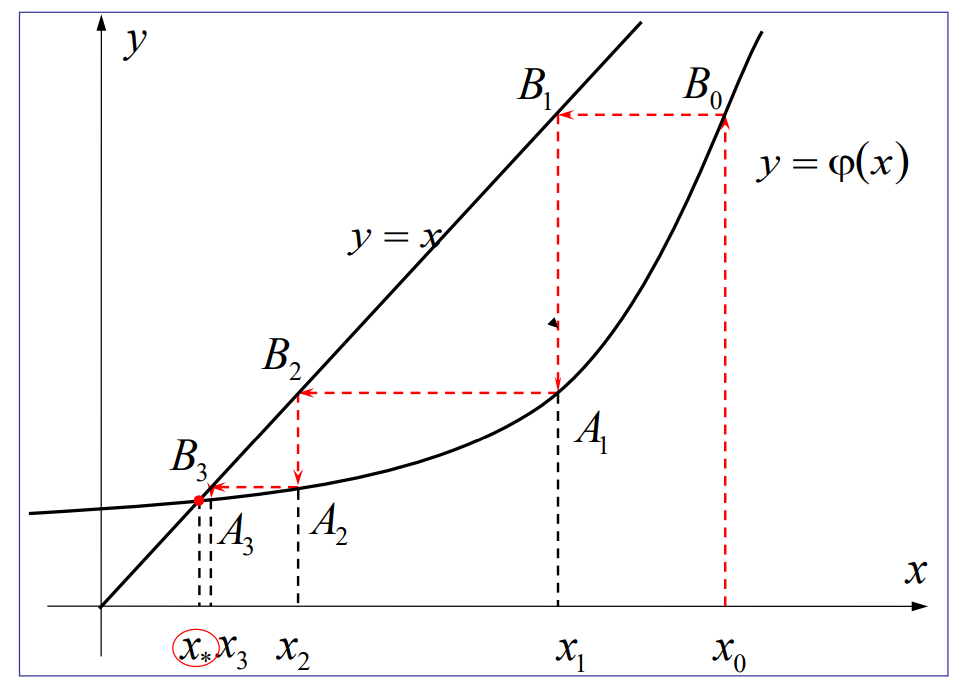
****

Рис. 4. Геометрична інтерпретація пошуку розв’язку методом послідовних наближень

**Основні етапи обчислювального алгоритму, реалізованого у програмному продукті мовою C++ :**

1. введення даних користувачем для ініціалізації змінних(leftLim, rightLim, eps), що визначають початковий проміжок на якому міститься корінь;
2. створюємо екземпляр класу SimpleIterations – findSimpIt та параметризуємо його величинами, які визначають шуканий інтервал локалізації кореня.
3. викликаємо об‘єктом findSimpIt функцію Find() (рис. 5.);
4. із використання функції Find() викликається функція SetLimits() (рис. 6.), яка дає змогу перевірити нерівність (leftLim < rightLim) та локалізувати корінь на визначеному проміжку;
5. перевірка ітераційного процесу на збіжність (рис. .)
6. якщо корінь локалізовано повертаємося у функцію Find(), яка описує алгоритм методу похідних. Функція повертає структуру, що містить кількість ітерацій та результат;
7. виводимо результат виконання програми на екран (рис. 8.);

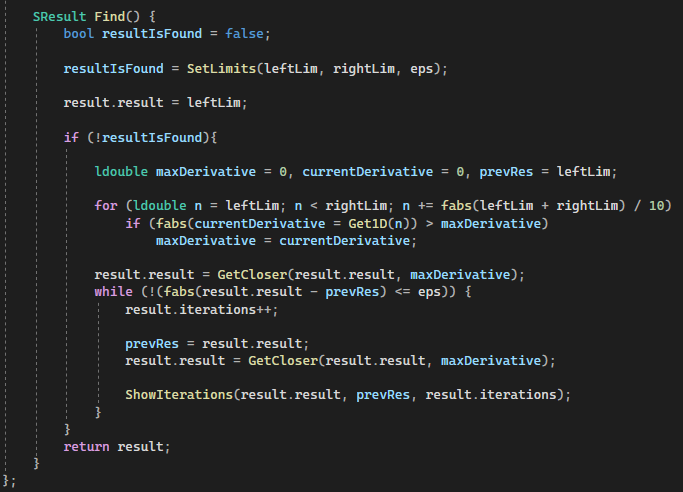


Рис. 5. Метод Find() об’єкту класу Secant

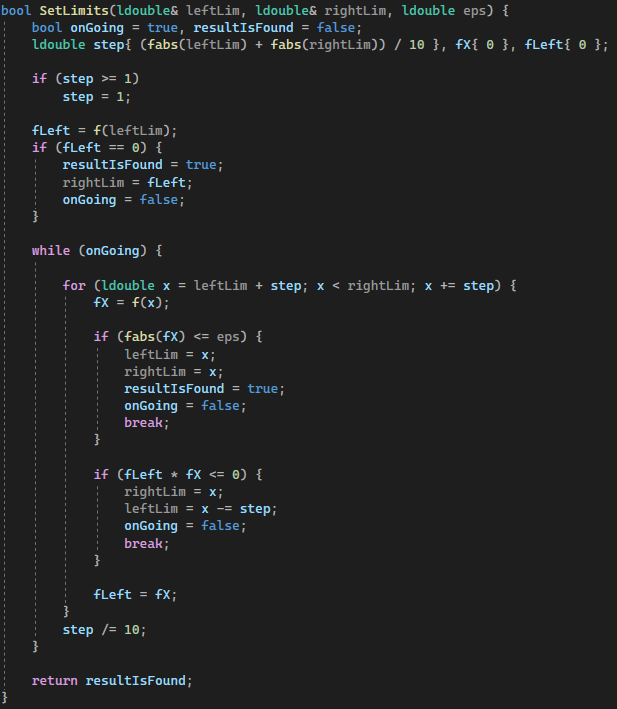


Рис. 6. Функція SetLimits()



Рис. 7. Перевірка ітераційного процесу на збіжність

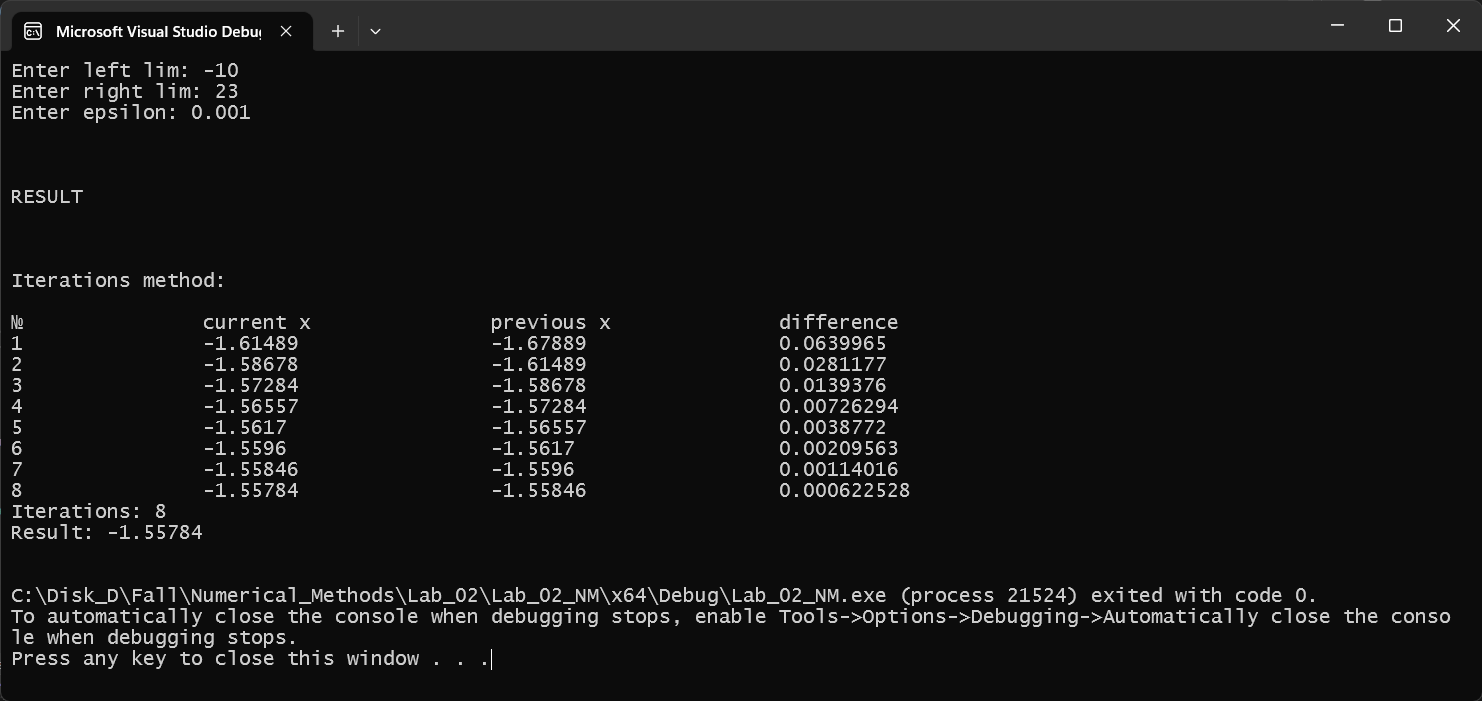
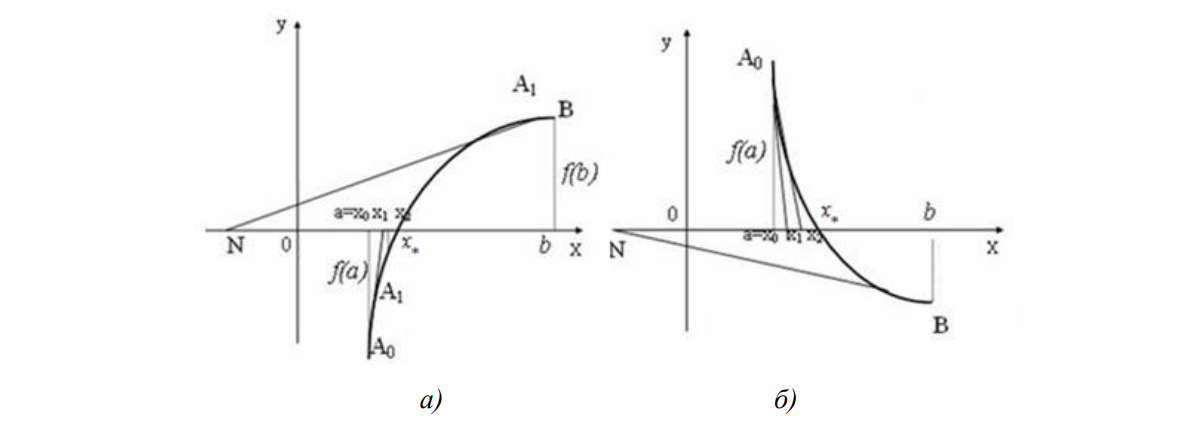
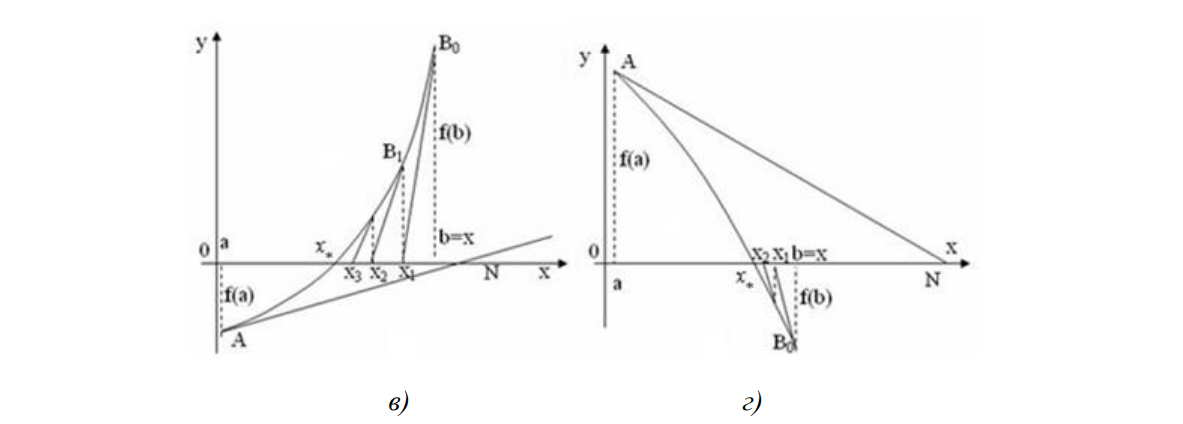


Рис. 8. Результат виконання програми

**Метод дотичних**

****

****

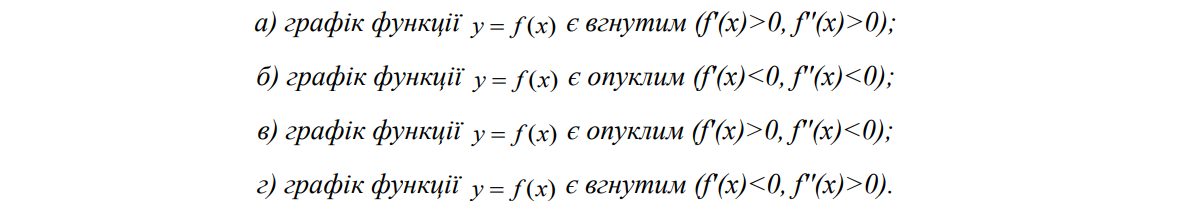
****

Рис. 9. Геометрична інтерпретація пошуку розв’язку методом дотичних

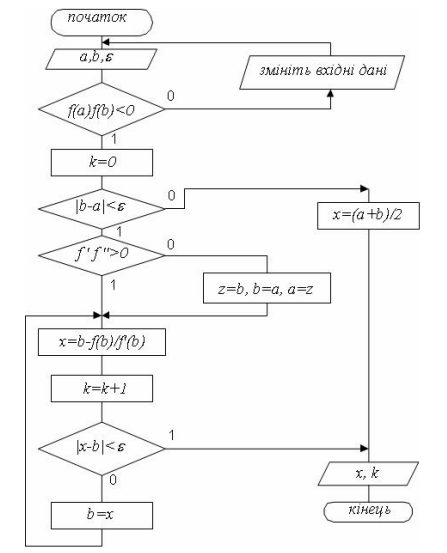


Рис. 10. Блок-схема пошуку коренів методом дотичних

**Хід виконання роботи програми:**

1. введення даних користувачем для ініціалізації змінних(leftLim, rightLim, eps), що визначають початковий проміжок на якому міститься корінь;
2. створюємо екземпляр класу Newton – findNewt та параметризуємо його величинами, які визначають шуканий інтервал локалізації кореня.
3. викликаємо об‘єктом findNewt функцію Find() (рис. 11.);
4. із використання функції Find() викликається функція SetLimits()

(рис. 6.), яка дає змогу перевірити нерівність (leftLim < rightLim) та локалізувати корінь на визначеному проміжку;

1. якщо корінь локалізовано повертаємося у функцію Find(), яка описує алгоритм методу похідних. Функція повертає структуру, що містить кількість ітерацій та результат;
2. виводимо результат виконання програми на екран (рис. 12.);



Рис. 11. Метод Find() об’єкту класу Newton

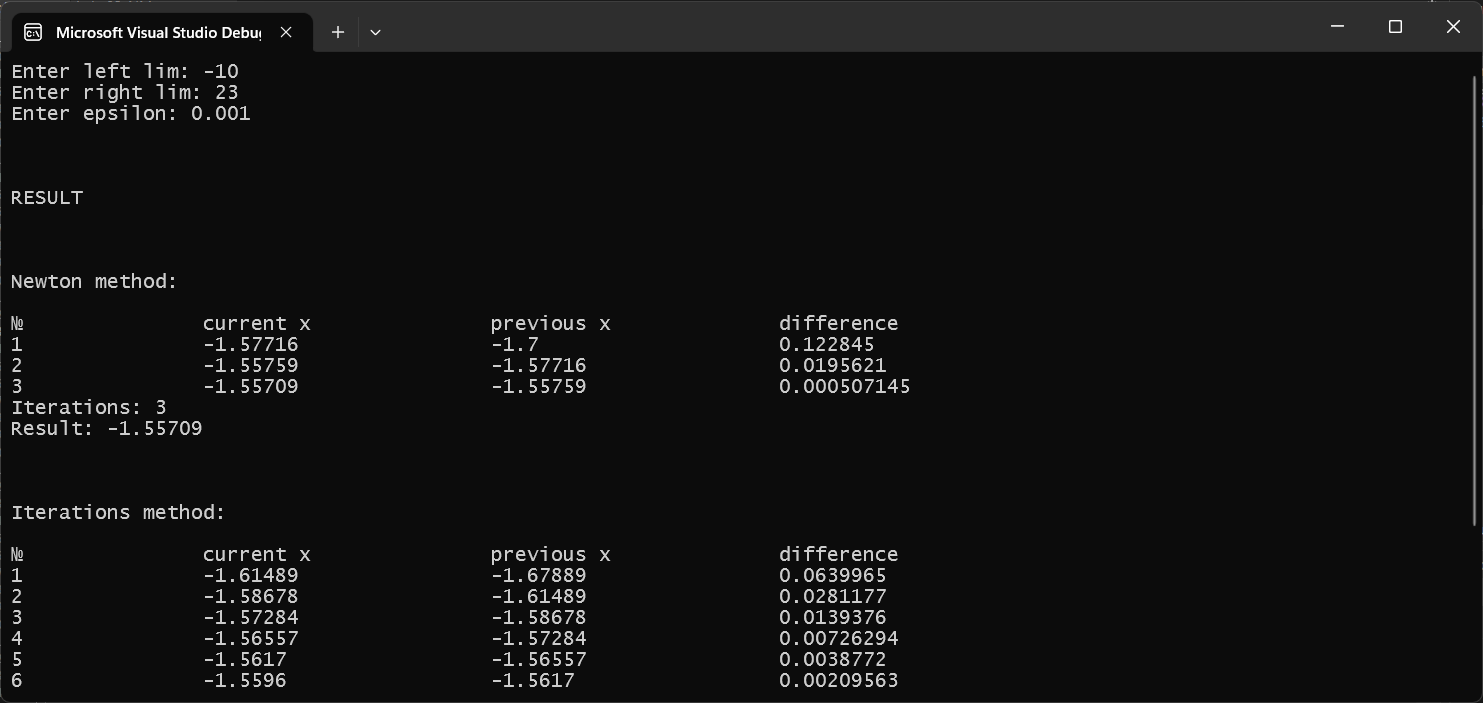


Рис. 12. Результат виконання програми

**Висновки**

У результаті виконання лабораторної роботи визначено дійсний корінь нелінійного алгебраїчного рівняння з заданою точністю 10-3 методом простої ітерації та методом дотичних. Розв’язок отриманий з заданою точністю методом простої ітерації за 8 кроків, а за методом дотичних – за 3 кроки. Дані методи дають змогу знайти дійсний корінь за меншу кількість кроків ніж методи дихотомії та хорд.

**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));

}

void ShowIterations(const ldouble current\_x, const ldouble previous\_x, const int number\_of\_iterations) {

if (number\_of\_iterations == 1)

cout << endl << endl << "Iterations method:" << endl << endl << "№\t\t" << "current x\t\t" << "previous x\t\t" << "difference\t\t" << endl;

cout << number\_of\_iterations << "\t\t" << current\_x << " \t\t" << previous\_x << " \t\t" << fabs(current\_x - previous\_x) << endl;

}

void ShowNewton(const ldouble current\_x, const ldouble previous\_x, const int number\_of\_iterations) {

if (number\_of\_iterations == 1)

cout << endl << endl << "Newton method:" << endl << endl << "№\t\t" << "current x\t\t" << "previous x\t\t" << "difference\t\t" << endl;

cout << number\_of\_iterations << "\t\t" << current\_x << "\t\t" << previous\_x << " \t\t" << fabs(current\_x - previous\_x) << endl;

}

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);

ShowNewton(runLim, prevLim, result.iterations);

}

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);

ShowIterations(result.result, prevRes, result.iterations);

}

}

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();

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

SResult SimpItRes = findSimpIt.Find();

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

return 0;

}