Міністерство освіти і науки, молоді та спорту України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**про виконання лабораторної роботи № 2**

«**«**Розв’язування нелінійних рівнянь методом дотичних та послідовних обчислень»

**з дисципліни «Чисельні методи»**

**Лектор:**

доцент кафедри ПЗ

Мельник Н.Б.

**Виконала:**

студ. групи ПЗ-12

Ліхтарчик С.Й.

**Прийняла:**

доцент кафедри ПЗ

Мельник Н.Б.

«\_\_\_» \_\_\_\_\_\_ 2020 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2020р.

**Тема роботи:** Розв’язування нелінійних рівнянь методом дотичних та наближених обчислень.

**Мета роботи:** Ознайомлення на практиці з методом дотичних та методом послідовних наближень для розв’язування нелінійних рівнянь.

.

**Теоретичні відомості**

1. Метод Ньютона (метод дотичних)

***Формулювання задачі****.* Розглянемо рівняння , де  є неперервною монотонною нелінійною функцією, яка на кінцях відрізку приймає значення різних знаків, причому її похідні та є неперервними та монотонними. Потрібно знайти значення кореня  з заданою похибкою .

Геометричний зміст методу Ньютона полягає в тому, що дугу кривої на відрізку замінюють дотичною до цієї кривої, а наближене значення кореня визначають як абсцису точки перетину дотичної з віссю *,* проведеної через один із кінців відрізка. Запишемо рівняння дотичної до кривої  в точці 

.

Ітераційні формули:

, .

Для вибору початкового наближення кореня рівняння  необхідно керуватися таким правилом: за початкову точку слід вибирати той кінець відрізка , в якому знак функції співпадає зі знаком її другої похідної. У першому випадку  і за початкову точку вибираємо , а в другому‑і тому . , де – задана точність шуканого розв’язку; – наближені значення кореня рівняння  на -му та ()-му кроках.

1. Метод простої ітерації (метод послідовних наближень)

Одним із найпоширеніших методів чисельного розв’язування нелінійних рівнянь є метод простої ітерації. Іноді його називають методом послідовних наближень.

***Формулювання задачі****.* Розглянемо нелінійне рівняння , де  є неперервною функцією. Потрібно знайти хоча б один дійсний корінь цього рівняння. Рівняннязапишемо у канонічній формі

. (3.4)

Довільним способом визначимо наближене значення кореня рівняння і підставимо його в праву частину співвідношення (3.4). У результаті отримаємо

. (3.5)

Підставивши тепер в праву частину рівняння (3.5) замість  значення , отримаємо . Повторюючи цей процес, отримаємо ітераційні формули

, . (3.6)

Кожний дійсний корінь рівняння (3.6) є абсцисою точки перетину  кривої з прямою .

визначати функцію  зі співвідношення

, (3.8)

де значення вибирають так, щоб виконувалась умова . Тут та знак  співпадає зі знаком  на відрізку.

Ітераційний процес продовжують до тих пір, поки не виконуватиметься умова



**Індивідуальне завдання**

**Індивідуальне завдання**

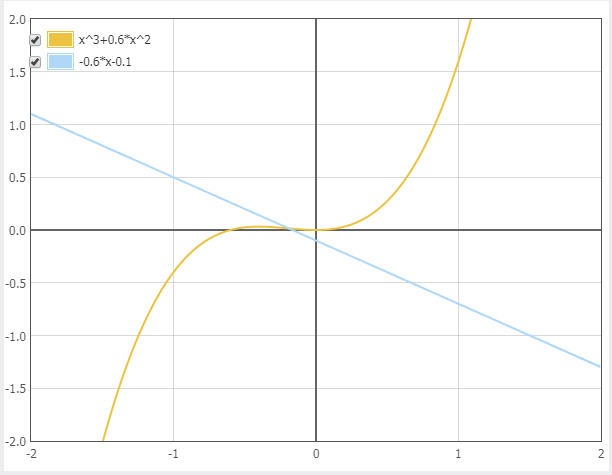
Написати програму розв’язку нелінійного рівняння відповідно до варіанту методами дотичних та ітерацій:

**Хід роботи**

1. Ознайомлення з теоретичним матеріалом.
2. Відокремлення дійсних коренів рівняння ***геометричним способом***:

Перепишемо рівняння у такому вигляді:

Будуємо графік.



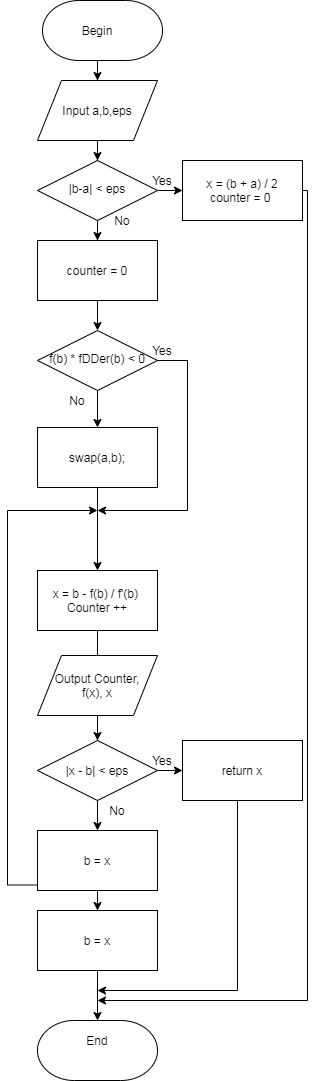
З графіку можна виокремити цілий інтервал у якому знаходиться корінь нелінійного алгебраїчного рівняння : [-0.3; 0.4]. Також було отримано наближений розв’язок :

Для знаходження розв’язків рівняння методом Ньтона було знайдено першу другу похідну :

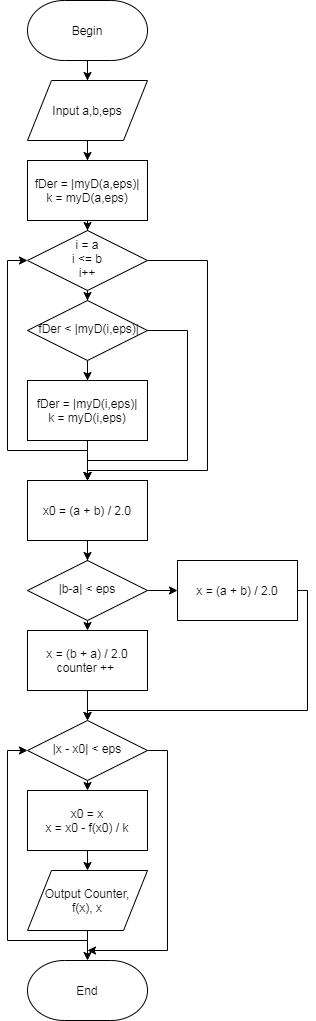
Для знаходження розв’язків рівняння методом ітерацій знайдено функцію .

1. **Блок-схема алгоритму:**

Метод дотичнх (метод Ньютона):



Метод ітерацій



1. Код програми:

main.cpp

#pragma once

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

double function(double);

double functionDer(double);

double functionDDer(double);

bool rootsExist(double, double); //checking for roots

void swap(double\*, double\*);

double newtonMethod(double, double, double);

double myDeritative(double x, double eps);

double iterationMethod(double a, double b, double eps);

int newtonIterationsCount;

int iterIterationsCount;

int main()

{

double eps = 1e-2;

double lBorder = 0.5;

double rBorder = 1.5;

cout << "This program computs given function using borders [0.5;1.5] and 1e-2 precision." << endl;

cout << "If you want to comput with these default value, press 1.\nTo enter new ones, 0 is your choice : ";

bool modeSwitch;

cin >> modeSwitch;

if (!modeSwitch)

{

while (cout << "Enter a, b and eps : " && !(cin >> lBorder >> rBorder >> eps)) {

std::cin.clear(); //clear bad input flag

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); //discard input

std::cout << "Invalid input; please re-enter.\n";

}

}

if (rootsExist(lBorder, rBorder))

{

cout << "\n" << setw(12) << right << "Step" << setw(13) << right << "y =" << setw(13) << right << "x =" << endl;

cout << "Newton method did " << newtonIterationsCount << " iterations and gave a result : " << newtonMethod(lBorder, rBorder, eps) << endl;

cout << "\n" << setw(12) << right << "Step" << setw(13) << right << "y =" << setw(13) << right << "x =" << endl;

cout << "Iterations method did " << iterIterationsCount << " iterations and gave a result : " << iterationMethod(lBorder, rBorder, eps) << endl;

}

else

{

cout << "No roots on given interval." << endl;

}

system("pause");

return 0;

}

double function(double x) //given function

{

return (x \* x \* x + 0.3 \* x \* x - 0.6 \* x - 0.1);

}

double functionDer(double x)

{

return (3 \* x \* x + 0.6 \* x - 0.6);

}

double functionDDer(double x) //second derivative

{

return (6 \* x + 0.6);

}

bool rootsExist(double lBorder, double rBorder)

{

return (function(lBorder) \* function(rBorder)) < 0 ? true : false;

}

void swap(double\* a, double\* b)

{

double\* temp = b;

b = a;

a = temp;

}

double newtonMethod(double lBorder, double rBorder, double eps)

{

double x = 0;

if (fabs(rBorder - lBorder) < eps)

{

x = (rBorder - lBorder) / 2;

newtonIterationsCount = 0;

}

else

{

newtonIterationsCount = 0;

if (!(function(rBorder) \* functionDDer(rBorder) > 0))

{

swap(lBorder, rBorder);

}

while (true)

{

x = rBorder - function(rBorder) / functionDer(rBorder);

newtonIterationsCount++;

cout << setw(12) << newtonIterationsCount << " " << setw(12) << function(x) << " " << setw(12) << x << endl;

if (fabs(x - rBorder) < eps)

{

break;

}

else

{

rBorder = x;

}

rBorder = x;

}

}

return x;

}

double myDeritative(double x, double eps) {

double d1 = (function(x + eps) - function(x - eps)) / (2 \* eps);

return d1;

}

double iterationMethod(double a, double b, double eps) {

double fDer = fabs(myDeritative(a, eps)); //first deritative

double k = myDeritative(a, eps) / 2; //we need k because |max(f`(x))| < 1

for (double i = a; i <= b; i += eps) //loop for finding k from l to r border

{

if (fDer < fabs(myDeritative(i, eps)))

{

fDer = fabs(myDeritative(i, eps));

k = myDeritative(i, eps) / 2;

}

} //by the end we have our k, which is used in algorithm

iterIterationsCount = 0;

double x, x0;

x0 = (a + b) / 2.0; // starting value for x0

if (fabs(b - a) < eps) { // if roos is already found

x = (b + a) / 2.0; // calculate root

iterIterationsCount++;

return x;

}

else {

x = (a + b) / 2.0;

do {

x0 = x;

x = x0 - (function(x0) / k);

cout << setw(12) << iterIterationsCount << " " << setw(12) << function(x) << " " << setw(12) << x << endl;

iterIterationsCount++;

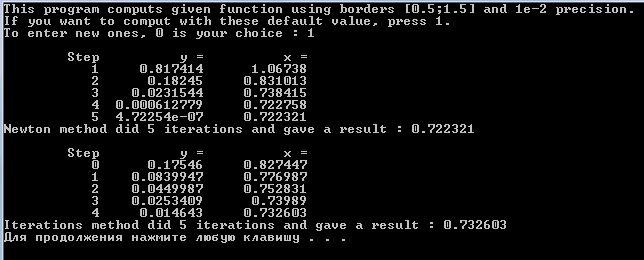
} while (fabs(x - x0) >= eps);

}

return x;

}

1. Результат програми:



**Висновки**

Під час виконання даної лабораторної роботи я ознайомився з двома методами уточнення розв’язку нелінійних рівнянь, а саме з методом хорд та методом послідовних ітерацій. Обидва методи реалізував програмно, та використав їх для розв’язування заданого рівняння.