**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

До лабораторної роботи №2

**На тему: “Розв’язування нелінійних рівнянь методом дотичних та методом послідолвних наближень”**

**З дисципліни:** *“Чисельні методи”*

**Лектор:**

доц. каф. ПЗ

Мельник Н. Б.

**Виконав:**

ст. гр. ПЗ-16

Кабачок Т. О.

**Прийняв:**

доц. каф. ПЗ

Мельник Н. Б.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

Львів – 2025

**Тема роботи:** Розвʼязування нелінійних рівнянь методом дотичних та методом послідовних наближень.

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

**Теоретичні відомості**

**Метод Ньютона (метод дотичних)**

Метод Ньютона — це ітераційний метод для наближеного розв'язання рівнянь виду , де є неперервною монотонною нелінійною функцією на відрізку , з різними знаками функції на кінцях відрізка. Важливими властивостями функції є її неперервність, а також неперервність і монотонність похідних та

Геометричний зміст методу Ньютона: на графіку функції на відрізку замінюється дуга кривої дотичною. Наближене значення кореня функції визначається як абсциса точки перетину дотичної з віссю .

**Формулювання методу:**

1. Рівняння дотичної до кривої в точці має вигляд:

При отримаємо:

Ітераційна формула методу Ньютона:

**Вибір початкового наближення:**

* Початкове значеннявибирається таким чином:
  + Якщо , то
  + Якщо , то

**Завершення ітерацій:**

Ітерації продовжуються до тих пір, поки не буде виконана умова точності:

де — задана точність шуканого кореня рівняння.

**Метод простої ітерації (метод послідовних наближень)**

Розглянемо нелінійне рівняння неперервна функція. Потрібно знайти хоча б один дійсний корінь цього рівняння. Рівняння записуємо у канонічній формі:

Наближене значення кореня підставляємо в праву частину співвідношення. Отримуємо:

Повторюючи цей процес, отримуємо ітераційні формули:

**Умова збіжності:**

Ітераційний процес збігається до єдиного кореня, якщо на відрізку , що містить цей корінь, виконується умова:

Збіжність процесу буде тим швидшою, чим меншим є максимальне значення

на відрізку.

**Перетворення рівняння:**

Якщо умова збіжності не виконується, можна перетворити рівняння до вигляду

так, щоб вона виконувалася. Наприклад, можна використати таке перетворення:

де — вибране таке значення, щоб умова збіжності виконувалась.

**Завершення ітерацій:**

Ітерації продовжуються, поки не буде виконана умова точності:

де — задана похибка шуканого кореня.

**Хід роботи**

Файл **main.rs**

mod functions;

use plotters::prelude::\*;

use std::io;

fn f(x: f64) -> f64 {

x.powi(3) - 6.0 \* x.powi(2) - 7.0

}

fn phi(x: f64) -> f64 {

(6.0 \* x.powi(2) + 7.0) / x.powi(2)

}

fn plot\_graphs() -> Result<(), Box<dyn std::error::Error>> {

let root = BitMapBackend::new("graphs.png", (800, 800)).into\_drawing\_area();

root.fill(&WHITE)?;

let mut chart = ChartBuilder::on(&root)

.caption("Графіки функцій", ("sans-serif", 50))

.margin(20)

.x\_label\_area\_size(30)

.y\_label\_area\_size(30)

.build\_cartesian\_2d(-50.0..50.0, -50.0..50.0)?;

chart.configure\_mesh().draw()?;

chart.draw\_series(LineSeries::new(

(-1000..=1000).map(|x| (x as f64 / 10.0, f(x as f64 / 10.0))),

&RED,

))?.label("y = f(x)").legend(|(x, y)| {

Rectangle::new(

[(x - 5, y - 5), (x + 5, y + 5)],

ShapeStyle {

color: RED.to\_rgba(),

filled: true,

stroke\_width: 0,

},

)

});

chart.draw\_series(LineSeries::new(

(-1000..=1000).map(|x| (x as f64 / 10.0, x as f64 / 10.0)),

&BLUE,

))?.label("y = x").legend(|(x, y)| {

Rectangle::new(

[(x - 5, y - 5), (x + 5, y + 5)],

ShapeStyle {

color: BLUE.to\_rgba(),

filled: true,

stroke\_width: 0,

},

)

});

chart.draw\_series(LineSeries::new(

(-1000..=1000).map(|x| (x as f64 / 10.0, phi(x as f64 / 10.0))),

&GREEN,

))?.label("y = φ(x)").legend(|(x, y)| {

Rectangle::new(

[(x - 5, y - 5), (x + 5, y + 5)],

ShapeStyle {

color: GREEN.to\_rgba(),

filled: true,

stroke\_width: 0,

},

)

});

chart.configure\_series\_labels().border\_style(&BLACK).draw()?;

root.present()?;

println!("Графіки збережено у graphs.png");

Ok(())

}

fn main() {

println!("Автоматизований режим (межі 6 - 7, точність 0.01) - введіть 1");

println!("Ручний режим (межі і точність через enter) - введіть 2");

let mut input = String::new();

io::stdin().read\_line(&mut input).unwrap();

let reg: i64 = input.trim().parse().unwrap();

input.clear();

if reg == 1 {

println!("Метод дихотомії:");

functions::dix(0, 6.0, 7.0, 0.01);

println!("Метод хорд:");

functions::hord(0, 6.0, 7.0, 0.01);

println!("Метод Ньютона:");

functions::newton(0, 6.0, 7.0, 0.01);

println!("Метод простої ітерації:");

functions::iteration(0, 6.0, 6.0, 7.0, 0.01);

} else if reg == 2 {

println!("Введіть a");

io::stdin().read\_line(&mut input).unwrap();

let a: f64 = input.trim().parse().unwrap();

input.clear();

println!("Введіть b");

io::stdin().read\_line(&mut input).unwrap();

let b: f64 = input.trim().parse().unwrap();

input.clear();

println!("Введіть точність eps");

io::stdin().read\_line(&mut input).unwrap();

let eps: f64 = input.trim().parse().unwrap();

println!("Метод дихотомії");

functions::dix(0, a, b, eps);

println!("Метод хорд");

functions::hord(0, a, b, eps);

println!("Метод Ньютона");

functions::newton(0, a, b, eps);

println!("Метод простої ітерації");

functions::iteration(0, a, a, b, eps);

}

plot\_graphs().expect("Не вдалося побудувати графіки");

}

Файл **functions.rs**

pub fn prnt(n: i64, a: f64, b: f64, absol: f64, eps: f64, x: f64){

if n == 0 {

println!("{:<5}{:<14}{:<14}{:<16}{:<14}{:<16}", "N", "a", "b", "|expected eps|", "eps", "x");

}

println!("{:<5}{:<14.5}{:<14.5}{:<16.5}{:<14.5}{:<16.5}", n, a, b, absol, eps, x);

}

pub fn dix(n: i64, mut a: f64, mut b: f64, eps: f64) -> f64{

let x: f64 = (a + b) / 2f64;

if func(a) \* func(b) > 0.0 {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

if (a-b).abs() < eps || func(x) == 0.0{

return x

}

else{

if func(a) \* func(x) > 0.0{

a = x;

}

if func(b) \* func(x) > 0.0{

b = x;

}

prnt(n, a, b, (a-b).abs(), eps, x);

return dix(n+1, a, b, eps)

}

}

pub fn hord(n: i64, a: f64, b: f64, eps: f64) -> f64{

if func(a) \* func(b) > 0.0 {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

let x: f64 = a - (func(a)\*(b-a))/(func(b)-func(a));

if (a-x).abs()<eps || func(x) == 0.0{

return x

}

else{

prnt(n, a, b, (a-x).abs(), eps, x);

return hord(n+1, x, b, eps)

}

}

pub fn newton(n: i64, a: f64, b: f64, eps: f64) -> f64 {

let fx = func(b);

let dfx = func\_derivative(b);

if func(a) \* func(b) > 0.0 {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

if fx.abs() < eps {

return b;

} else {

let x = b - fx / dfx;

if x < a || x > b {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

prnt(n, a, b, (a - b).abs(), eps, b);

return newton(n + 1, a, x, eps);

}

}

pub fn iteration(n: i64, x: f64, a: f64, b: f64, eps: f64) -> f64 {

let g = |x: f64| (6.0 \* x.powi(2) + 7.0) / x.powi(2);

if func(a) \* func(b) > 0.0 {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

let x\_next = g(x);

if x < a || x > b {

panic!("Корінь не знайдений у межах відрізка [{}, {}]", a, b);

}

if (x\_next - x).abs() < eps {

return x\_next;

} else {

prnt(n, x, x\_next, (x - x\_next).abs(), eps, x\_next);

return iteration(n + 1, x\_next, a, b, eps);

}

}

pub fn func(x: f64) -> f64{

x.powi(3) - 6f64 \* x.powi(2) - 7f64

}

pub fn func\_derivative(x: f64) -> f64{

3f64 \* x.powi(2) - 12f64 \* x

}

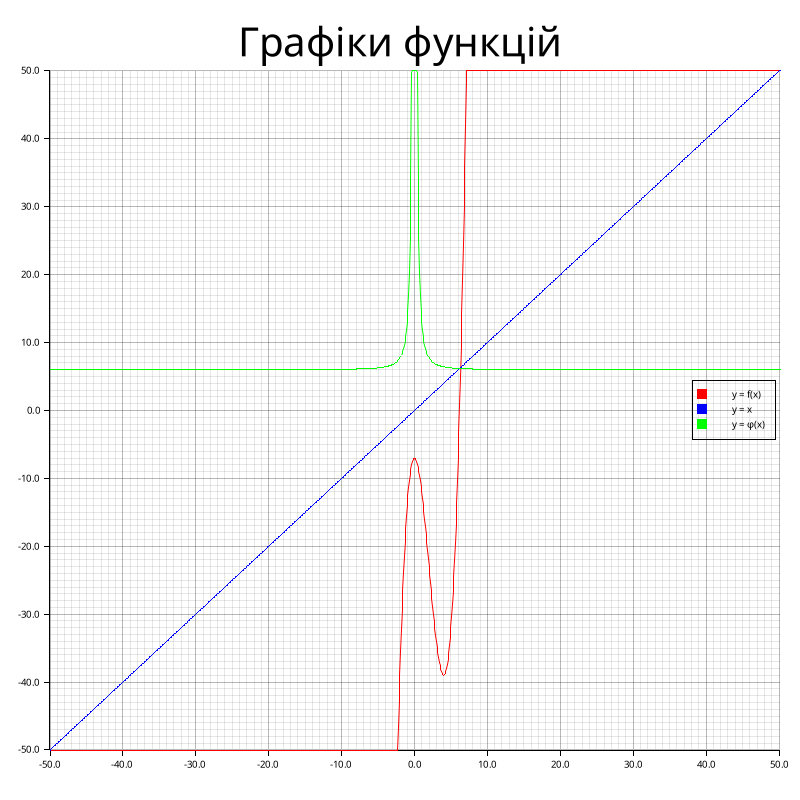
****

Рис. 1. Графіки функцій

**Результат виконання програми**

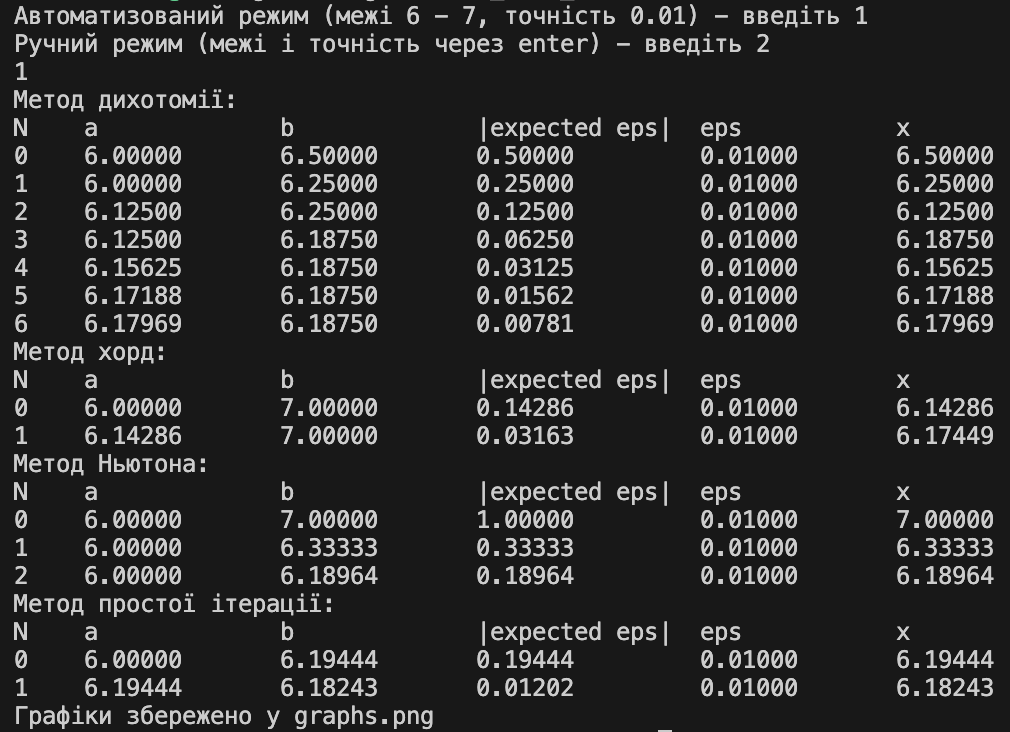


Рис. 2. Результат автоматизованого режиму виконання програми

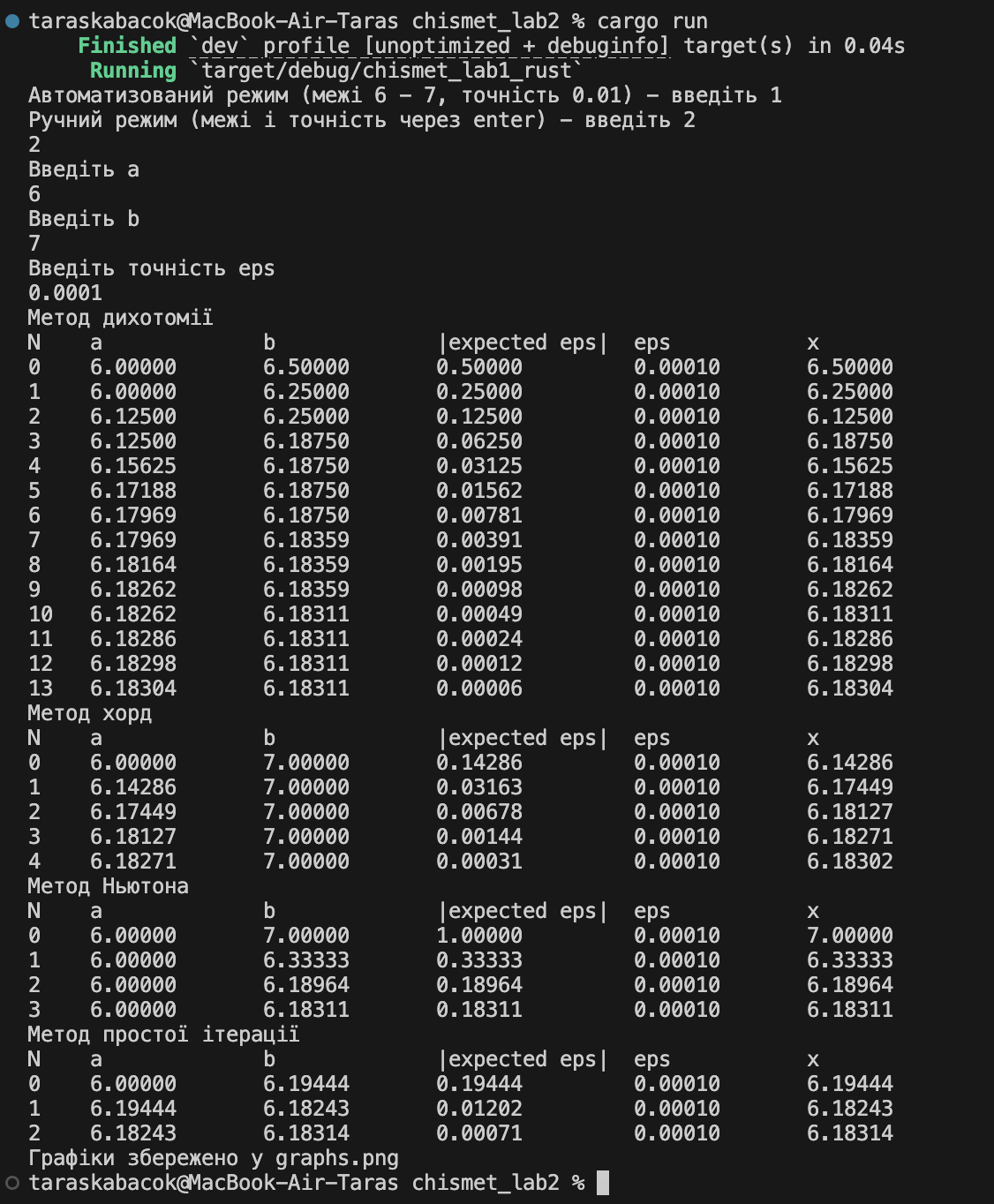


Рис. 3. Результат ручного режиму виконання програми

**Висновки**

Під час виконання даної лабораторної роботи я ознайомився з двома методами чисельного розв'язання нелінійних рівнянь: методом Ньютона та методом простої ітерації. Я розглянув теоретичні основи кожного методу, а також їхні геометричні та аналітичні характеристики. З допомогою мови програмування Rust я реалізував обидва методи, а також програму для виведення результатів на кожному кроці ітерацій. Результати наближених розв'язків підтвердили коректність реалізації, і я отримав значення кореня з заданою точністю. Всі методи показали свою ефективність та застосовність у розв'язанні нелінійних рівнянь.