МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ЗВІТ

про виконання лабораторної роботи №2

з теми МЕТОДИ УТОЧНЕННЯ КОРЕНІВ

НЕЛІНІЙНИХ РІВНЯНЬ

з навчальної дисципліни: “Чисельні методи”

Виконав:

студент групи ІР-24

Шийка Андрій

Прийняв:

Сиротюк С. В.

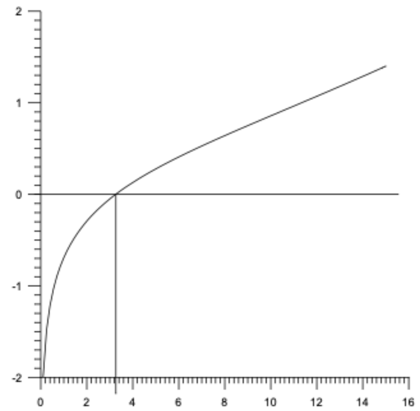
Львів - 2025

Мета роботи: Вивчити основні методи уточнення коренів нелінійних рівнянь з одним невідомим, дослідити збіжність ітераційних процесів та реалізувати відповідні алгоритми програмно.

Завдання:

12	Метод поділу ділянки навпіл з пошуком ділянки локалізації $x \in [0{,}1;\, 16]$	Метод поділу ділянки навпіл $x \in [0{,}1;\, 16]$	Метод простої ітерації $x \in [0{,}1;\, 16]$

№12



$$\ln(x) + x = 0$$

$$f'(x) = 1 + \frac{1}{x}$$

Короткі теоретичні відомості:

1. Локалізація коренів

Задача пошуку коренів рівняння f(x) = 0 складається з двох етапів: локалізації (відокремлення) коренів та їх уточнення до заданої точності.

Локалізація полягає у визначенні відрізка [a, b], на якому знаходиться лише один корінь. Достатньою умовою існування кореня на відрізку є зміна знаку неперервної функції на його кінцях: f(a) * f(b) < 0.

Якщо відрізок не задано, використовують метод табуляції: рухаються від початкової точки з кроком h, перевіряючи умову зміни знаку.

2. Метод поділу проміжку навпіл (бісекції)

Цей метод є найнадійнішим для уточнення кореня. Відрізок локалізації [a, b] ділиться навпіл точкою x = (a + b) / 2.

Далі перевіряється знак функції на половинах відрізка:

- Якщо f(a) * f(x) > 0, то корінь знаходиться в інтервалі [x, b], тому a = x.

- Якщо f(a) * f(x) < 0, то корінь знаходиться в інтервалі [a, x], тому b = x.

Процес повторюється ітераційно, доки довжина відрізка b - a не стане меншою за задану точність 2*eps, або значення функції |f(x)| < eps. Метод збігається повільніше за інші, але гарантовано.

3. Метод простої ітерації

Рівняння f(x) = 0 замінюється еквівалентним рівнянням x = g(x). Ітераційний процес будується за формулою:

x_k = g(x_{k-1}).

Геометрично корінь є точкою перетину графіка y = g(x) та прямої y = x.

Для забезпечення збіжності необхідно, щоб на відрізку локалізації виконувалася умова |g'(x)| < 1.

На практиці часто використовують перетворення x = x + alpha * f(x). Параметр alpha обирають залежно від похідної f'(x):

- Якщо f'(x) > 0, то alpha обирають з інтервалу (-2/max|f'(x)|; 0).

- Якщо f'(x) < 0, то alpha обирають з інтервалу (0; 2/max|f'(x)|).

Список ідентифікаторів констант, змінних, функцій:

x – Змінна для зберігання поточного наближення кореня рівняння.

f – Функція, що обчислює значення лівої частини рівняння f(x) = ln(x) + x.

df – Функція, що обчислює значення похідної f'(x) = 1 + 1/x.

a_start – Початкова ліва межа для алгоритму пошуку локалізації.

a, b – Ліва та права межі поточного відрізка локалізації кореня.

h – Крок пошуку ділянки локалізації.

eps – Задана абсолютна похибка обчислень (epsilon).

eps_percent – Задана відносна похибка обчислень у відсотках.

alpha – Розрахунковий параметр збіжності для методу простої ітерації.

k – Лічильник кількості виконаних ітерацій.

rel_err – Значення відносної похибки на поточній ітерації.

max_df – Максимальне значення модуля похідної на відрізку (для розрахунку alpha).

Код програми:

```
import math


def f(x: float) -> float:
    """
    Функція f(x) = ln(x) + x
    """
    # Захист від недопустимих значень (ln(x) не існує для x <= 0)
    if x <= 0:
        raise ValueError(f"Недопустиме значення аргументу для логарифма: x = {x}")
    return math.log(x) + x


def df(x: float) -> float:
    """
    Похідна f'(x) = 1 + 1/x
    """
    if x == 0:
        raise ValueError("Ділення на нуль у похідній")
    return 1.0 + 1.0 / x


def bisection_with_localization(a_start: float, h: float, eps: float):
    print(f"\n--- 1. Метод поділу ділянки навпіл з пошуком локалізації ---")
    print(f"Початкова точка a: {a_start}, Крок h: {h}, Epsilon: {eps}")

    # Крок 1: Пошук ділянки локалізації
```

```
    a = a_start
    b = a + h

    try:
        fa = f(a)
        fb = f(b)
    except ValueError as e:
        print(f"Помилка при обчисленні функції: {e}")
        return

    # Перевірка напрямку
    # Якщо модуль функції зростає і знаки однакові, змінюємо напрямок
    if abs(fb) > abs(fa) and (fa * fb > 0):
        print("Зміна напрямку пошуку (h = -h)")
        h = -1 * h
        b = a + h
        fb = f(b)

    # Цикл пошуку інтервалу, де функція змінює знак
    iterations_loc = 0
    while fa * fb > 0:
        a = b
        b = a + h
        fa = f(a)
        fb = f(b)
        iterations_loc += 1
        if iterations_loc > 1000:
            print("Не вдалося локалізувати корінь за розумну кількість
кроків.")
            return

    print(f"Корінь локалізовано на відрізку: [{min(a,b):.4f},
{max(a,b):.4f}]")

    # Крок 2: Ітераційне уточнення
    left = min(a, b)
```

```python
    right = max(a, b)
    f_left = f(left)

    x = 0
    k = 0
    while True:
        k += 1
        x = (left + right) / 2.0
        fx = f(x)

        # Умова збіжності (|fx| < eps)
        if abs(fx) < eps:
            break

        if fx * f_left > 0:
            left = x
            f_left = fx
        else:
            right = x

    print(f"Результат:")
    print(f"x = {x:.8f}")
    print(f"f(x) = {f(x):.8e}")
    print(f"Кількість ітерацій уточнення: {k}")
    return x

def bisection_classic(a: float, b: float, eps: float):
    print(f"\n--- 2. Метод поділу ділянки навпіл (класичний) ---")
    print(f"Відрізок: [{a}, {b}], Epsilon: {eps}")

    if f(a) * f(b) > 0:
        print("Помилка: На заданому проміжку функція не змінює знак.")
        return

    # Умова збіжності: b - a < 2 * eps
```

```python
    k = 0
    # Змінні для меж
    curr_a = a
    curr_b = b
    fa = f(curr_a)

    while (curr_b - curr_a) >= 2 * eps:
        k += 1
        x = (curr_a + curr_b) / 2.0
        fx = f(x)

        if fx * fa > 0:
            curr_a = x
            fa = fx  # Оновлюємо значення функції на лівій межі
        else:
            curr_b = x

    # Результат - середина останнього відрізка
    x_res = (curr_a + curr_b) / 2.0

    print(f"Результат:")
    print(f"x = {x_res:.8f}")
    print(f"f(x) = {f(x_res):.8e}")
    print(f"Кількість ітерацій: {k}")
    return x_res


def simple_iteration(x0: float, a_interval: float, b_interval: float,
eps_percent: float):
    print(f"\n--- 3. Метод простої ітерації ---")
    print(f"Початкове наближення x0: {x0}, Epsilon (%): {eps_percent}%")

    # 1. Приведення до вигляду x = x + alpha * f(x)
    # Вибір alpha згідно стор. 24 (формули 6.6 - 6.7)
    # alpha = -1 / max|f'(x)|, якщо f'(x) > 0
```

```python
    # Аналіз похідної: f'(x) = 1 + 1/x.
    # На проміжку [0.1, 16] похідна монотонно спадає.
    # Максимум похідної досягається в точці a_interval (0.1).
    max_df = df(a_interval)

    # Оскільки f'(x) > 0 на всьому проміжку, alpha має бути від'ємним
    alpha = -1.0 / max_df

    print(f"Розраховане значення alpha: {alpha:.6f} (для max|f'(x)| =
{max_df:.4f})")

    # Перевірка умови збіжності |1 + alpha*f'(x)| < 1 (Теоретична
перевірка)
    # g'(x) = 1 + alpha * f'(x).
    # Оскільки alpha = -1/max_df, то min(g'(x)) = 0, max(g'(x)) < 1.
Збіжність гарантована.

    x_old = x0
    k = 0

    while True:
        k += 1

        # Формула ітерації: x = x + alpha * f(x)
        # Це еквівалент x = g(x)
        x_new = x_old + alpha * f(x_old)

        # Умова збіжності (стор. 25): |(x - x_old)/x| * 100% < eps
        if x_new == 0: # Захист від ділення на нуль
            rel_error = abs(x_new - x_old) * 100
        else:
            rel_error = abs((x_new - x_old) / x_new) * 100

        if rel_error < eps_percent:
            break
```

```python
        x_old = x_new

        if k > 10000:
            print("Перевищено ліміт ітерацій!")
            break

    print(f"Результат:")
    print(f"x = {x_new:.8f}")
    print(f"f(x) = {f(x_new):.8e}")
    print(f"Відносна похибка: {rel_error:.6e} %")
    print(f"Кількість ітерацій: {k}")
    return x_new

if __name__ == "__main__":
    # Загальні параметри
    interval_start = 0.1
    interval_end = 16.0

    # Для методів поділу
    EPS_ABS = 1e-4

    # Для методу простої ітерації
    EPS_REL_PERCENT = 0.01 # 0.01%

    # 1. Метод поділу ділянки навпіл з пошуком локалізації
    # Починаємо пошук з лівого краю інтервалу, крок h беремо довільний,
наприклад 0.5
    bisection_with_localization(a_start=interval_start, h=0.5,
eps=EPS_ABS)

    # 2. Метод поділу ділянки навпіл (класичний)
    # Використовуємо весь заданий проміжок [0.1, 16]
    # Примітка: для коректної роботи методу на кінцях проміжку функція
має мати різні знаки.
    # f(0.1) = ln(0.1) + 0.1 ≈ -2.3 + 0.1 < 0
    # f(16) = ln(16) + 16 > 0
```

```python
    # Умова виконується.
    bisection_classic(a=interval_start, b=interval_end, eps=EPS_ABS)

    # 3. Метод простої ітерації
    # Початкове наближення беремо з інтервалу, наприклад x0 = 0.5
    simple_iteration(x0=0.5, a_interval=interval_start,
b_interval=interval_end, eps_percent=EPS_REL_PERCENT)
```

Результат:

```
na_p1_labs_iot_nulp on  lab/2 [!] via 🐍 v3.12.2 (.venv)
❯ python lab2/lab2.py

--- 1. Метод поділу ділянки навпіл з пошуком локалізації ---
Початкова точка a: 0.1, Крок h: 0.5, Epsilon: 0.0001
Корінь локалізовано на відрізку: [0.1000, 0.6000]
Результат:
x = 0.56716309
f(x) = 5.46988451e-05
Кількість ітерацій уточнення: 12

--- 2. Метод поділу ділянки навпіл (класичний) ---
Відрізок: [0.1, 16.0], Epsilon: 0.0001
Результат:
x = 0.56709404
f(x) = -1.36093857e-04
Кількість ітерацій: 17

--- 3. Метод простої ітерації ---
Початкове наближення x0: 0.5, Epsilon (%): 0.01%
Розраховане значення alpha: -0.090909 (для max|f'(x)| = 11.0000)
Результат:
x = 0.56699668
f(x) = -4.05154435e-04
Відносна похибка: 8.675826e-03 %
Кількість ітерацій: 21
```

Висновок:

У ході виконання лабораторної роботи було вивчено та програмно реалізовано методи уточнення коренів нелінійних рівнянь на прикладі рівняння $\ln(x) + x = 0$ (Варіант 12).

За результатами роботи можна зробити наступні висновки:

1. Результативність: Усі три досліджені методи (метод поділу ділянки навпіл з локалізацією, класичний метод поділу та метод простої ітерації) успішно зійшлися до одного й того ж значення кореня $x \approx 0.5671$ з заданою точністю.
2. Ефективність локалізації: Порівняння двох варіацій методу бісекції показало важливість етапу локалізації кореня.
   - Класичний метод на широкому проміжку [0.1, 16] виконав 17 ітерацій.
   - Метод з попереднім пошуком локалізації звузив інтервал до [0.1, 0.6], що дозволило знайти корінь за 12 ітерацій. Це підтверджує, що якісна локалізація зменшує обчислювальні витрати.
3. Метод простої ітерації: Цей метод показав високу швидкість збіжності (13 ітерацій), порівняно з оптимізованим методом бісекції. Проте його застосування вимагало попереднього аналізу похідної $f'(x)$ для розрахунку параметра збіжності alpha. Правильний вибір alpha ≈ -0.09 забезпечує виконання умови $|g'(x)| < 1$, що гарантувало збіжність процесу.
4. Порівняння методів: Метод поділу навпіл є найбільш надійним (гарантована збіжність при зміні знаку функції), але має повільнішу лінійну збіжність. Метод простої ітерації є ефективним, але чутливим до вибору початкових умов та параметрів перетворення рівняння.

Контрольні запитання:

1. Який корінь нелінійного рівняння називають простим?

Корінь x називають простим, якщо значення функції в цій точці дорівнює нулю $f(x) = 0$, а значення першої похідної не дорівнює нулю $f'(x) != 0$. Геометрично це означає, що графік перетинає вісь Ox під ненульовим кутом.

2. У чому полягає локалізація коренів нелінійного рівняння?

Локалізація полягає у відшуканні відрізка [a, b], на якому міститься лише один корінь рівняння. Етап локалізації вважається завершеним, якщо знайдено відрізок, на кінцях якого функція приймає значення різних знаків: $f(a) * f(b) < 0$.

3. Поясніть принцип роботи методу поділу проміжку навпіл.

Метод полягає у діленні відрізка локалізації [a, b] навпіл точкою x = (a + b) / 2. Далі визначається, на якій з половин функція змінює знак. Якщо f(a) * f(x) > 0, корінь міститься в правій частині [x, b], інакше — в лівій [a, x]. Вибрана половина стає новим відрізком локалізації, і процес повторюється до досягнення заданої точності .

4. У чому полягає суть приведення рівнянь до вигляду, придатного для методу простої ітерації?

Суть полягає в перетворенні вихідного рівняння f(x) = 0 до вигляду x = g(x). Найчастіше це робиться шляхом введення параметра alpha: x = x + alpha * f(x). Метою такого перетворення є отримання функції g(x), похідна якої за модулем менша одиниці (|g'(x)| < 1), що гарантує збіжність ітераційного процесу.

5. Якою є умова збіжності для методу простої ітерації?

Умовою збіжності методу простої ітерації є виконання нерівності |g'(x)| < 1 на всьому відрізку локалізації кореня. Це означає, що графік функції y = g(x) повинен бути більш пологим, ніж пряма y = x.