Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 3

з дисципліни

​*Чисельні методи*

на тему:

“Методи розв’язування нелінійних рівнянь

»

Виконав: студент гр. ОІ-25

Коновалюк О.В.

Прийняв:

Асистент кафудри АСУ

Зварич В.І.

Львів – 2025

**Лабораторна робота №3**

**Варіант 12**

**Методи розв’язування нелінійних рівнянь**

Мета роботи: вивчити та засвоїти ітераційні методи розв’язування

алгебраїчних і трансцендентних рівнянь.

Завдання :

1. Знайти аналітично один з коренів рівняння f(x)=0, використовуючи методи

• половинного ділення,

• дотичних,

• хорд

• простої ітерації

з точністю  = 0,00001 .

2. Вивести кількість ітерацій, необхідних для досягнення точності для кожного з

методів.

3. Провести дослідження кількості ітерацій для кожного з методів при зміні

точності від

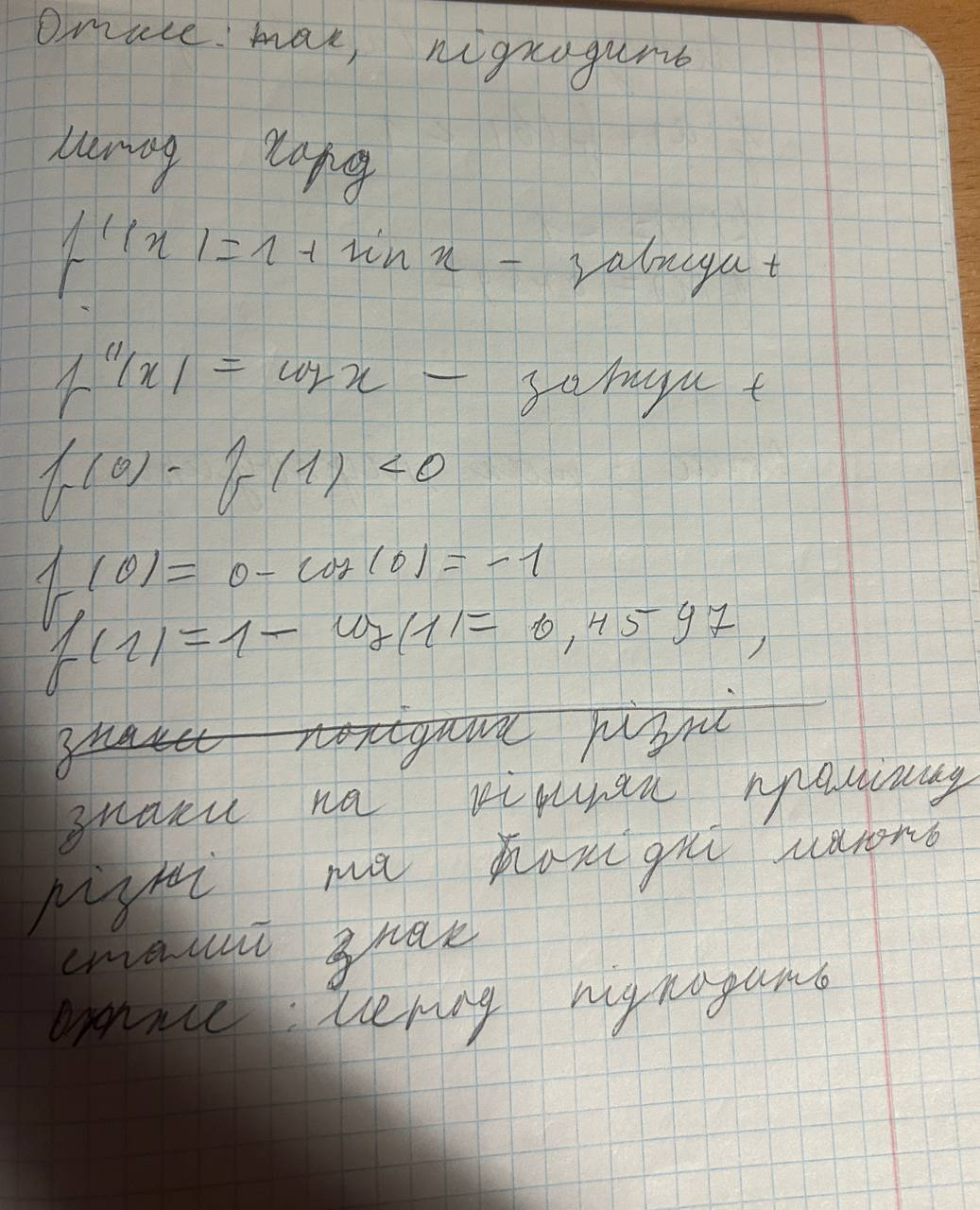
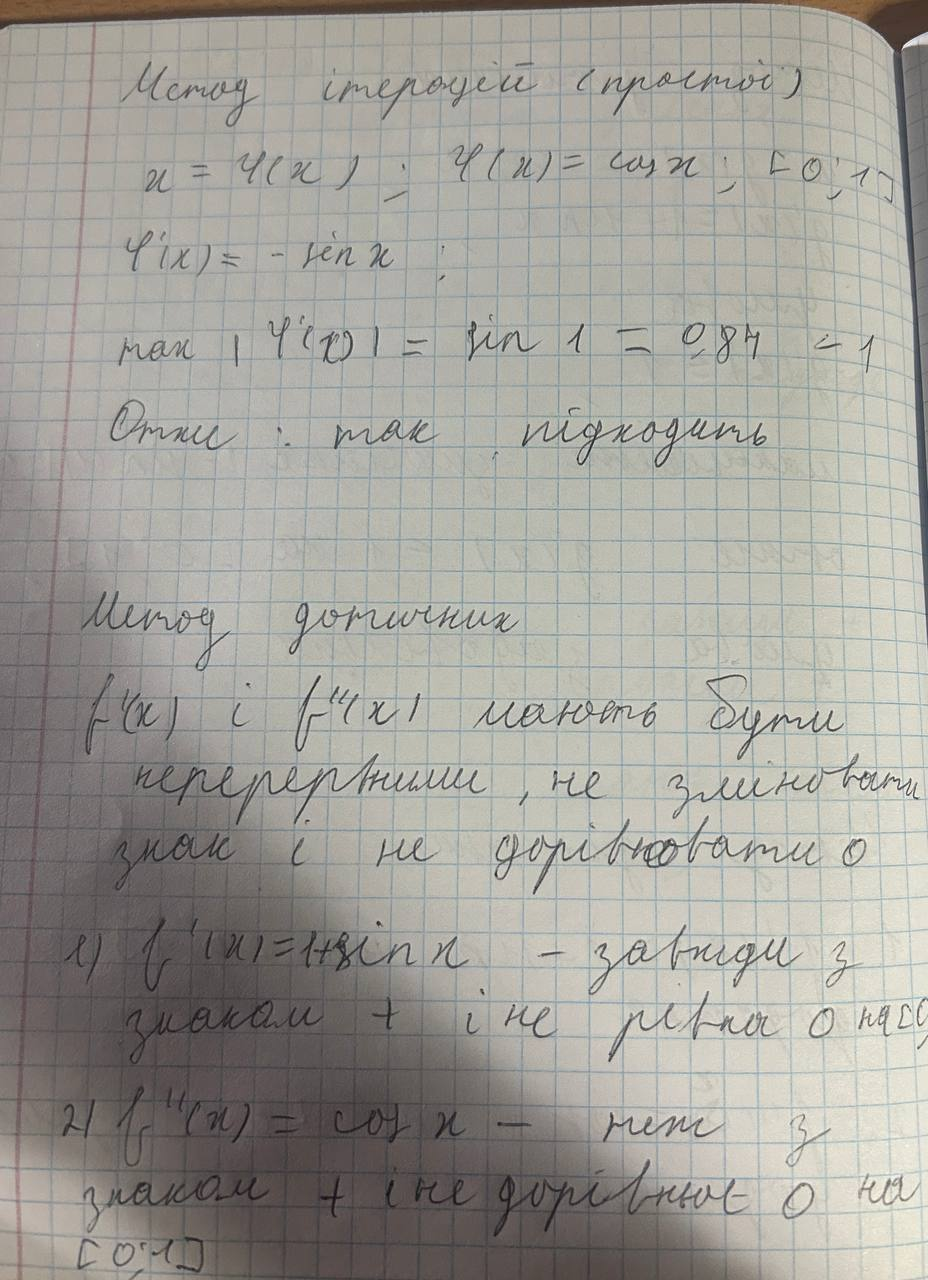
до

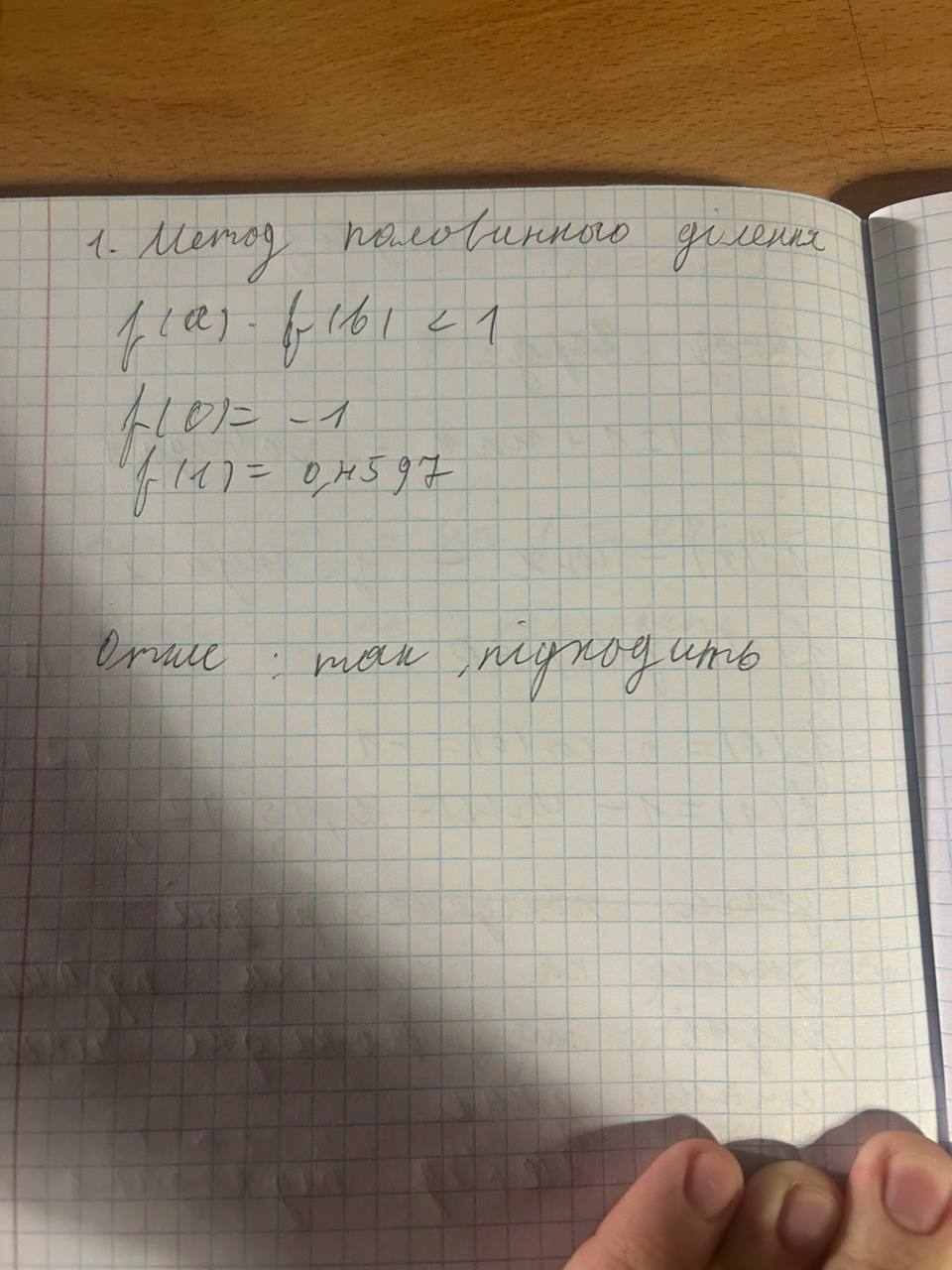
.

4. У звіті подати відповідні таблиці та графіки.

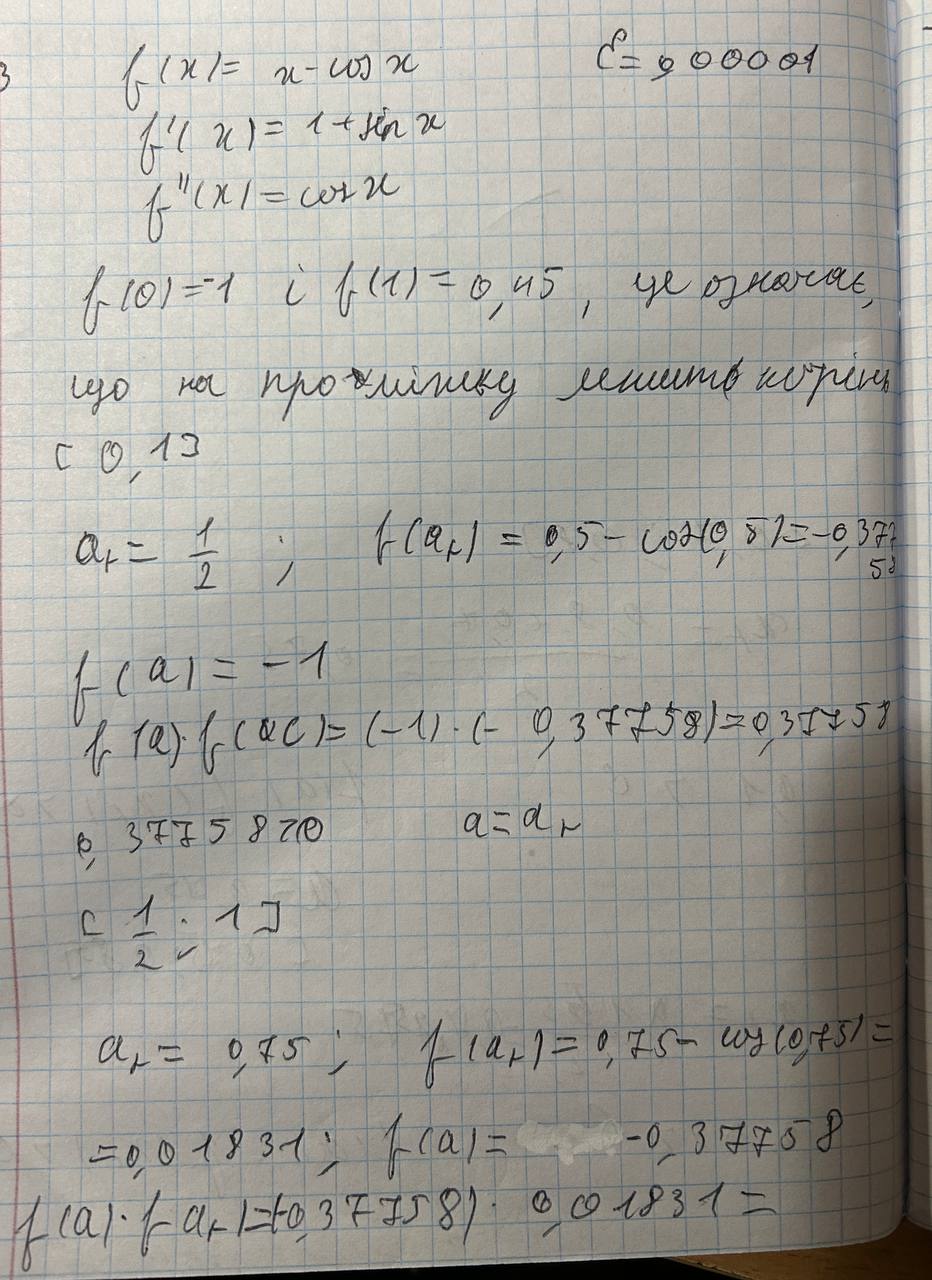
{6A64D8B4-3D51-4947-92E0-7EB05A9E266B}

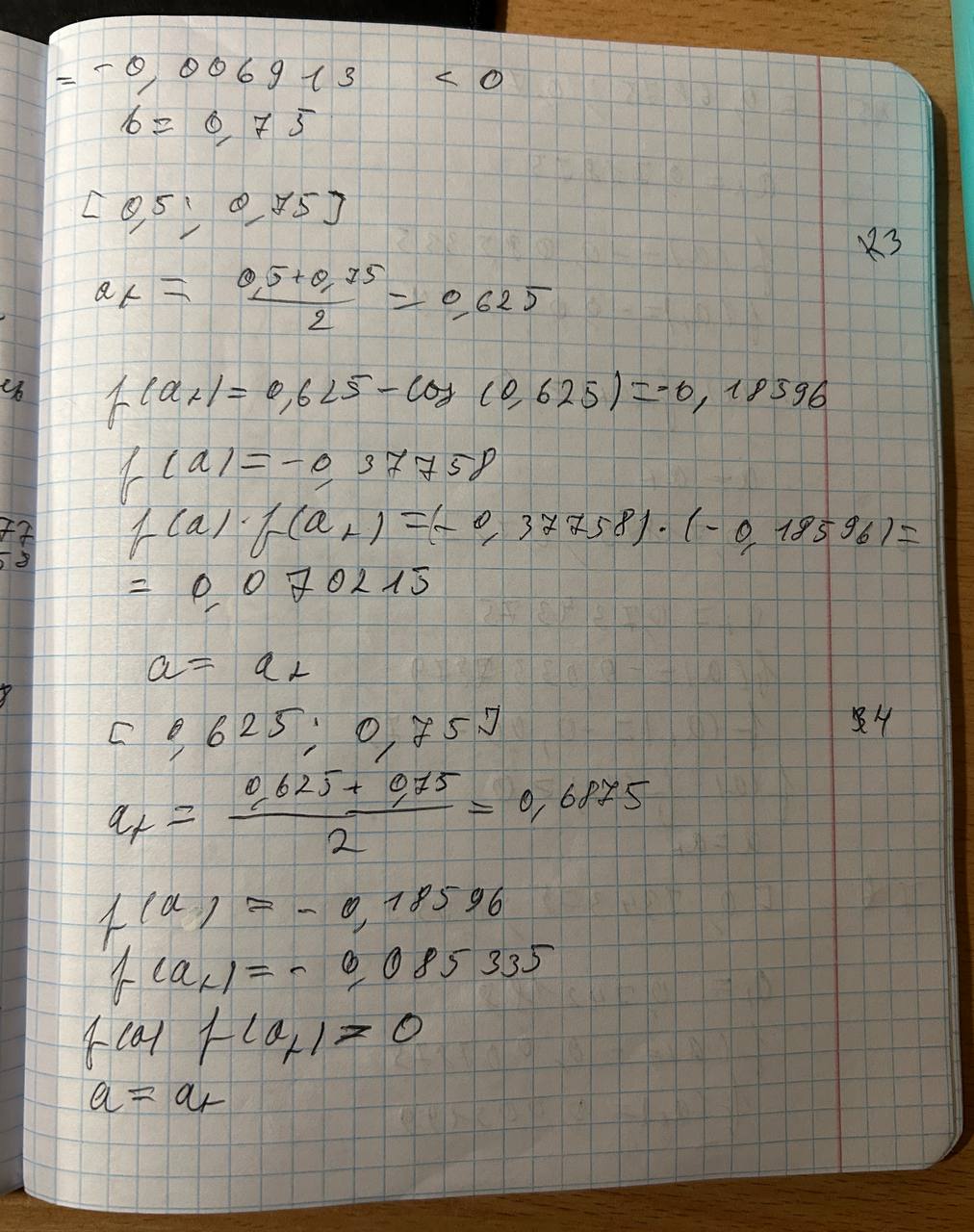
Перевірка на збіжність :

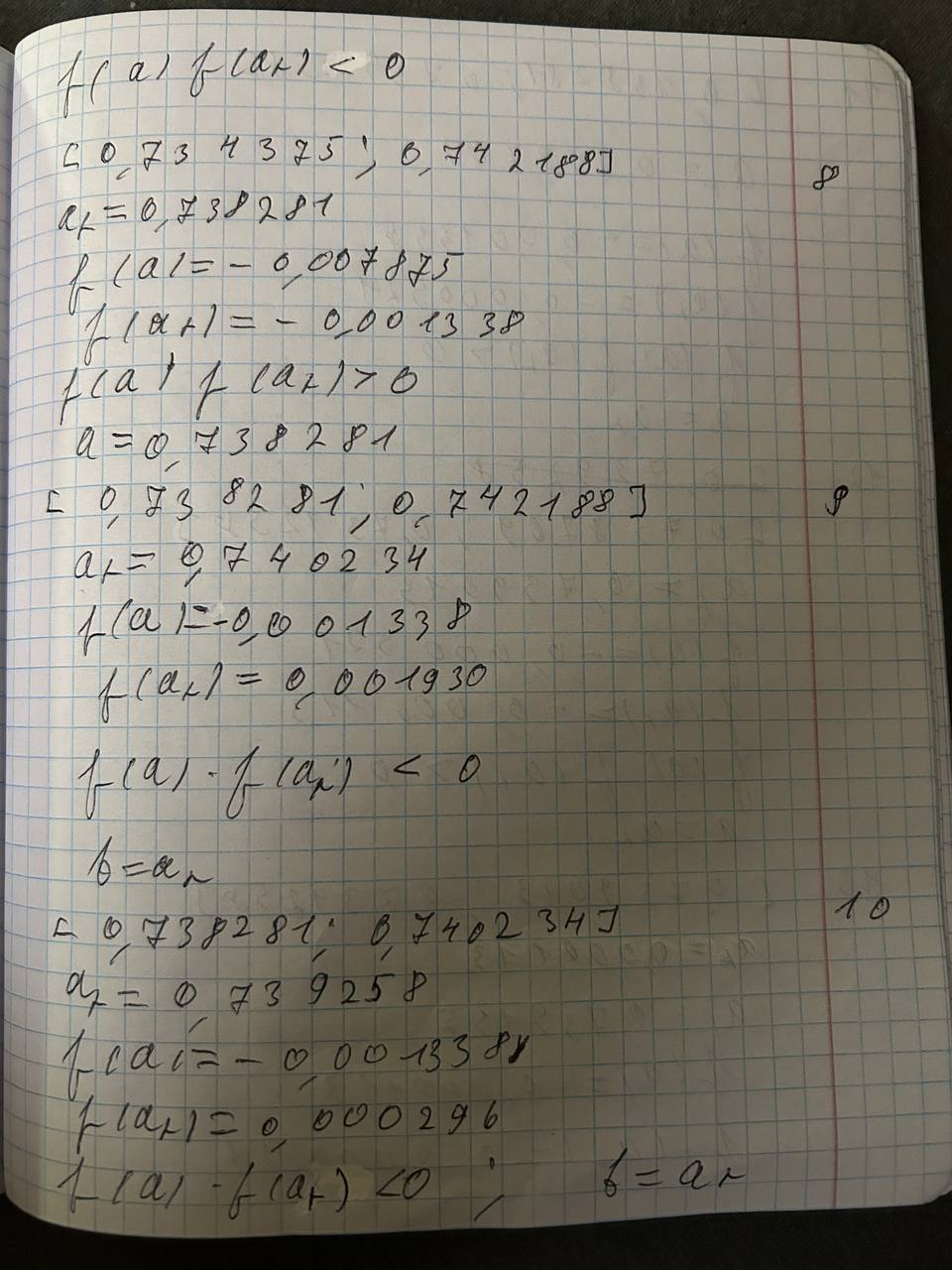


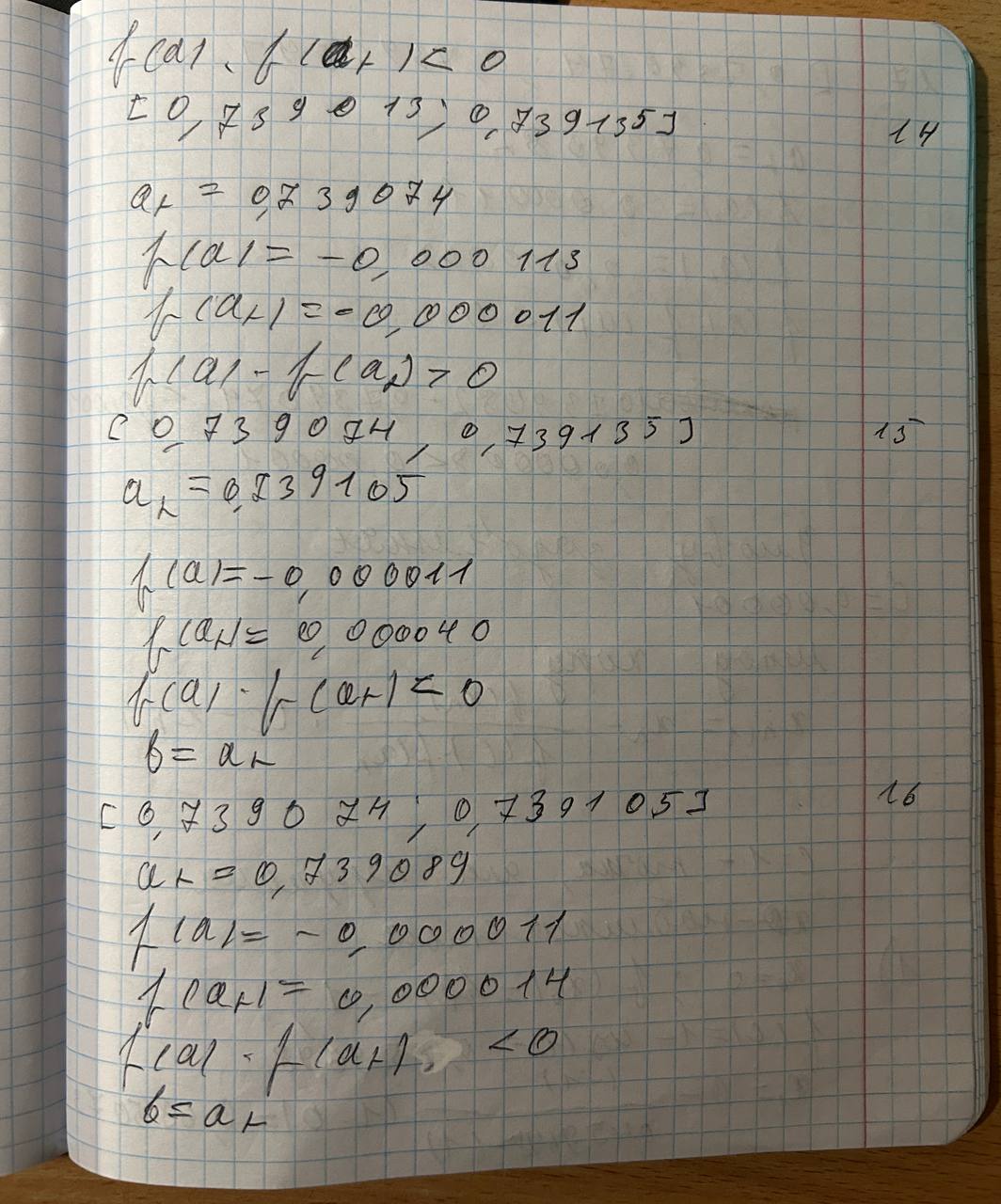
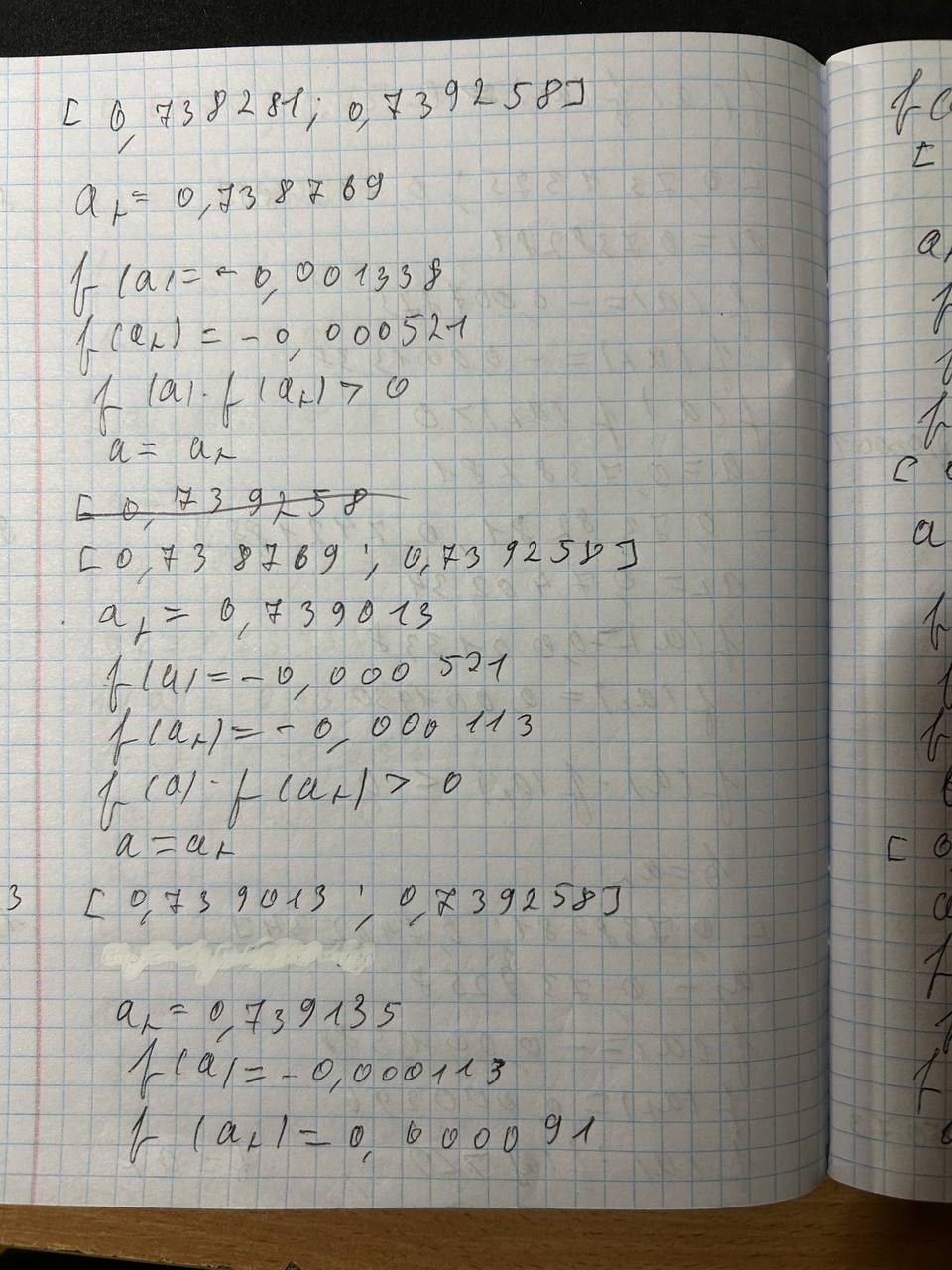


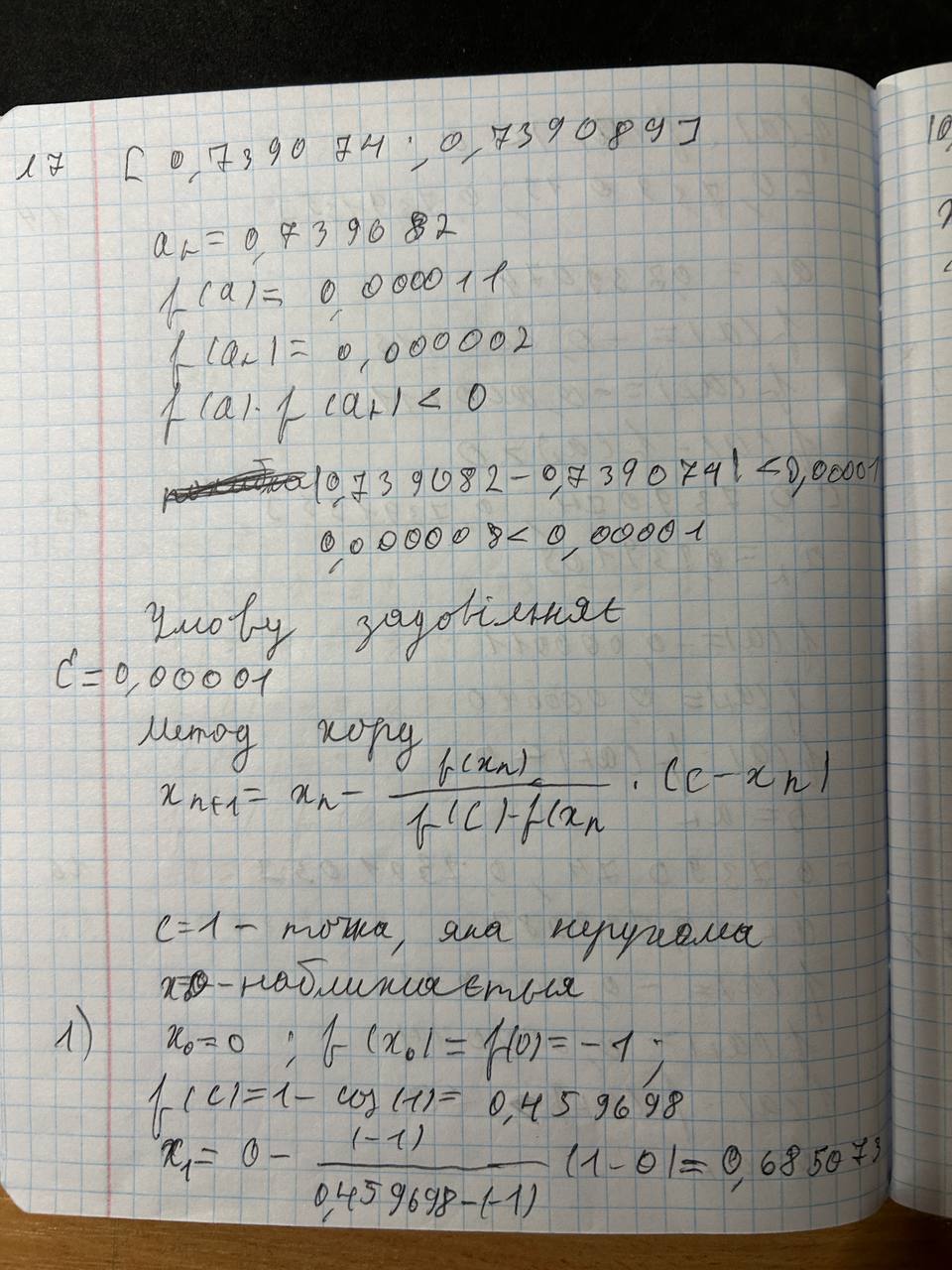
Метод половинного ділення:



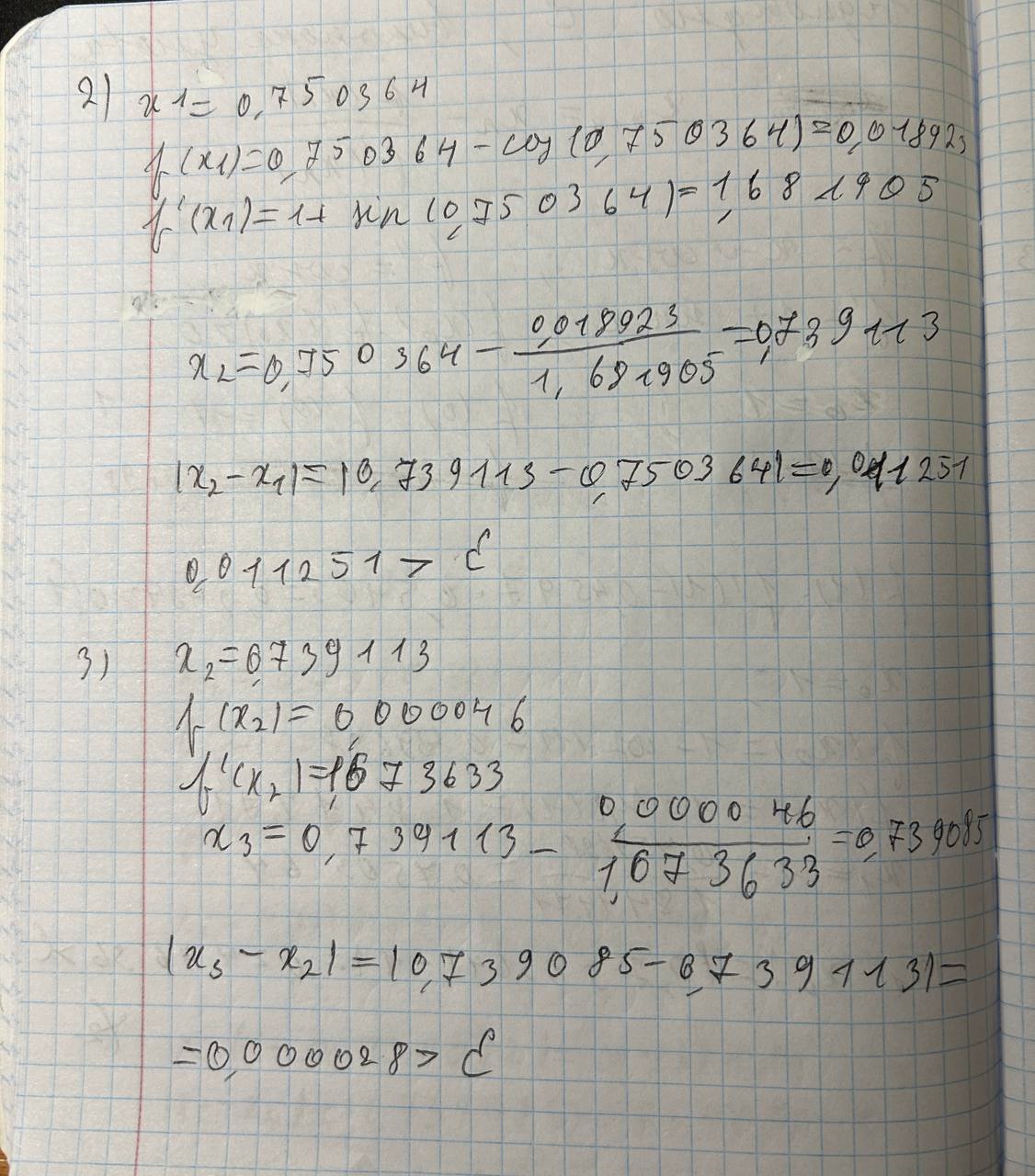
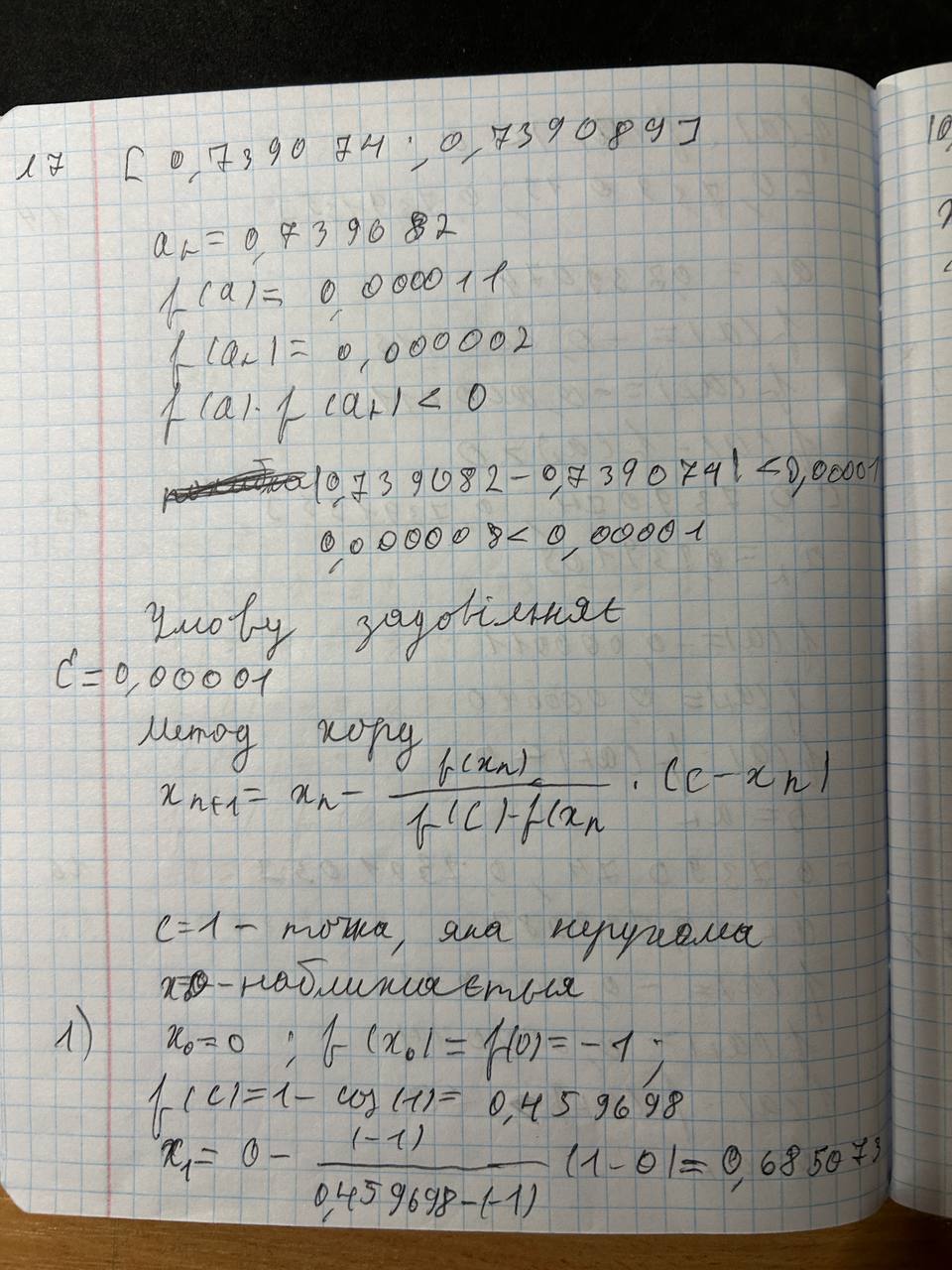


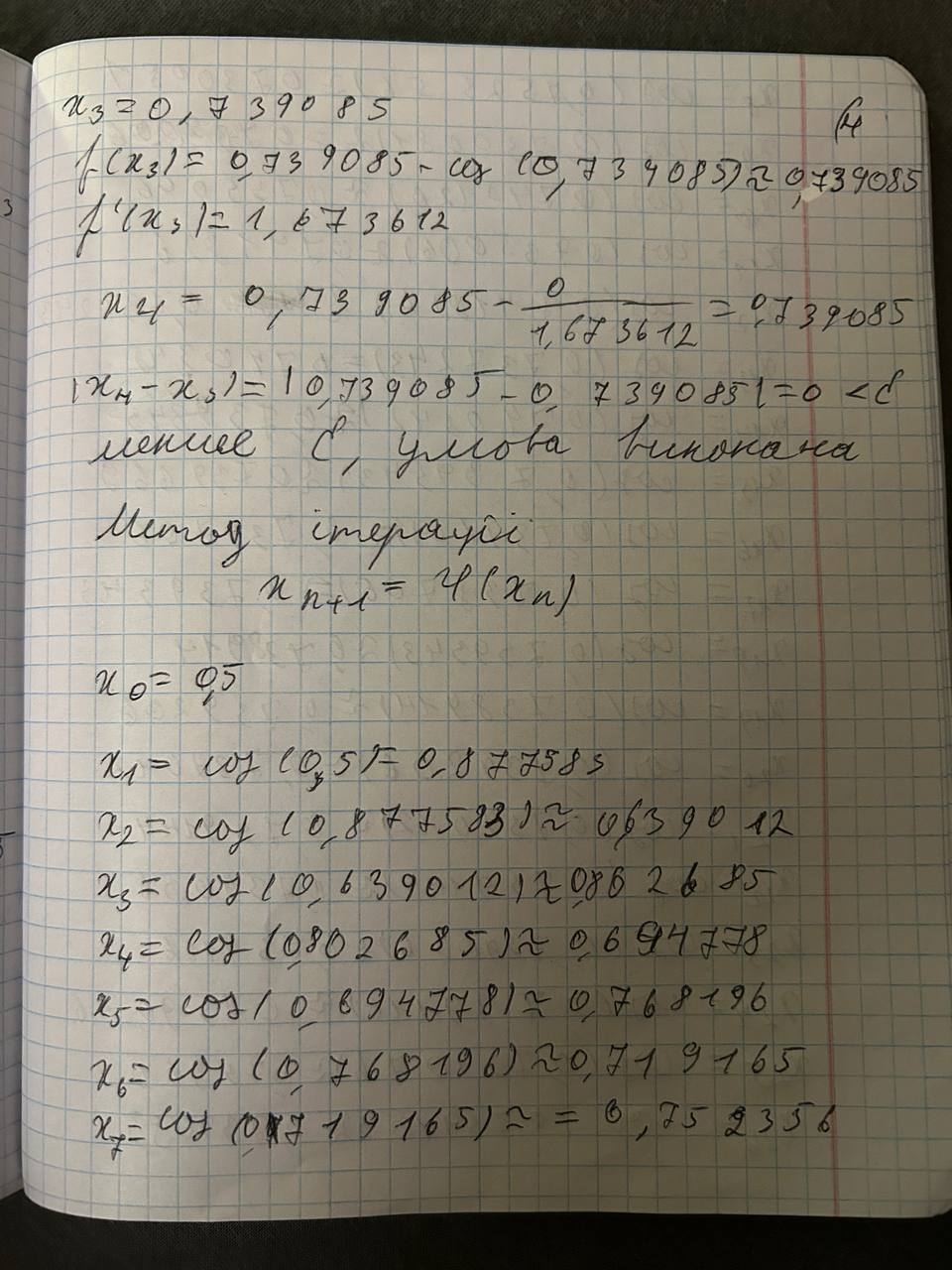




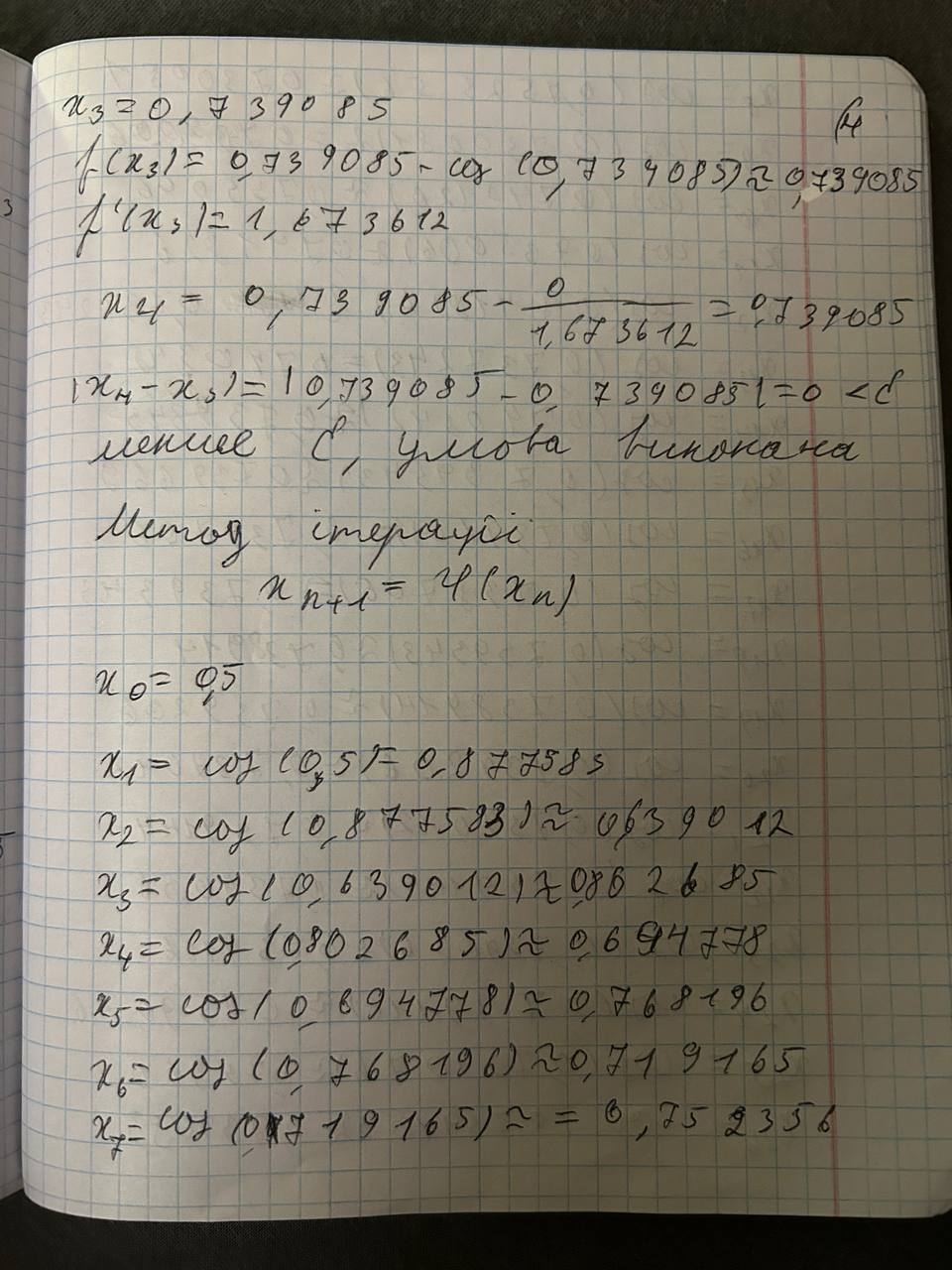


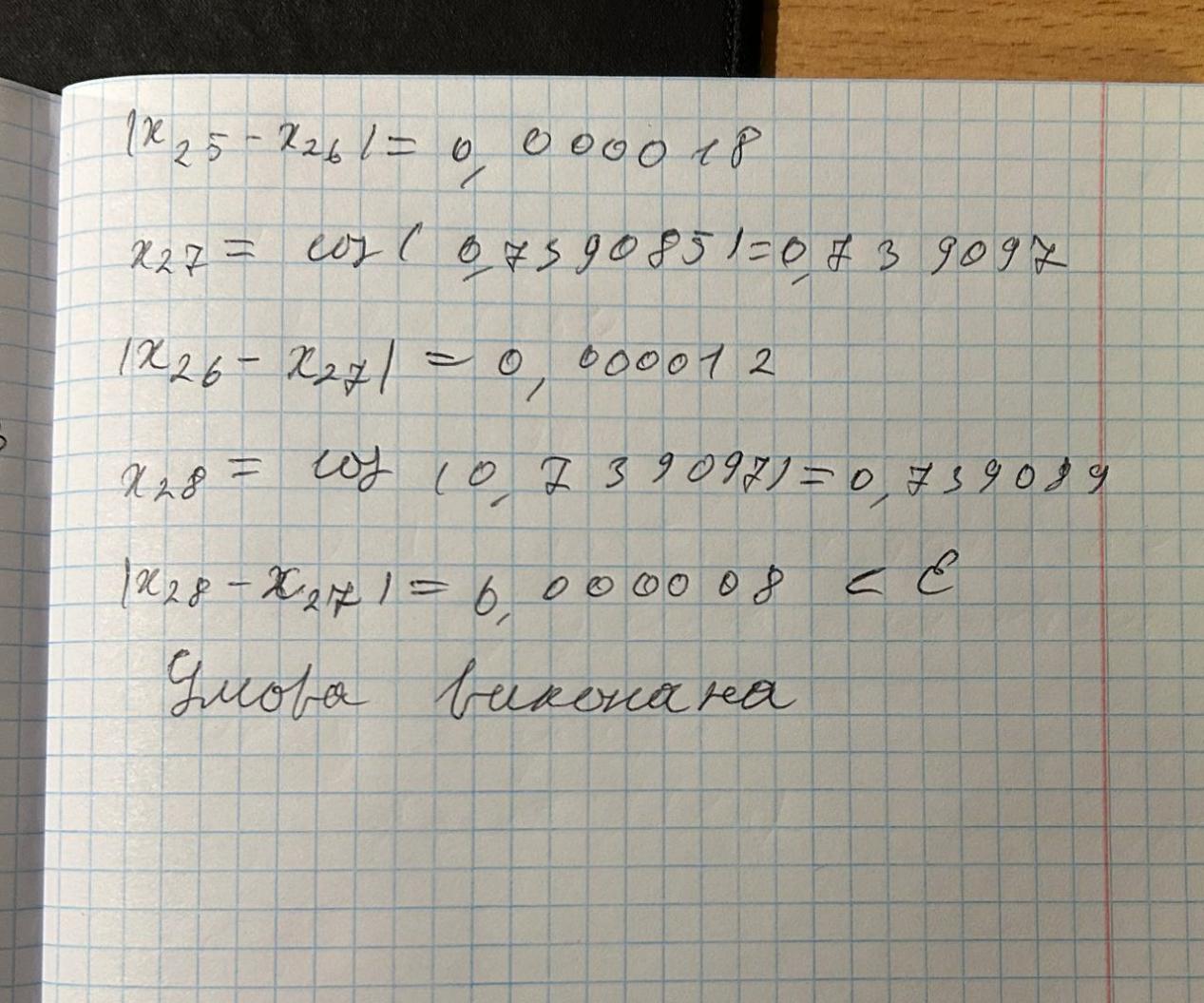
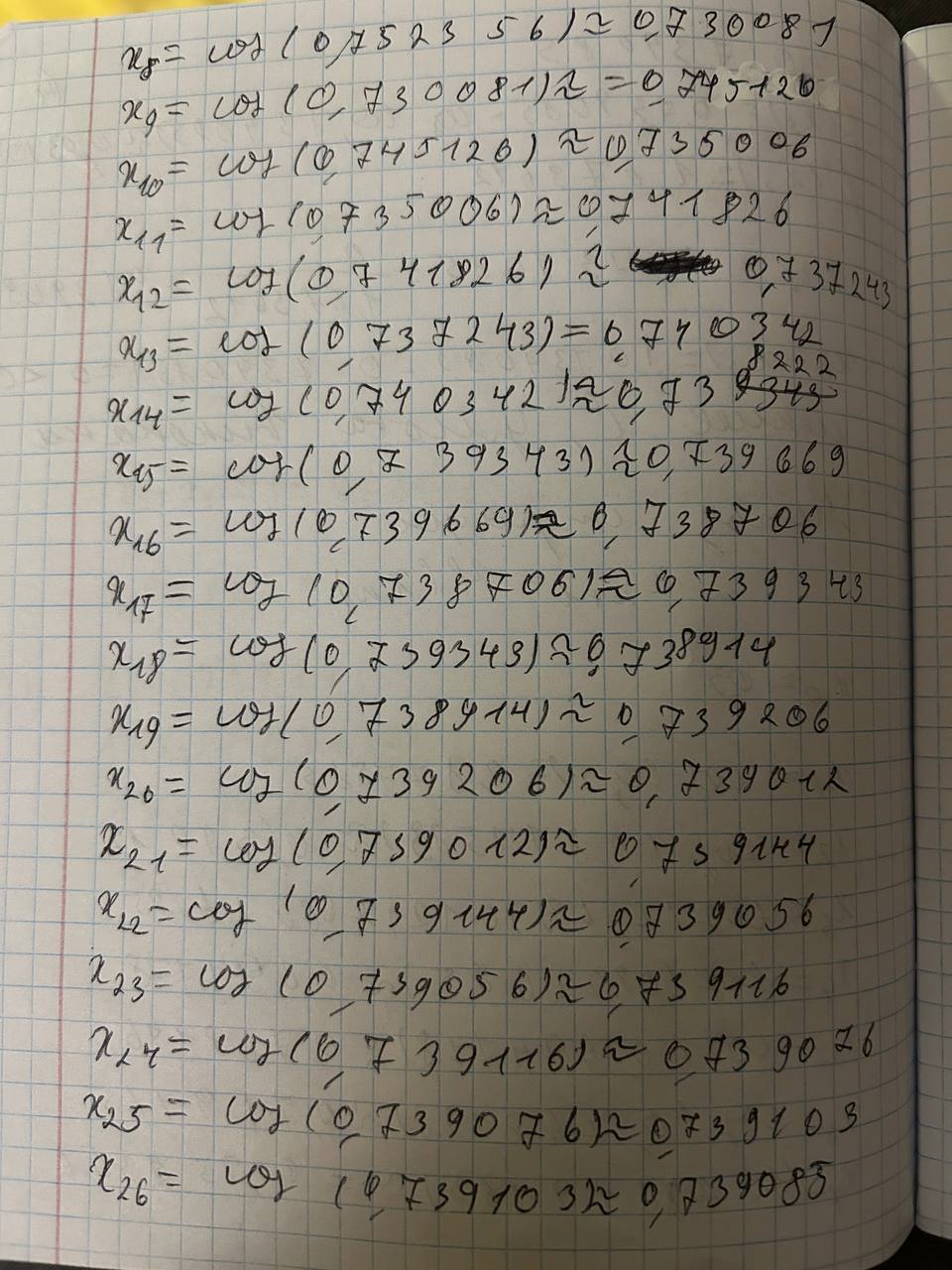
Метод хорд



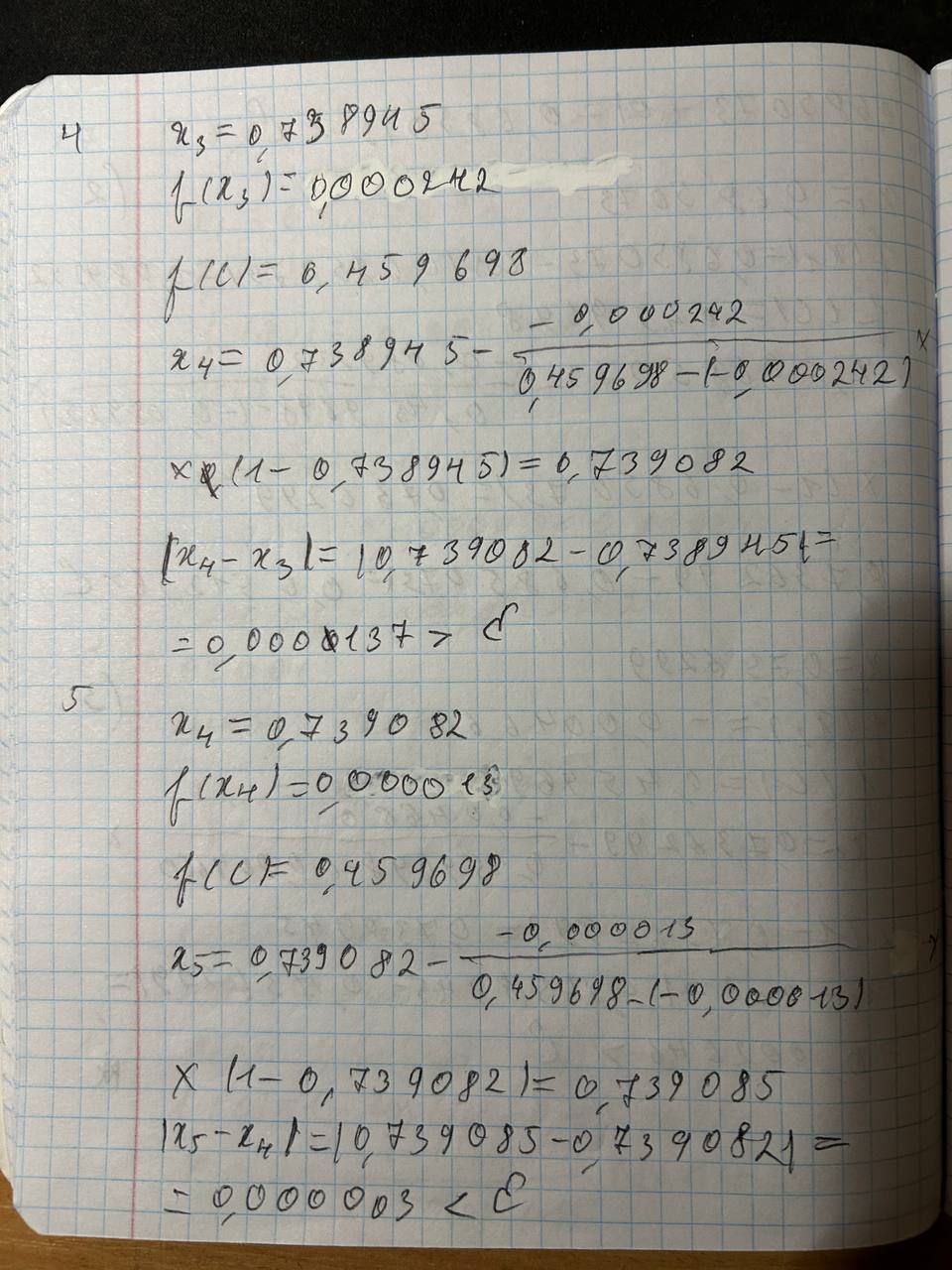
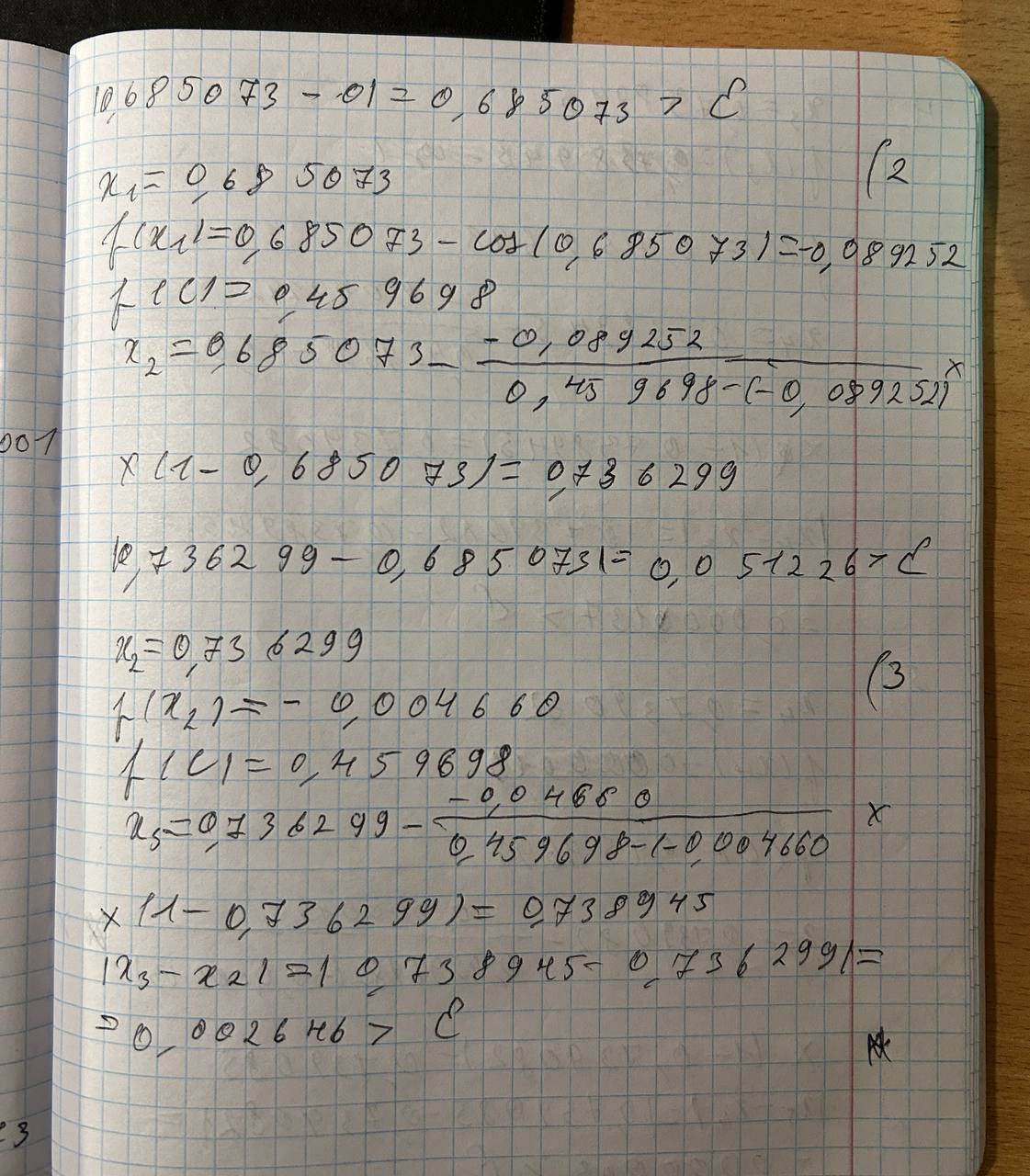
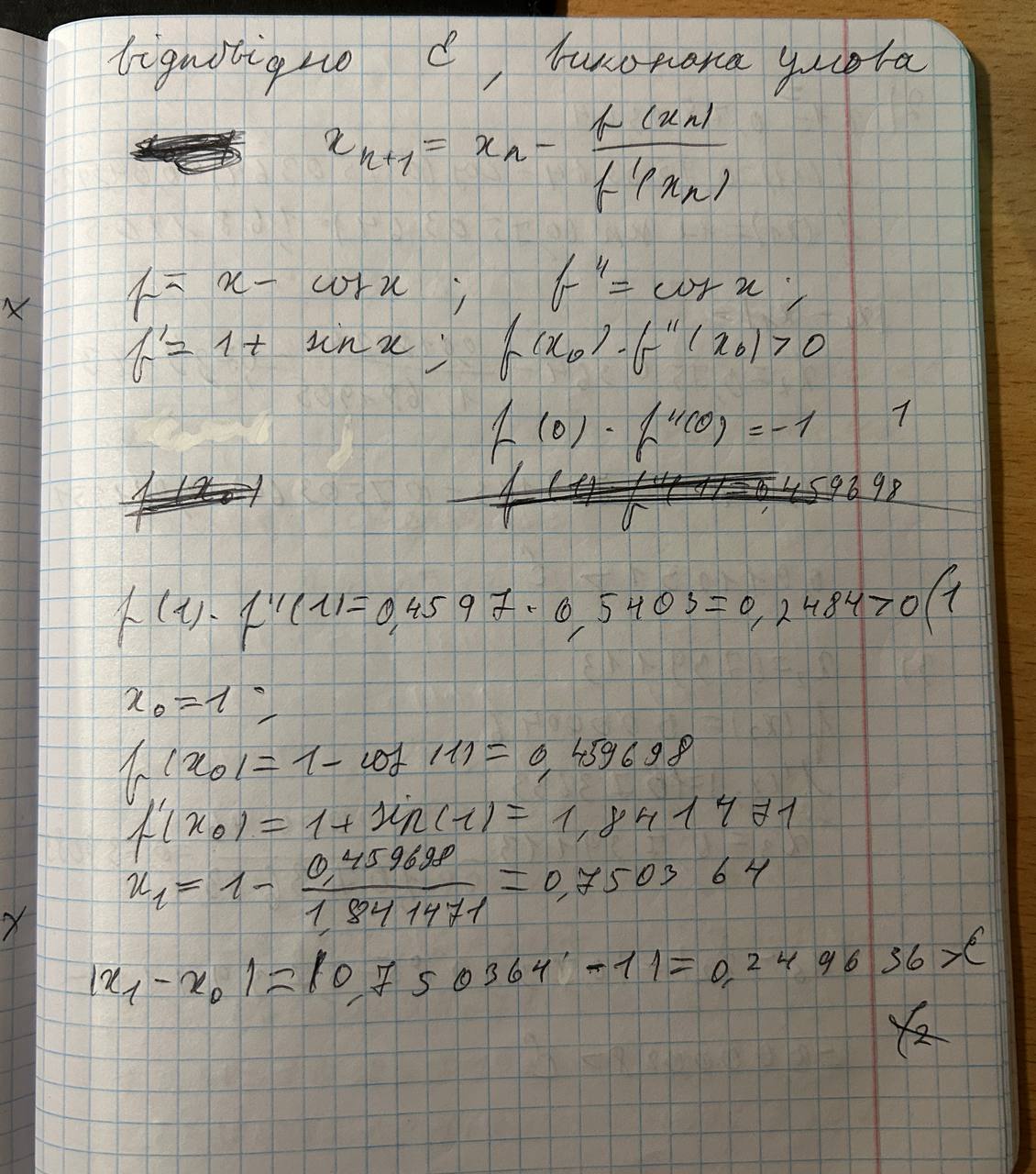


Метод ітерації:





Метод дотичних:

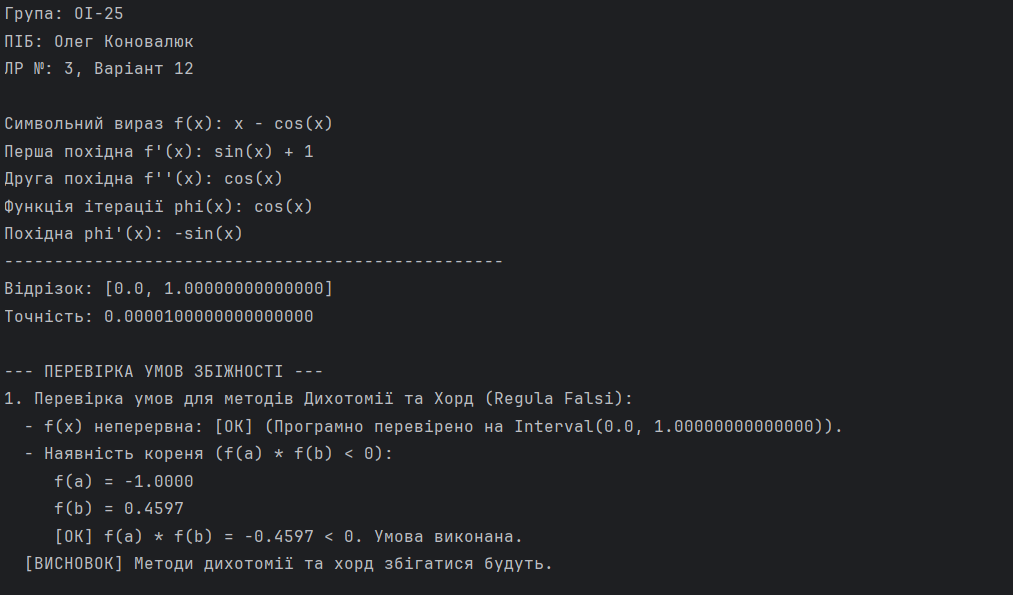


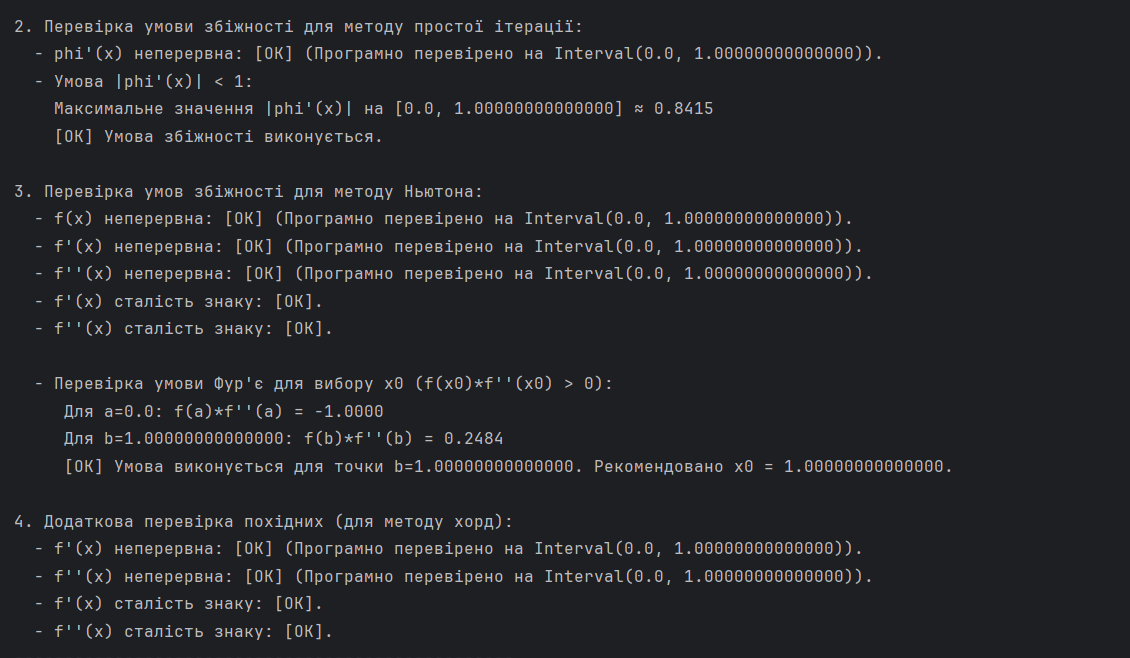
{2BDC54B6-E7E9-4493-9847-F19977893D76}

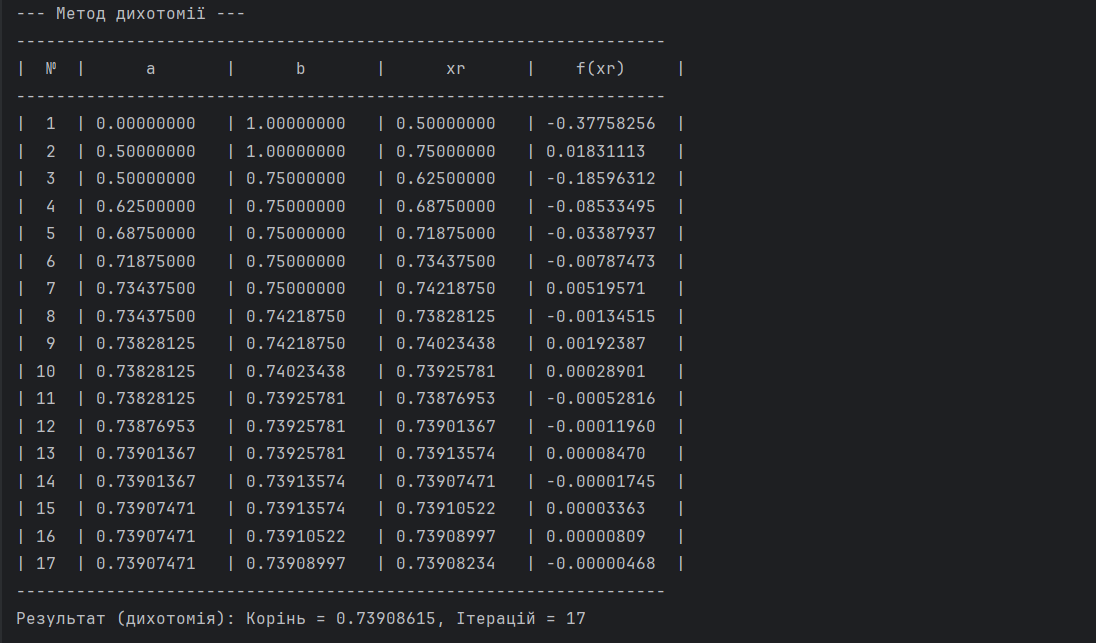
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Точність | Метод дихотомії | Метод хорд | Метод Ньютона | Метод простої ітерації |
| 0.1 | 4 | 2 | 2 | 5 |
| 0.01 | 3 | 3 | 3 | 11 |
| 0.001 | 10 | 4 | 3 | 16 |
| 0.0001 | 14 | 5 | 3 | 22 |
| 0.00001 | 17 | 5 | 4 | 28 |
| 0.000001 | 20 | 6 | 4 | 34 |
| 0.0000001 | 24 | 7 | 4 | 40 |
| 0.00000001 | 27 | 8 | 4 | 45 |
| 0.000000001 | 30 | 8 | 4 | 51 |
| 0.0000000001 | 34 | 9 | 5 | 57 |

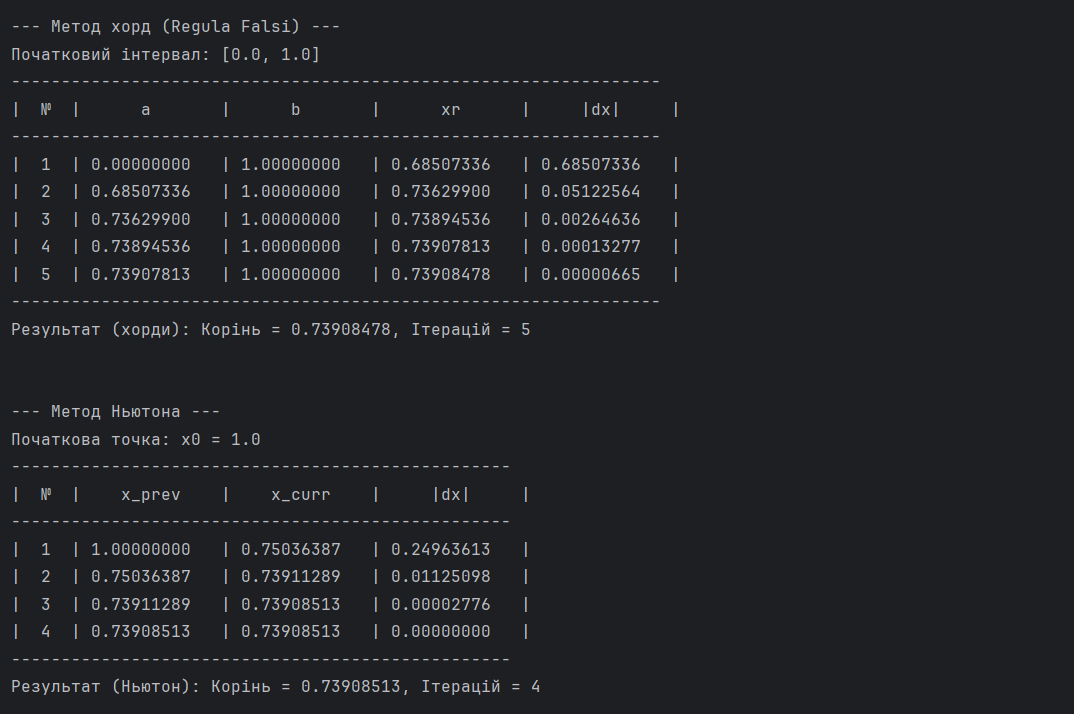
Код програми

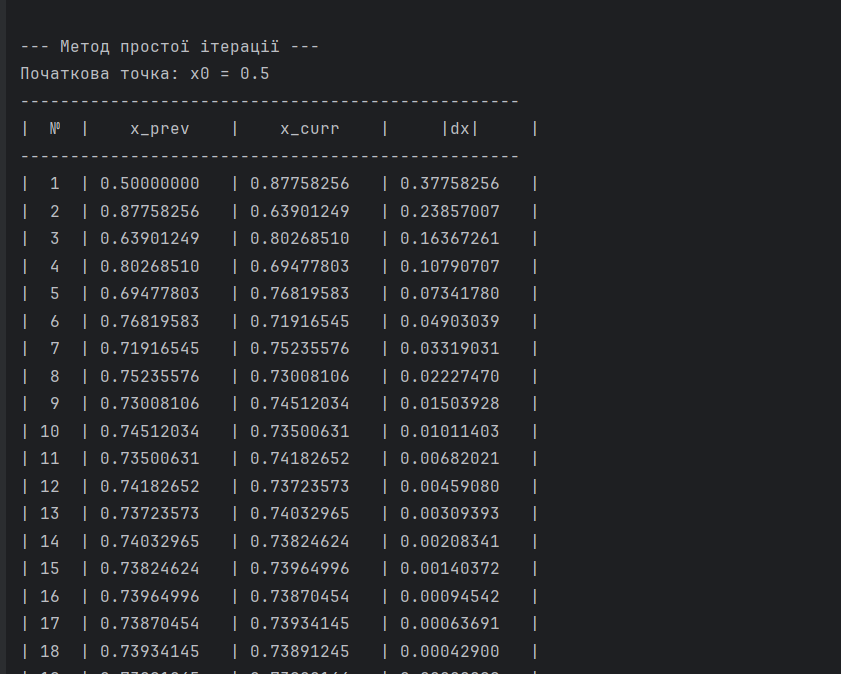
import sympy as sp  
import math  
from sympy.sets import Reals  
from sympy.solvers.solveset import continuous\_domain  
  
# --- 1. Символьне визначення та автоматичне диференціювання ---  
  
# Створюємо символьну змінну x  
x\_sym = sp.Symbol('x')  
  
# Визначаємо f(x) та phi(x) як символьні вирази  
f\_sym = x\_sym - sp.cos(x\_sym)  
phi\_sym = sp.cos(x\_sym)  
  
# Автоматично обчислюємо похідні  
f\_prime\_sym = sp.diff(f\_sym, x\_sym)  
f\_double\_prime\_sym = sp.diff(f\_prime\_sym, x\_sym)  
phi\_prime\_sym = sp.diff(phi\_sym, x\_sym)  
  
  
# --- 2. Створюємо функції-обгортки для обчислень (заміна lambdify) ---  
  
def f(x\_val):  
 """Обчислює f(x) шляхом підстановки значення у символьний вираз."""  
 return f\_sym.subs(x\_sym, x\_val).evalf()  
  
  
def f\_prime(x\_val):  
 """Обчислює f'(x) шляхом підстановки."""  
 return f\_prime\_sym.subs(x\_sym, x\_val).evalf()  
  
  
def f\_double\_prime(x\_val):  
 """Обчислює f''(x) шляхом підстановки."""  
 return f\_double\_prime\_sym.subs(x\_sym, x\_val).evalf()  
  
  
def phi(x\_val):  
 """Обчислює phi(x) шляхом підстановки."""  
 return phi\_sym.subs(x\_sym, x\_val).evalf()  
  
  
def phi\_prime(x\_val):  
 """Обчислює phi'(x) шляхом підстановки."""  
 return phi\_prime\_sym.subs(x\_sym, x\_val).evalf()  
  
  
# --- 3. Функції перевірки умов збіжності ---  
  
def check\_continuity(func\_sym, var\_sym, interval\_sym, func\_name):  
 """  
 Програмно перевіряє, чи є символьна функція неперервною на заданому інтервалі.  
 """  
 try:  
 domain = continuous\_domain(func\_sym, var\_sym, Reals)  
 is\_cont = interval\_sym.is\_subset(domain)  
  
 if is\_cont:  
 print(f" - {func\_name} неперервна: [OK] (Програмно перевірено на {interval\_sym}).")  
 return True  
 else:  
 print(f" - {func\_name} неперервна: [ПОМИЛКА] Функція має розриви на {interval\_sym}.")  
 print(f" Область неперервності: {domain}")  
 return False  
 except Exception as e:  
 print(f" - {func\_name} неперервна: [УВАГА] Не вдалося визначити неперервність: {e}")  
 return False  
  
  
def check\_bisection\_and\_chord\_conditions(f\_func, f\_sym, var\_sym, interval\_sym):  
 print("1. Перевірка умов для методів Дихотомії та Хорд (Regula Falsi):")  
  
 if not check\_continuity(f\_sym, var\_sym, interval\_sym, "f(x)"):  
 return False  
  
 a = interval\_sym.start  
 b = interval\_sym.end  
 fa = f\_func(a)  
 fb = f\_func(b)  
 print(f" - Наявність кореня (f(a) \* f(b) < 0):")  
 print(f" f(a) = {fa:.4f}")  
 print(f" f(b) = {fb:.4f}")  
  
 if fa \* fb >= 0:  
 print(f" [ПОМИЛКА] f(a) \* f(b) = {fa \* fb:.4f} >= 0. Умова не виконана.")  
 return False  
  
 print(f" [OK] f(a) \* f(b) = {fa \* fb:.4f} < 0. Умова виконана.")  
 print(" [ВИСНОВОК] Методи дихотомії та хорд збігатися будуть.")  
 return True  
  
  
def check\_simple\_iteration\_convergence(phi\_prime\_func, phi\_prime\_sym, var\_sym, interval\_sym):  
 print("\n2. Перевірка умови збіжності для методу простої ітерації:")  
  
 if not check\_continuity(phi\_prime\_sym, var\_sym, interval\_sym, "phi'(x)"):  
 return False  
  
 print(f" - Умова |phi'(x)| < 1:")  
 a = interval\_sym.start  
 b = interval\_sym.end  
 test\_points = [a, (a + b) / 2, b]  
 max\_derivative = max(abs(phi\_prime\_func(x)) for x in test\_points)  
 print(f" Максимальне значення |phi'(x)| на [{a}, {b}] ≈ {max\_derivative:.4f}")  
 if max\_derivative < 1:  
 print(" [OK] Умова збіжності виконується.")  
 return True  
 else:  
 print(" [ПОМИЛКА] Умова збіжності не виконується.")  
 return False  
  
  
def check\_newton\_convergence(f\_func, f\_prime\_func, f\_double\_prime\_func,  
 f\_sym, f\_prime\_sym, f\_double\_prime\_sym, var\_sym, interval\_sym):  
 print("\n3. Перевірка умов збіжності для методу Ньютона:")  
  
 a = interval\_sym.start  
 b = interval\_sym.end  
  
 cont\_ok = check\_continuity(f\_sym, var\_sym, interval\_sym, "f(x)")  
 cont\_p\_ok = check\_continuity(f\_prime\_sym, var\_sym, interval\_sym, "f'(x)")  
 cont\_pp\_ok = check\_continuity(f\_double\_prime\_sym, var\_sym, interval\_sym, "f''(x)")  
 if not (cont\_ok and cont\_p\_ok and cont\_pp\_ok):  
 print(" [ПОМИЛКА] Порушено базові умови неперервності.")  
 return False, None  
  
 test\_points = [a, (a + b) / 2, b]  
 f\_prime\_vals = [f\_prime\_func(x) for x in test\_points]  
 f\_double\_prime\_vals = [f\_double\_prime\_func(x) for x in test\_points]  
  
 if any(v == 0 for v in f\_prime\_vals) or f\_prime\_vals[0] \* f\_prime\_vals[-1] < 0:  
 print(" - f'(x) сталість знаку: [ПОМИЛКА] f'(x) дорівнює 0 або змінює знак.")  
 return False, None  
 print(f" - f'(x) сталість знаку: [OK].")  
  
 if any(v == 0 for v in f\_double\_prime\_vals) or f\_double\_prime\_vals[0] \* f\_double\_prime\_vals[-1] < 0:  
 print(" - f''(x) сталість знаку: [ПОМИЛКА] f''(x) дорівнює 0 або змінює знак.")  
 return False, None  
 print(f" - f''(x) сталість знаку: [OK].")  
  
 print(f"\n - Перевірка умови Фур'є для вибору x0 (f(x0)\*f''(x0) > 0):")  
 check\_a = f\_func(a) \* f\_double\_prime\_vals[0]  
 check\_b = f\_func(b) \* f\_double\_prime\_vals[-1]  
 print(f" Для a={a}: f(a)\*f''(a) = {check\_a:.4f}")  
 print(f" Для b={b}: f(b)\*f''(b) = {check\_b:.4f}")  
  
 if check\_b > 0:  
 print(f" [OK] Умова виконується для точки b={b}. Рекомендовано x0 = {b}.")  
 return True, b  
 elif check\_a > 0:  
 print(f" [OK] Умова виконується для точки a={a}. Рекомендовано x0 = {a}.")  
 return True, a  
 else:  
 print(" [ПОМИЛКА] Умова Фур'є не виконується на кінцях відрізка.")  
 return False, None  
  
  
# <-- НОВЕ: Функція для перевірки похідних для методу хорд (тільки виводить інформацію) -->  
def check\_chord\_derivatives\_advisory(f\_prime\_func, f\_double\_prime\_func,  
 f\_prime\_sym, f\_double\_prime\_sym, var\_sym, interval\_sym):  
 print("\n4. Додаткова перевірка похідних (для методу хорд):")  
  
 a = interval\_sym.start  
 b = interval\_sym.end  
  
 # Перевіряємо неперервність похідних  
 cont\_p\_ok = check\_continuity(f\_prime\_sym, var\_sym, interval\_sym, "f'(x)")  
 cont\_pp\_ok = check\_continuity(f\_double\_prime\_sym, var\_sym, interval\_sym, "f''(x)")  
 if not (cont\_p\_ok and cont\_pp\_ok):  
 print(" [УВАГА] Порушено неперервність похідних.")  
 return # Не зупиняємо, просто інформуємо  
  
 # Перевірка f'(x) та f''(x) на сталість знаку  
 test\_points = [a, (a + b) / 2, b]  
 f\_prime\_vals = [f\_prime\_func(x) for x in test\_points]  
 f\_double\_prime\_vals = [f\_double\_prime\_func(x) for x in test\_points]  
  
 if any(v == 0 for v in f\_prime\_vals) or f\_prime\_vals[0] \* f\_prime\_vals[-1] < 0:  
 print(" - f'(x) сталість знаку: [УВАГА] f'(x) дорівнює 0 або змінює знак.")  
 else:  
 print(f" - f'(x) сталість знаку: [OK].")  
  
 if any(v == 0 for v in f\_double\_prime\_vals) or f\_double\_prime\_vals[0] \* f\_double\_prime\_vals[-1] < 0:  
 print(" - f''(x) сталість знаку: [УВАГА] f''(x) дорівнює 0 або змінює знак.")  
 else:  
 print(f" - f''(x) сталість знаку: [OK].")  
  
  
# --- 4. Реалізація числових методів ---  
  
def bisection\_method(f\_func, a, b, eps):  
 print("\n--- Метод дихотомії ---")  
 print("-" \* 65)  
 print(f"| {'№':^3} | {'a':^12} | {'b':^12} | {'xr':^12} | {'f(xr)':^12} |")  
 print("-" \* 65)  
 iters = 0  
 while (b - a) > eps:  
 iters += 1  
 xr = (a + b) / 2  
 fxr = f\_func(xr)  
 print(f"| {iters:^3} | {a:<12.8f} | {b:<12.8f} | {xr:<12.8f} | {fxr:<12.8f} |")  
 if fxr == 0: break  
 if f\_func(a) \* fxr < 0:  
 b = xr  
 else:  
 a = xr  
 root = (a + b) / 2  
 print("-" \* 65)  
 return root, iters  
  
  
def newton\_method(f\_func, f\_prime\_func, x0, eps):  
 print("\n--- Метод Ньютона ---")  
 print(f"Початкова точка: x0 = {x0:.1f}")  
 print("-" \* 50)  
 print(f"| {'№':^3} | {'x\_prev':^12} | {'x\_curr':^12} | {'|dx|':^12} |")  
 print("-" \* 50)  
 iters = 0  
 x\_prev = x0  
 while True:  
 iters += 1  
 f\_val = f\_func(x\_prev)  
 f\_prime\_val = f\_prime\_func(x\_prev)  
 if f\_prime\_val == 0:  
 print(" [ПОМИЛКА] Похідна дорівнює нулю.")  
 break  
 x\_curr = x\_prev - f\_val / f\_prime\_val  
 delta = abs(x\_curr - x\_prev)  
 print(f"| {iters:^3} | {x\_prev:<12.8f} | {x\_curr:<12.8f} | {delta:<12.8f} |")  
 if delta < eps or iters > 100: break  
 x\_prev = x\_curr  
 print("-" \* 50)  
 return x\_curr, iters  
  
  
# <-- ЗМІНЕНО: Метод хорд (Regula Falsi) без нерухомої точки -->  
def chord\_method(f\_func, a, b, eps):  
 print("\n--- Метод хорд (Regula Falsi) ---")  
 print(f"Початковий інтервал: [{a:.1f}, {b:.1f}]")  
 print("-" \* 65)  
 print(f"| {'№':^3} | {'a':^12} | {'b':^12} | {'xr':^12} | {'|dx|':^12} |")  
 print("-" \* 65)  
  
 iters = 0  
 xr = a  
  
 while True:  
 iters += 1  
 x\_prev = xr  
  
 fa = f\_func(a)  
 fb = f\_func(b)  
  
 denominator = fb - fa  
 if denominator == 0:  
 print(" [ПОМИЛКА] f(b) - f(a) = 0. Ділення на нуль.")  
 return xr, iters  
  
 xr = a - (fa \* (b - a)) / denominator  
 delta = abs(xr - x\_prev)  
  
 print(f"| {iters:^3} | {a:<12.8f} | {b:<12.8f} | {xr:<12.8f} | {delta:<12.8f} |")  
  
 if delta < eps and iters > 1: break  
 if iters > 100: break  
  
 fxr = f\_func(xr)  
 if fxr == 0: break  
  
 if fa \* fxr < 0:  
 b = xr  
 else:  
 a = xr  
  
 print("-" \* 65)  
 return xr, iters  
  
  
def simple\_iteration\_method(phi\_func, x0, eps):  
 print("\n--- Метод простої ітерації ---")  
 print(f"Початкова точка: x0 = {x0:.1f}")  
 print("-" \* 50)  
 print(f"| {'№':^3} | {'x\_prev':^12} | {'x\_curr':^12} | {'|dx|':^12} |")  
 print("-" \* 50)  
 iters = 0  
 x\_prev = x0  
 while True:  
 iters += 1  
 x\_curr = phi\_func(x\_prev)  
 delta = abs(x\_curr - x\_prev)  
 print(f"| {iters:^3} | {x\_prev:<12.8f} | {x\_curr:<12.8f} | {delta:<12.8f} |")  
 if delta < eps or iters > 100: break  
 x\_prev = x\_curr  
 print("-" \* 50)  
 return x\_curr, iters  
  
  
# --- 5. Основна частина програми ---  
if \_\_name\_\_ == "\_\_main\_\_":  
 a = sp.Float(0.0)  
 b = sp.Float(1.0)  
 epsilon = sp.Float(0.00001)  
  
 sym\_interval = sp.Interval(a, b)  
  
 print("Група: ОІ-25")  
 print("ПІБ: Олег Коновалюк")  
 print("ЛР №: 3, Варіант 12\n")  
 print(f"Символьний вираз f(x): {f\_sym}")  
 print(f"Перша похідна f'(x): {f\_prime\_sym}")  
 print(f"Друга похідна f''(x): {f\_double\_prime\_sym}")  
 print(f"Функція ітерації phi(x): {phi\_sym}")  
 print(f"Похідна phi'(x): {phi\_prime\_sym}")  
 print("-" \* 50)  
 print(f"Відрізок: [{a}, {b}]")  
 print(f"Точність: {epsilon}\n")  
  
 # --- Перевірка умов ---  
 print("--- ПЕРЕВІРКА УМОВ ЗБІЖНОСТІ ---")  
  
 if not check\_bisection\_and\_chord\_conditions(f, f\_sym, x\_sym, sym\_interval):  
 print("\n[СТОП] Не виконано базові умови. Робота програми зупинена.")  
 exit()  
  
 check\_simple\_iteration\_convergence(phi\_prime, phi\_prime\_sym, x\_sym, sym\_interval)  
  
 is\_newton\_ok, x0\_newton = check\_newton\_convergence(  
 f, f\_prime, f\_double\_prime,  
 f\_sym, f\_prime\_sym, f\_double\_prime\_sym, x\_sym, sym\_interval  
 )  
  
 # <-- НОВЕ: Виклик дорадчої функції перевірки похідних для хорд -->  
 check\_chord\_derivatives\_advisory(  
 f\_prime, f\_double\_prime,  
 f\_prime\_sym, f\_double\_prime\_sym, x\_sym, sym\_interval  
 )  
 print("-" \* 50)  
  
 # --- Виклик методів ---  
 root, iters = bisection\_method(f, a, b, epsilon)  
 print(f"Результат (дихотомія): Корінь = {root:.8f}, Ітерацій = {iters}\n")  
  
 # Виклик методу хорд (Regula Falsi)  
 root, iters = chord\_method(f, a, b, epsilon)  
 print(f"Результат (хорди): Корінь = {root:.8f}, Ітерацій = {iters}\n")  
  
 if is\_newton\_ok:  
 root, iters = newton\_method(f, f\_prime, x0\_newton, epsilon)  
 print(f"Результат (Ньютон): Корінь = {root:.8f}, Ітерацій = {iters}\n")  
 else:  
 print("\n--- Метод Ньютона ---")  
 print("Пропущено через невідповідність умов збіжності.")  
  
 x0\_simple = (a + b) / 2  
 root, iters = simple\_iteration\_method(phi, x0\_simple, epsilon)  
 print(f"Результат (ітерації): Корінь = {root:.8f}, Ітерацій = {iters}\n")

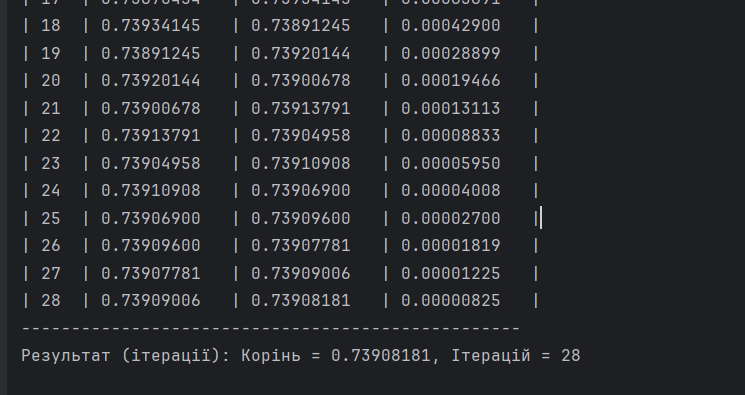
Приклад роботи програми :











Висновок :Навчився використовувати метод хорд,дотичних,ітераційний та половинного ділення.Закріпив роблячи завдання лабораторної роботи №3.Перевіряв кожен з методів на збіжність використовуючи матеріали з лекцій в внс.Результати ,зроблені вручну зійшлися з обчисленими за допомогою пайтона.Також зроблено експеремент,результатом якого є таблиця кількості ітерацій в кожному методі за різної похибки.