Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до виконаної лабораторної роботи № 4

з дисципліни

“Чисельні методи”

на тему:

***«Методи розв’язування нелінійних рівнянь»***

Виконав

студент групи *ОІ-11 сп*

*Вальчевський П. В.*

Викладач:

*Сенета М. Я.*

Львів – 2023

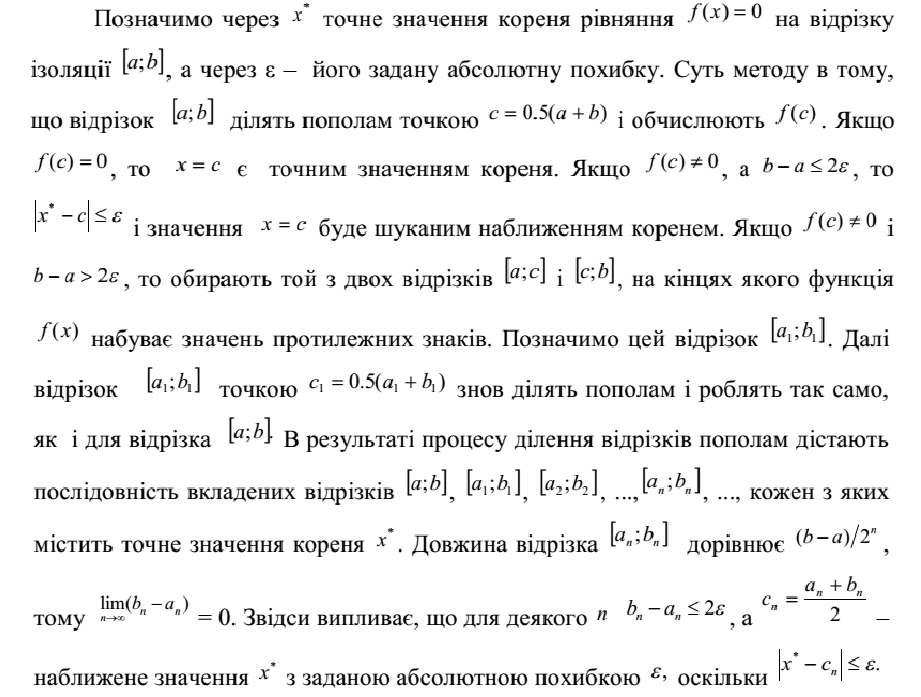
## Лабораторна робота № 4

*Тема роботи:*  **Методи розв’язування нелінійних рівнянь.**

*Мета роботи:* ***вивчити та засвоїти ітераційні методи розв’язування***

***алгебраїчних і трансцендентних рівнянь.***

***Порядок виконання роботи***

1. **Основні теоретичні відомості**
2. **Метод дихотомії (половинного ділення).** При розв’язанні нелінійного рівняння методом половинного поділу задається проміжок [a;b], на якому існує лише один розв'язок і бажана точність *є > 0* розв’язання задачі.
3. 

**Метод Хорд.** Ідея методу полягає в тому, що на досить малому проміжку [a;b], дугу кривої *y = f(x)* заміняють хордою, а за наближене значення кореня приймають точку перетину хорди з віссю ОХ.

1. При розв’язанні нелінійного рівняння методом хорд задається проміжок [a;b], на якому існує лише один розв'язок і бажана точність *є > 0* розв’язання задачі.
2. Це класичний ітераційний метод наближеного обчислення кореня рівняння f(x) = 0 на відрізку його ізоляції [a;b]. Цей метод визначає формула

**Метод Дотичних (Ньютона).** В основі цього методу лежить розкладання функції в ряд Тейлора:

Члени, що містять h у другому і вищих степенях, відкидаються, внаслідок чого одержується наведена вище наближена формула для оцінки .

Початкове наближення слід вибирати з умови

1. При розв’язанні нелінійного рівняння методом дотичних задається проміжок [a;b], на якому існує лише один розв'язок, початкове наближення розв’язку і бажана точність *є > 0* розв’язання задачі.
2. Для визначення точки перетину (k+1)-ї дотичної з віссю абсцис користуються формулою
3. **Метод простої ітерації.** Виберемо будь-яке значення x0 із відрізка [a;b] і підставимо його в праву частину рівняння Отримаємо значення Потім значення x1 підставимо знову у рівняння та одержимо х2. Повторюючи цей процес, отримаємо послідовність чисел x1, x2,… xn. Обчислюваний процес закінчимо тоді, коли
4. При обчисленнях можливі два випадки:
5. а) послідовність чисел x1, x2,… xn… **збігається**, тобто є межа і ця межа буде коренем рівнянням f(x) = 0;
6. б) послідовність чисел x1, x2,… xn… **збігається**, тобто немає межі.
7. Метод ітерації використовується, коли виконується умова: При цьому чим менше , тим краще збіжність ітераційного процесу.
8. **Номер варіанту – 3.**

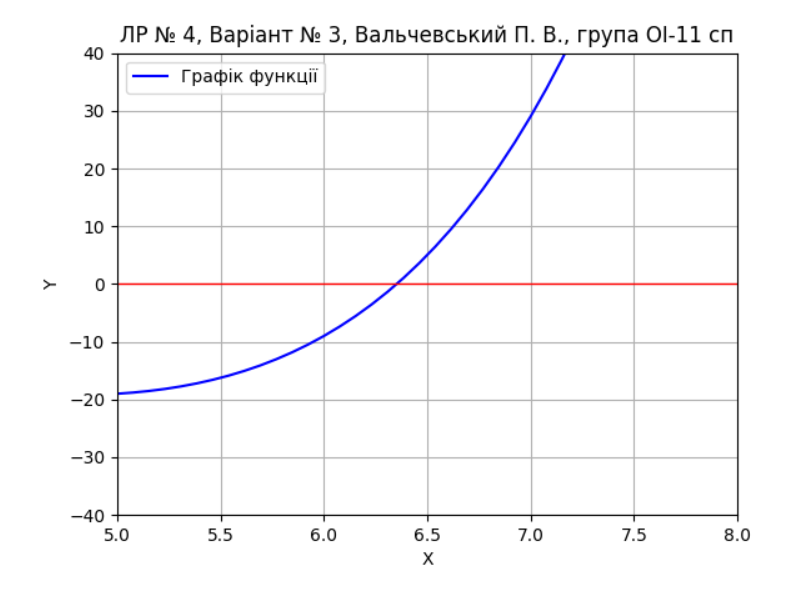
****

1. **Умова завдання**
2. Знайти один з коренів рівняння f(x)=0, використовуючи методи:

* половинного ділення
* дотичних
* хорд
* простої ітерації

з точністю .

1. Вивести кількість ітерацій, необхідних для досягнення точності для кожного з методів.
2. Провести дослідження кількості ітерацій для кожного з методів при зміні точності від до
3. У звіті подати відповідні таблиці та графіки.
4. **Алгоримт розв’язання завдання**
5. Обраховую усі дані для моєї функції:
   1. Фрагмент функції на проміжку [5; 8]:



* 1. Значення кореня: 6.3523448924;
  2. Похідна: ;
  3. Похідна другого порядку: ;
  4. Еквівалентна функція: ;
  5. Похідна еквівалентної функції: ;

1. Аналіз та процес вивчення блок-схем реалізації кожного методу з теоретичних матеріалів лабораторної роботи для таких методів:
   1. Метод половинного ділення;
   2. Метод хорд;
   3. Метод дотичних;
   4. Метод простої ітерації;
2. Процес проектування кожного методу, допоміжних функцій та їх графіків залежностей у програмній реалізації для отримання кількості ітерацій, кореня, списку усіх ітерацій, таблиці кожного методу у вигляді рядка, графіка залежностей та функції.
3. Процес кодування та тестування кожного методу та виправлення неполадок при їх наявності в програмному коді.
4. Процес послідовного та форматованого табличного виводу усіх даних та їх графіків (в окремих вікнах) – до кожного методу.
5. **Виконане завдання вручну**
6. Метод половинного ділення:
   1. Беру проміжок [5; 8] => a = 5, b = 8;
   2. Обраховую значення ;
   3. Перевіряю умову |x| > :
      1. Первіряю умову f(a) \* f(x) > 0 при правдивій умові;
         1. Визначаю нову межу зліва a = x при правдивій умові;
         2. Визначаю нову межу справа b = x при неправдивій умові;
         3. Повторюю умови пункту 1.2;
      2. Записую значення х та завершую роботу при неправдивій умові.
   4. Перші три ітерації:
      1. a0: 4, b0: 8, x0: 6, a0 = x0;
      2. a0: 6, b0: 8, x0: 7, b0 = x0;
      3. a0: 7, b0: 8, x0: 6.5, b0 = x0;
7. Метод хорд:
   1. Перевіряю умову на f(a) > 0;
      1. Міняю місцями проміжки a, b при правдивій умові;
   2. Перевіряю умову |b – a| > :
      1. При правдивій умові:
         1. Перевіряю чи дріб не неульовий для f(a) – f(b) = 0;
            1. Завершую роботу для цього методу, бо дріб виявся нульовим;
         2. Обчислюю вираз ;
         3. Перевіряю ще умову f(a) \* f(c) > 0;
            1. Роблю заміну a = c при правдивій умові;
            2. Роблю заміну b = c при правдивій умові;
         4. Повертаюсь до пункту 2.2;
      2. При неправдивій умові завершую роботу та повертаю значення х (с у моєму випадку);
   3. Перші три ітерації:
      1. a0: 4, b0: 8, c0: 4.47222, a0 = c0;
      2. a1: 4.47222, b1: 8, c1: 4.927226, a1 = c1;
      3. a2: 4.927226, b2: 8, c2: 5.32947, a2 = c2;
8. Метод дотичних:
   1. Перевіряю умову f(a) \* f(a)’’ > 0 для задання початкового значення х0;
      1. При правдивій умові задаю значення для x0 = a;
      2. При неправдивій умові задаю значення x0 = b;
   2. Перевіряю чи дріб не є нульовим f(x0)’ = 0;
      1. Якщо дріб виявився нульовим завершую роботу з даним методом;
   3. Розраховую наближене значення хr на даному етапі ;
   4. Перевіряю умову умову |xr – x0| > ;
      1. Роблю заміну x0 = xr при правдивій умові та повертаюсь до пункту 3.2;
      2. При неправдивій умові повертаю значення хr;
   5. Перші три ітерації:
      1. x0: 8 (x0 = b тому що f(a) \* f(a)'' < 0), xr0: 7.12682, x = xr;
      2. x1: 7.12682, xr1: 6.58302, x = xr;
      3. x2: 6.58302, xr2: 6.37867, x = xr;
9. Метод простої ітерації:
   1. Розраховую функції та перевіряю умову при неправдивій умові завершую роботу з цим методом.
   2. Розраховую значення xr = ;
   3. Перевіряю умову |xr – x0| > :
      1. При правдивій умові роблю заміну x0 = xr та повертаюсь до пункту 4.2;
      2. При неправдивій умові повертаю значення xr та завершую ітерації.
   4. Перші три ітерації:
      1. x0: 4 тому що , xr0: 5.04439, x0 = xr;
      2. x1: 5.04439, xr1: 5.69743, x0 = xr;
      3. x2: 5.69743, xr2: 6.04267, x0 = xr;
10. **Код програмної реалізації**

*Файл main.py*

from math import fabs, log, log2 *# Модлуль числа, логарифми.*from tabulate import tabulate *# Табуляція списку для отримання рядка форматованої таблиці списку.*from sympy import symbols, Eq, real\_roots, nsolve *# Для обрахунку коренів.*import matplotlib.pyplot as plt *# Малювання графіку залежності.*import numpy as np *# Для зберігання й обчислення діпазону значення Х і У функції.  
  
# Обчислення значення функції.*def f(x=float) -> float:  
 return 2 \*\* x - 2 \* x \*\* 2 - 1  
  
  
*# Обчислення значення першої похідної функції.*def f1(x=float) -> float:  
 return 2 \*\* x \* log(2) - 4 \* x  
  
  
*# Обчислення значення другої похідної функції.*def f2(x=float) -> float:  
 return 2 \*\* x \* log(2) \*\* 2 - 4  
  
  
*# Обчислення значення ітераційної функції.*def phi(x=float) -> float:  
 return log2(2 \* x \*\* 2 + 1)  
  
  
*# Обчислення значення похідної ітераційної функції.*def phi1(x=float) -> float:  
 return 4 \* x / (log(2) \* (2 \* x \*\* 2 + 1))  
  
  
*# Метод половинного поділу (дихотомії).*def middleDivisionMethod(a=float, b=float, eps=float, countMaxIteration=int) -> list:  
 allListIteration = []  
 for i in range(countMaxIteration):  
 listIteration = []  
 x = (a + b) / 2  
 listIteration.append(f"Ітерація # {i + 1}")  
 listIteration.append(f"a{i}: {a}")  
 listIteration.append(f"b{i}: {b}")  
 listIteration.append(f"x{i}: {x}")  
 listIteration.append(f"{fabs(f(x)) > eps}")  
 if fabs(f(x)) > eps:  
 listIteration.append(f"{f(a) \* f(x) > 0}")  
 if f(a) \* f(x) > 0:  
 a = x  
 listIteration.append(f"a{i} = x{i}")  
 else:  
 b = x  
 listIteration.append(f"b{i} = x{i}")  
 else:  
 listIteration[-1] += " =>"  
 listIteration.append(f"Корінь знайдено => ")  
 listIteration.append(f"x = {x}")  
 allListIteration.append(listIteration)  
 return [i + 1, x, allListIteration]  
 allListIteration.append(listIteration)  
 else:  
 raise Exception(  
 f"На цьому проміжку [{a}; {b}] - коренів не знайдено або цей проміжок не підходить для цього методу!")  
  
  
*# Метод хорд.*def chordMethod(a=float, b=float, eps=float, countMaxIteration=int) -> list:  
 allListIteration = []  
 c = 0.0  
 for i in range(countMaxIteration):  
 listIteration = []  
 listIteration.append(f"Ітерація # {i + 1}")  
 if i == 0 and f(a) > 0:  
 a, b = b, a  
 listIteration.append(f"a{i}: {a} тому що f(a) > 0")  
 listIteration.append(f"b{i}: {b} тому що f(a) > 0")  
 else:  
 listIteration.append(f"a{i}: {a}")  
 listIteration.append(f"b{i}: {b}")  
 listIteration.append(f"{fabs(b - a) > eps}")  
 if fabs(b - a) > eps:  
 if (f(b) - f(a)) == 0:  
 raise ZeroDivisionError("Ділення на нуль!")  
 c = a - (b - a) / (f(b) - f(a)) \* f(a)  
 listIteration.append(f"c{i}: {c}")  
 listIteration.append(f"{f(a) \* f(c) > 0}")  
 if f(a) \* f(c) > 0.0:  
 a = c  
 listIteration.append(f"a{i} = c{i}")  
 else:  
 b = c  
 listIteration.append(f"b{i} = c{i}")  
 else:  
 listIteration[-1] += " =>"  
 listIteration.append(f"Корінь знайдено => ")  
 listIteration.append(f"x = {c}")  
 listIteration.append("-" \* 10)  
 allListIteration.append(listIteration)  
 return [i + 1, a, allListIteration]  
 allListIteration.append(listIteration)  
 else:  
 raise Exception(  
 f"На цьому проміжку [{a}; {b}] - коренів не знайдено або цей проміжок не підходить для цього методу!")  
  
  
*# Метод дотичних.*def secantMethod(a=float, b=float, eps=float, countMaxIteration=int) -> list:  
 x0 = 0.0  
 allListIteration = []  
 for i in range(countMaxIteration):  
 listIteration = []  
 listIteration.append(f"Ітерація # {i + 1}")  
 if i == 0:  
 if f(a) \* f2(a) > 0.0:  
 x0 = a  
 listIteration.append(f"x{i}: {x0} (x0 = a тому що f(a) \* f2(a) > 0.0)")  
 else:  
 x0 = b  
 listIteration.append(f"x{i}: {x0} (x0 = b тому що f(a) \* f2(a) <= 0.0)")  
 else:  
 listIteration.append(f"x{i}: {x0}")  
 if (f1(x0) == 0.0):  
 raise ZeroDivisionError("Ділення на нуль!")  
 xr = x0 - f(x0) / f1(x0)  
 listIteration.append(f"xr{i}: {xr}")  
 listIteration.append(f"{fabs(x0 - xr) > eps}")  
 if fabs(x0 - xr) > eps:  
 x0 = xr  
 listIteration.append(f"x = xr")  
 else:  
 listIteration[-1] += " => Корінь знайдено => "  
 listIteration.append(f"x = {xr}")  
 allListIteration.append(listIteration)  
 return [i + 1, xr, allListIteration]  
 allListIteration.append(listIteration)  
 else:  
 raise Exception(  
 f"На цьому проміжку [{a}; {b}] - коренів не знайдено або цей проміжок не підходить для цього методу!")  
  
  
*# Метод простої ітерації.*def simpleIterationMethod(a=float, b=float, eps=float, countMaxIteration=int) -> list:  
 x0 = 0.0  
 allListIteration = []  
 for i in range(countMaxIteration):  
 listIteration = []  
 listIteration.append(f"Ітерація # {i + 1}")  
 if i == 0:  
 if not (fabs(phi1(a)) < 1 and fabs(phi1(b)) < 1):  
 raise Exception(  
 f"На цьому проміжку [{a}; {b}] - коренів не знайдено або цей проміжок не підходить для цього методу!")  
 else:  
 x0 = a  
 listIteration.append(f"x{i}: {x0} тому що phi'(a) i phi'(b) < 1")  
 else:  
 listIteration.append(f"x{i}: {x0}")  
 xr = phi(x0)  
 listIteration.append(f"xr{i}: {xr}")  
 listIteration.append(f"{fabs(xr - x0) > eps}")  
 if fabs(xr - x0) > eps:  
 x0 = xr  
 listIteration.append(f"x0 = xr")  
 else:  
 listIteration[-1] += " => Корінь знайдено => "  
 listIteration.append(f"x = {xr}")  
 allListIteration.append(listIteration)  
 return [i + 1, xr, allListIteration]  
 allListIteration.append(listIteration)  
 else:  
 raise Exception(  
 f"На цьому проміжку [{a}; {b}] - коренів не знайдено або цей проміжок не підходить для цього методу!")  
  
  
*# Рядок (таблиця) Методу.*def strMethodInTable(func, a=float, b=float, eps=float, countMaxIteration=int, nameMethod=str,  
 headersTable=list) -> str:  
 nameTable = f"ЛР № 4, варіант № 3, Вальчевський П. В., ОІ-11 сп - Знаходження кореня за допомогою {nameMethod}:"  
 descriptionTable = f"\t\* взято відрізок [{a}; {b}], епсолон: {eps}, максимальна кількість ітерацій: {countMaxIteration}."  
 table = tabulate(func(a, b, eps, countMaxIteration)[2], headersTable, tablefmt="pretty")  
 return f"{nameTable}\n{descriptionTable}\n{table}"  
  
  
*# Отримати дані результату виконання Методу.*def getDataOfResultMethod(func, nameMethod=str, a=float, b=float, eps=float, countMaxIteration=int) -> list:  
 funcResult = func(a, b, eps, countMaxIteration)  
 return [nameMethod, f"{funcResult[0]}", f"{funcResult[1]}"]  
  
  
*# Рядок (таблиця) результатів виконання програми.*def strResultInTable(a=float, b=float, eps=float, countMaxIteration=int) -> str:  
 nameTable = f"ЛР № 4, варіант № 3, Вальчевський П. В., ОІ-11 сп - Результати програми:"  
 descriptionTable = f"\t\* взято відрізок [{a}; {b}], епсолон: {eps}, максимальна кількість ітерацій: {countMaxIteration}."  
 listResult = [  
 getDataOfResultMethod(middleDivisionMethod, "Метод половинного поділу", a, b, eps, countMaxIteration),  
 getDataOfResultMethod(chordMethod, "Метод хорд", a, b, eps, countMaxIteration),  
 getDataOfResultMethod(secantMethod, "Метод дотичних", a, b, eps, countMaxIteration),  
 getDataOfResultMethod(simpleIterationMethod, "Метод простої ітерації", a, b, eps, countMaxIteration)  
 ]  
 table = tabulate(listResult, ["Назва методу", "Кількість ітерацій", "Корінь"], tablefmt="pretty")  
 return f"{nameTable}\n{descriptionTable}\n{table}"  
  
  
*# Демонстрація графіку залежності кількості ітерацій до епсолону.*def showGraphicMethodIterationAndEps(func, nameFunc, a=float, b=float, countMaxIteration=int) -> None:  
 powEps = [-i for i in range(1, 10 + 1)]  
 countIterationInPowEps = [func(a, b, 10 \*\* i, countMaxIteration)[0] for i in powEps]  
 plt.plot(powEps, countIterationInPowEps, label='Залежність')  
 plt.xlabel("Степінь епсолона в форматі 10 ^ i")  
 plt.ylabel("Кікільсть ітерацій")  
 plt.title(f"ЛР № 4, варіант № 3, Вальчевський П. В., ОІ-11 сп\nГрафік {nameFunc}")  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
*# Вивід графіку функції.*def drawFx(a=int, b=int) -> None:  
 *# Константи підібрані вручну.* num = 40 *# Кількість Х для подальшого зображення графіку.* aY = -40 *# Нижня межа У.* bY = 40 *# Верхня межа У.  
 # Діапазон Х.* X = np.linspace(a, b, num)  
 *# Обраховуємий діапазон У.* Y = 2 \*\* X - 2 \* X \*\* 2 - 1  
 *# Малювання графіку.* plt.plot(X, Y, label='Графік функції', color='b')  
 plt.xlabel('X')  
 plt.ylabel('Y')  
 plt.title('ЛР № 4, Варіант № 3, Вальчевський П. В., група ОІ-11 сп')  
 plt.legend()  
 plt.grid(True)  
 plt.xlim(a, b)  
 plt.ylim(aY, bY)  
 plt.axvline(0, color='r', linewidth=1) *# Вісь X.* plt.axhline(0, color='r', linewidth=1) *# Вісь Y.* plt.show()  
  
*# Обрахунок й вивід коренів функції.*def calcAndOutputRoots() -> None:  
 X = symbols('X') *# Оголошення символу для подальшого обрахунку функції.* equation = Eq(2 \*\* X - 2 \* X \*\* 2 - 1, 0) *# Розрахунок функції.* print("Один з коренів рівння, який шукався у програмі (для перевірки за допомогою сторінніх бібліотек):")  
 root = nsolve(equation, X, 6.0) *# Пошук кореня близького до певного значення.* print(f"\tКорінь рівняння: {round(root, 10)}")  
  
  
*# Програма виконання.*if \_\_name\_\_ == "\_\_main\_\_":  
 a, b, eps, countMaxIteration = 5, 8, 10 \*\* -5, 10 \*\* 10  
 print(strMethodInTable(middleDivisionMethod, a, b, eps, countMaxIteration, "Методу половинного поділу (дихотомії)",  
 ["Ітерація", "a", "b", "x", "Умова: |f(x)| > eps", "Умова: f(a) \* f(x) > 0", "Заміна"]))  
 print(strMethodInTable(chordMethod, a, b, eps, countMaxIteration, "Методу хорд",  
 ["Ітерація", "a", "b", "Умова: |b - a| > eps", "c", "Умова: f(a) \* f(c) > 0", "Заміна"]))  
 print(strMethodInTable(secantMethod, a, b, eps, countMaxIteration, "Методу дотичних",  
 ["Ітерація", "x0", "xr", "Умова: |x0 - xr| > eps", "Заміна"]))  
 print(strMethodInTable(simpleIterationMethod, a, b, eps, countMaxIteration, "Методу простої ітерації",  
 ["Ітерація", "x0", "xr", "Умова: |x0 - xr| > eps", "Заміна"]))  
 print(strResultInTable(a, b, eps, countMaxIteration))  
 showGraphicMethodIterationAndEps(middleDivisionMethod, "Методу половинного поділу (дихотомії)", a, b,  
 countMaxIteration)  
 showGraphicMethodIterationAndEps(chordMethod, "Методу хорд", a, b, countMaxIteration)  
 showGraphicMethodIterationAndEps(secantMethod, "Методу дотичних", a, b, countMaxIteration)  
 showGraphicMethodIterationAndEps(simpleIterationMethod, "Методу простих ітерацій", a, b, countMaxIteration)  
 drawFx(a, b)  
 calcAndOutputRoots()  
 print("Графіки залежності епсолона й кількості ітерацій в інших вікнах!")

1. **Скріншот отриманого результату**

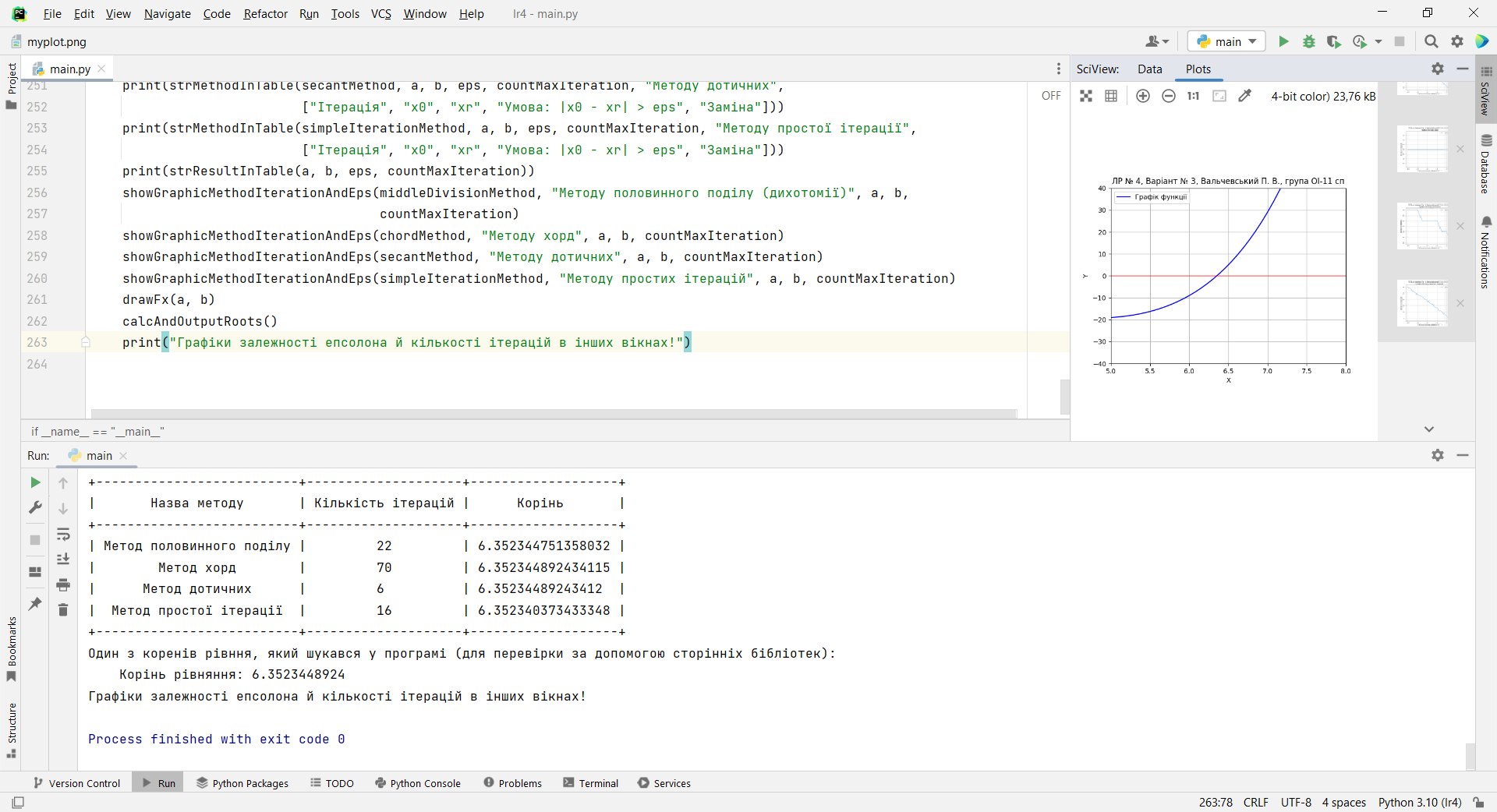


Рис. 1 Знімок з робочого середовища для підтвердження реалізації завдання.

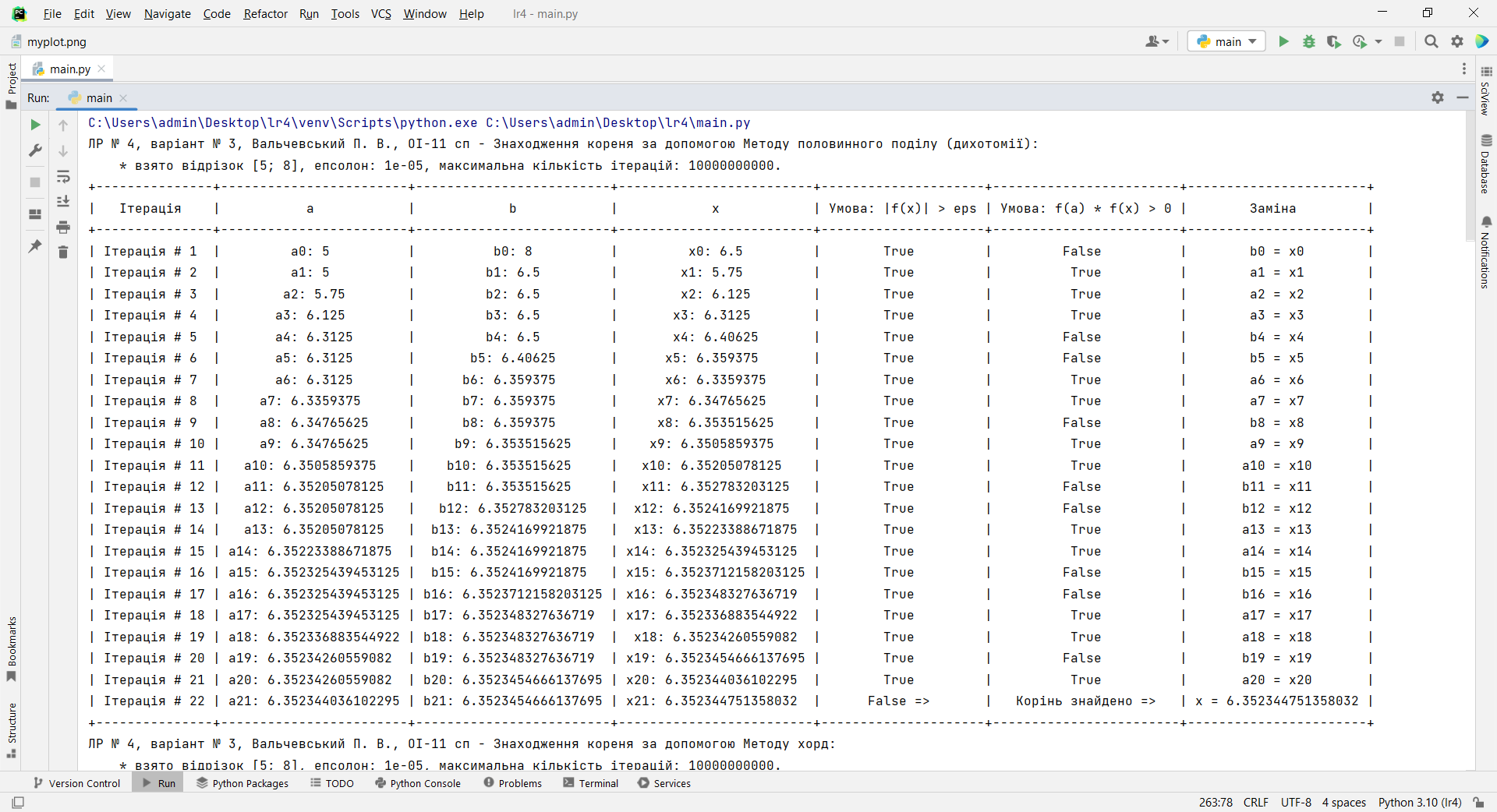


Рис. Результат у консолі.

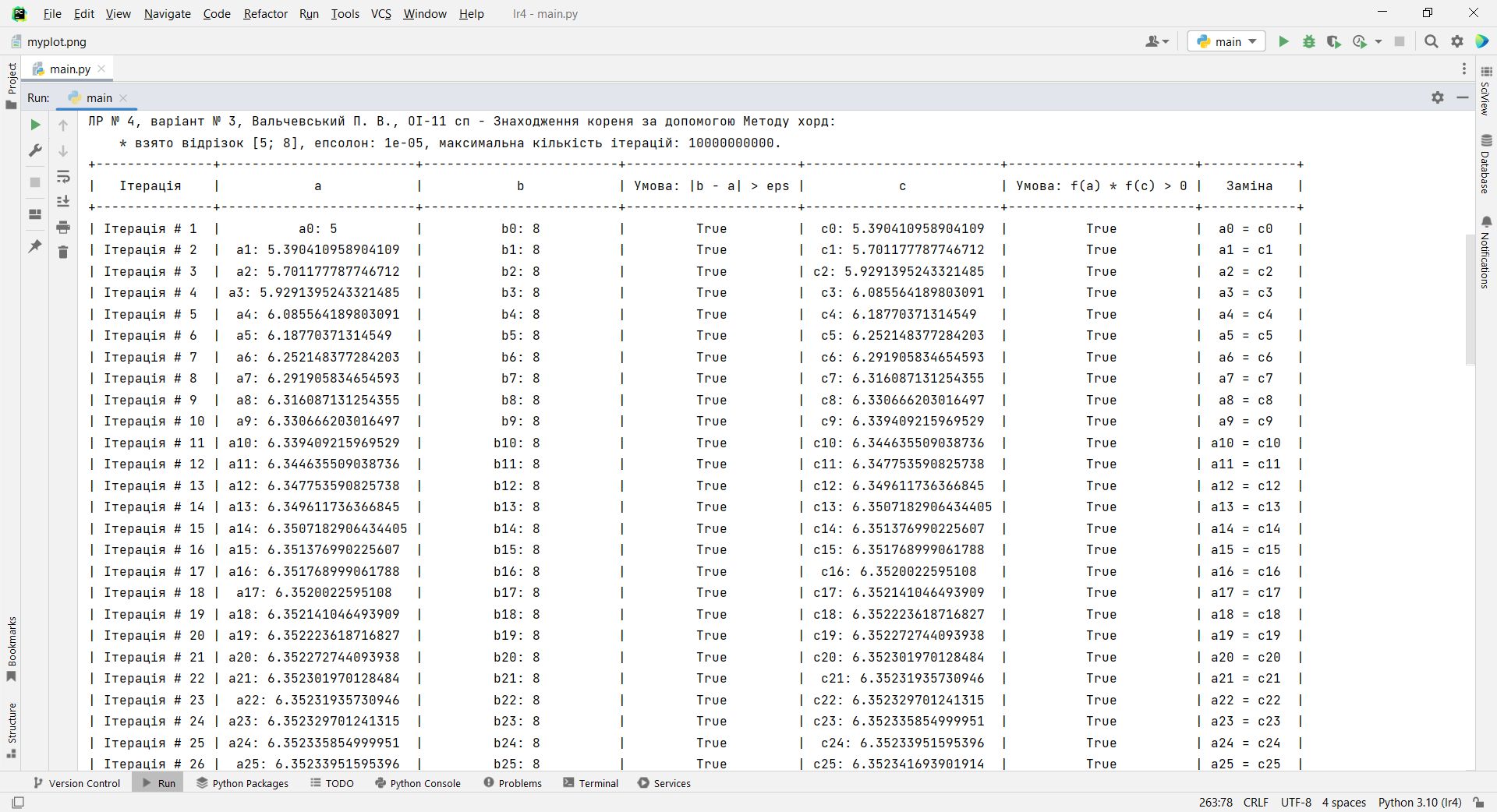


Рис. 3 Результат у консолі (продовження).

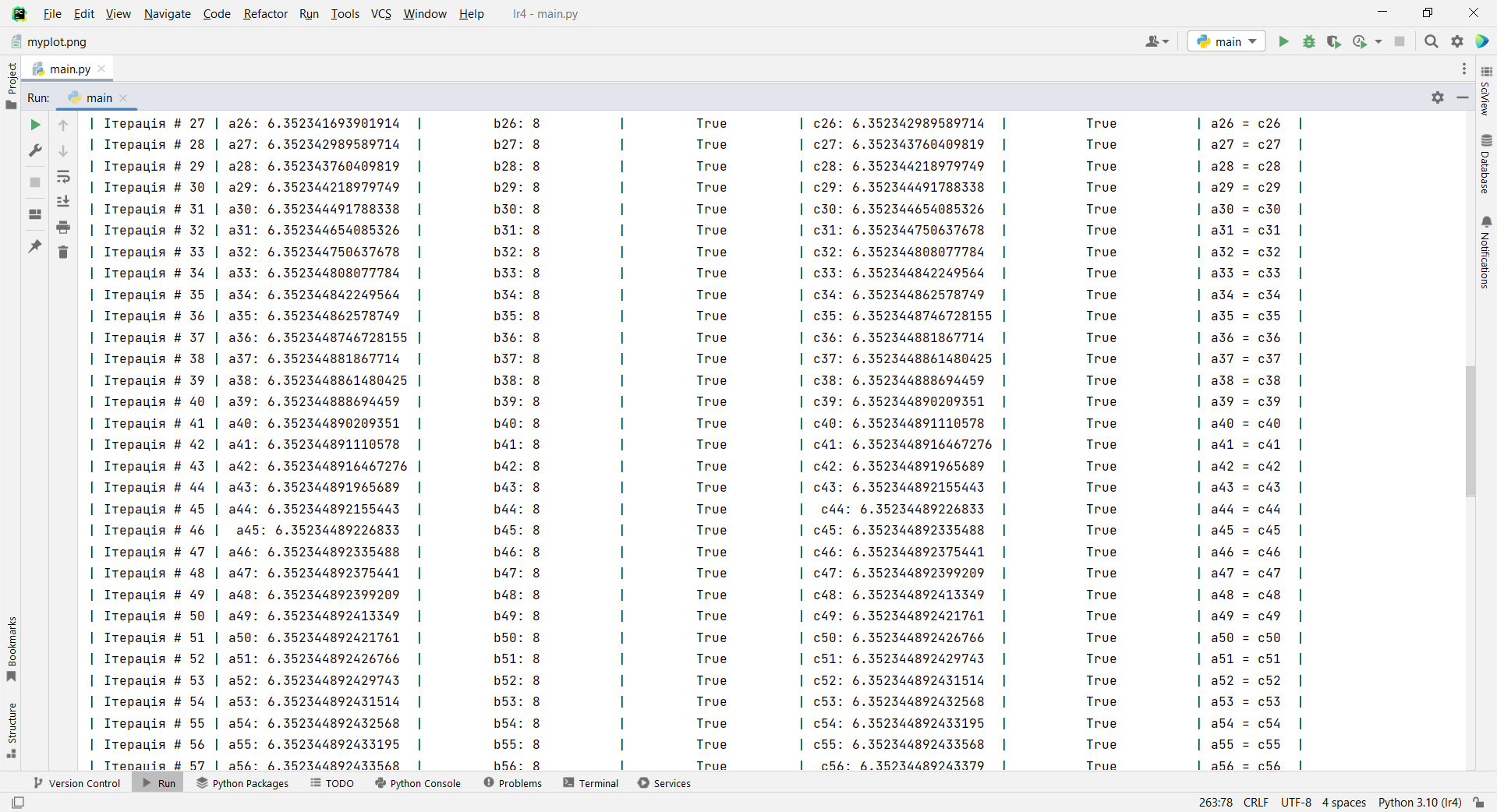


Рис. 4 Результат у консолі (продовження).

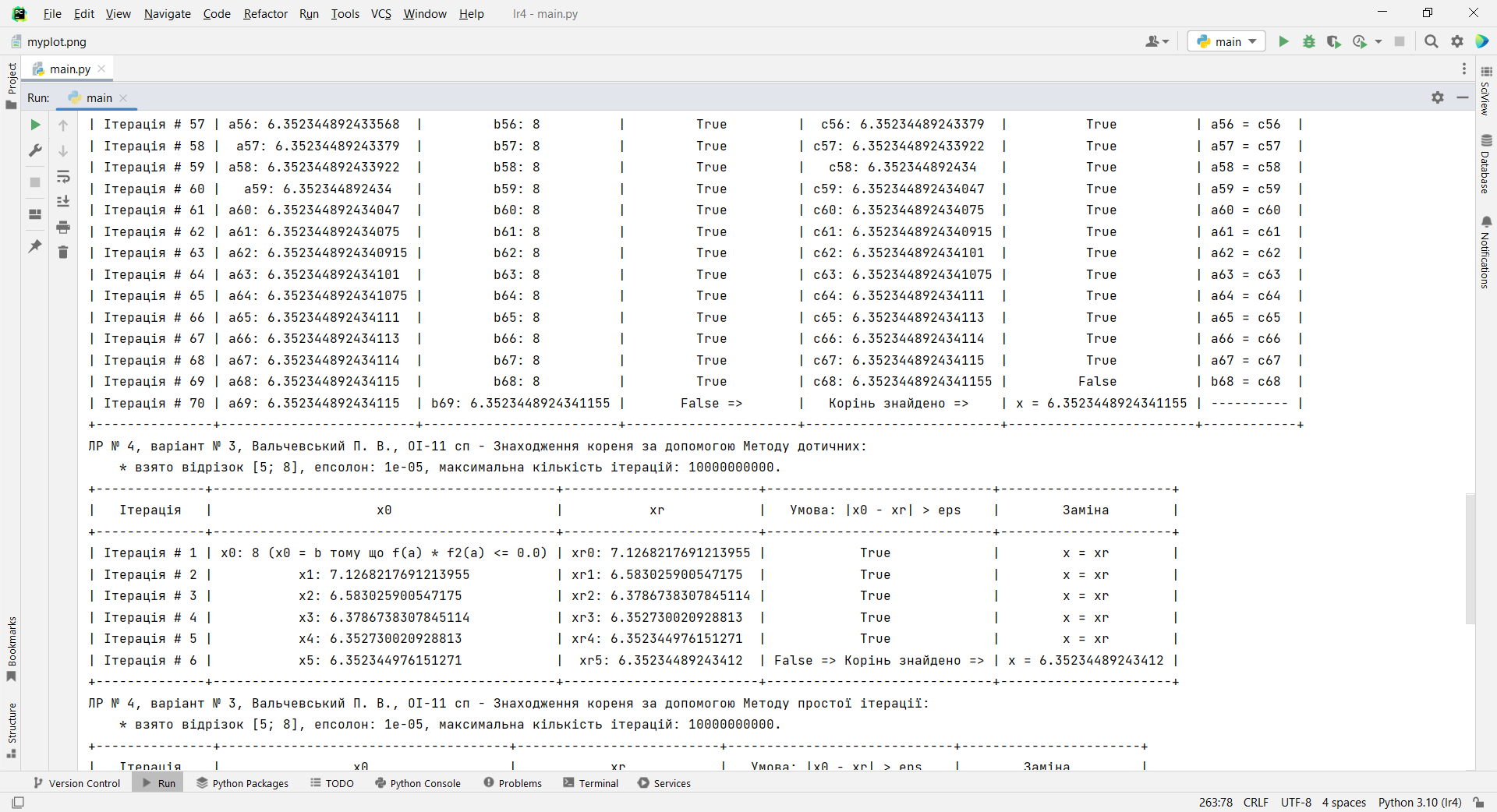


Рис. 5 Результат у консолі (продовження).

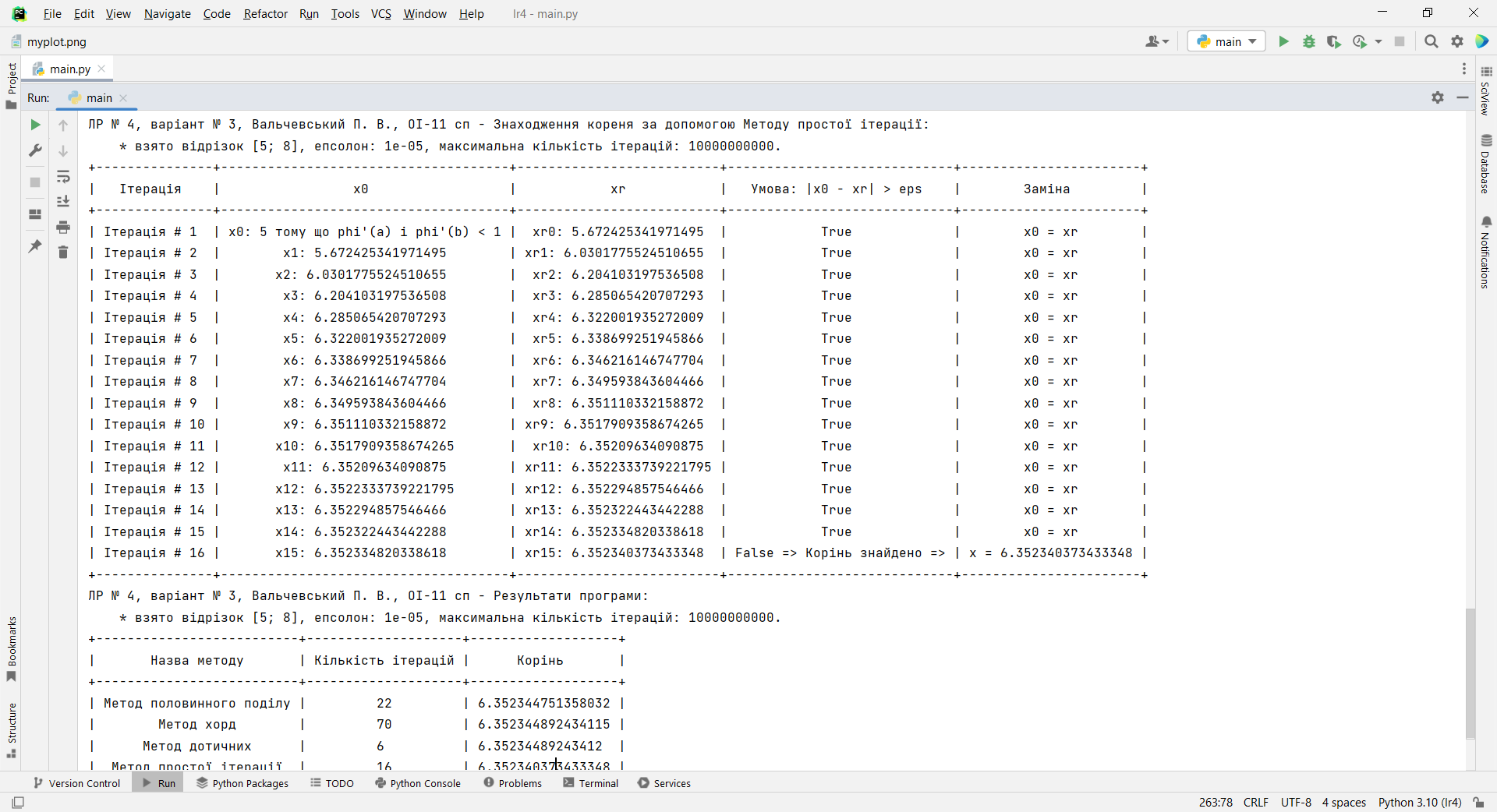


Рис. 6 Результат у консолі (продовження).

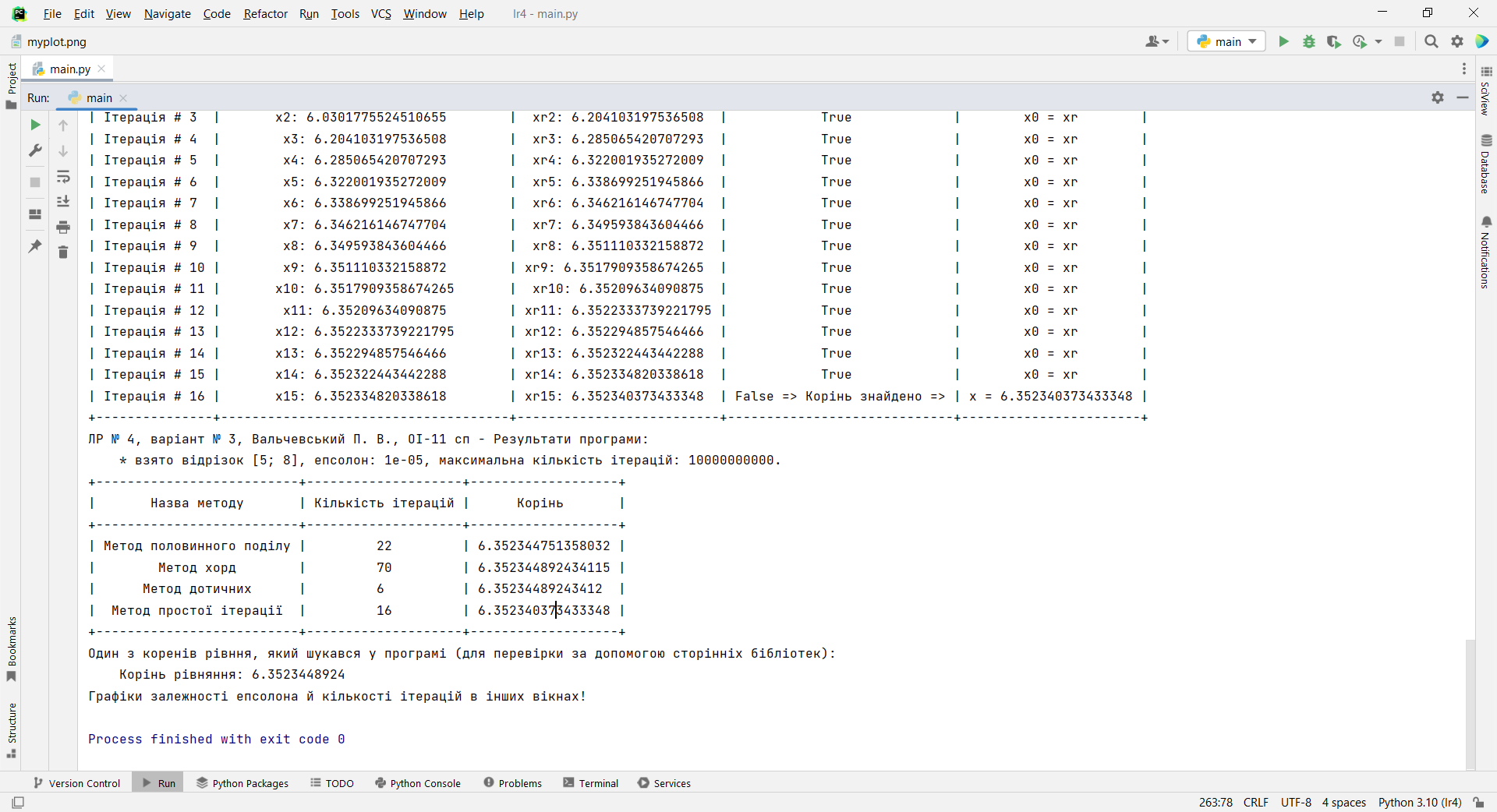


Рис. 7 Результат у консолі (продовження).

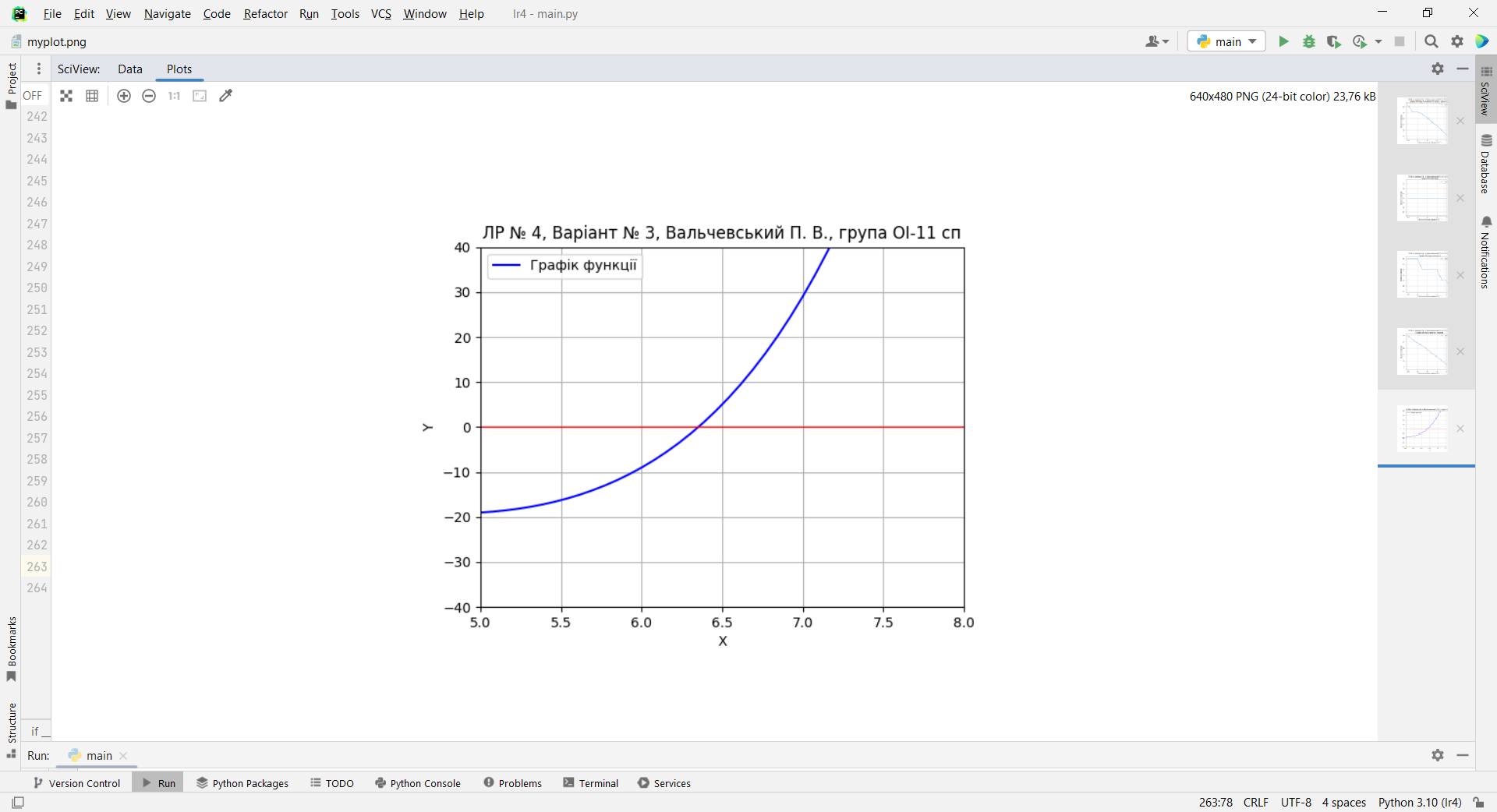


Рис. 8 Графік функції (додаткове вікно програми).

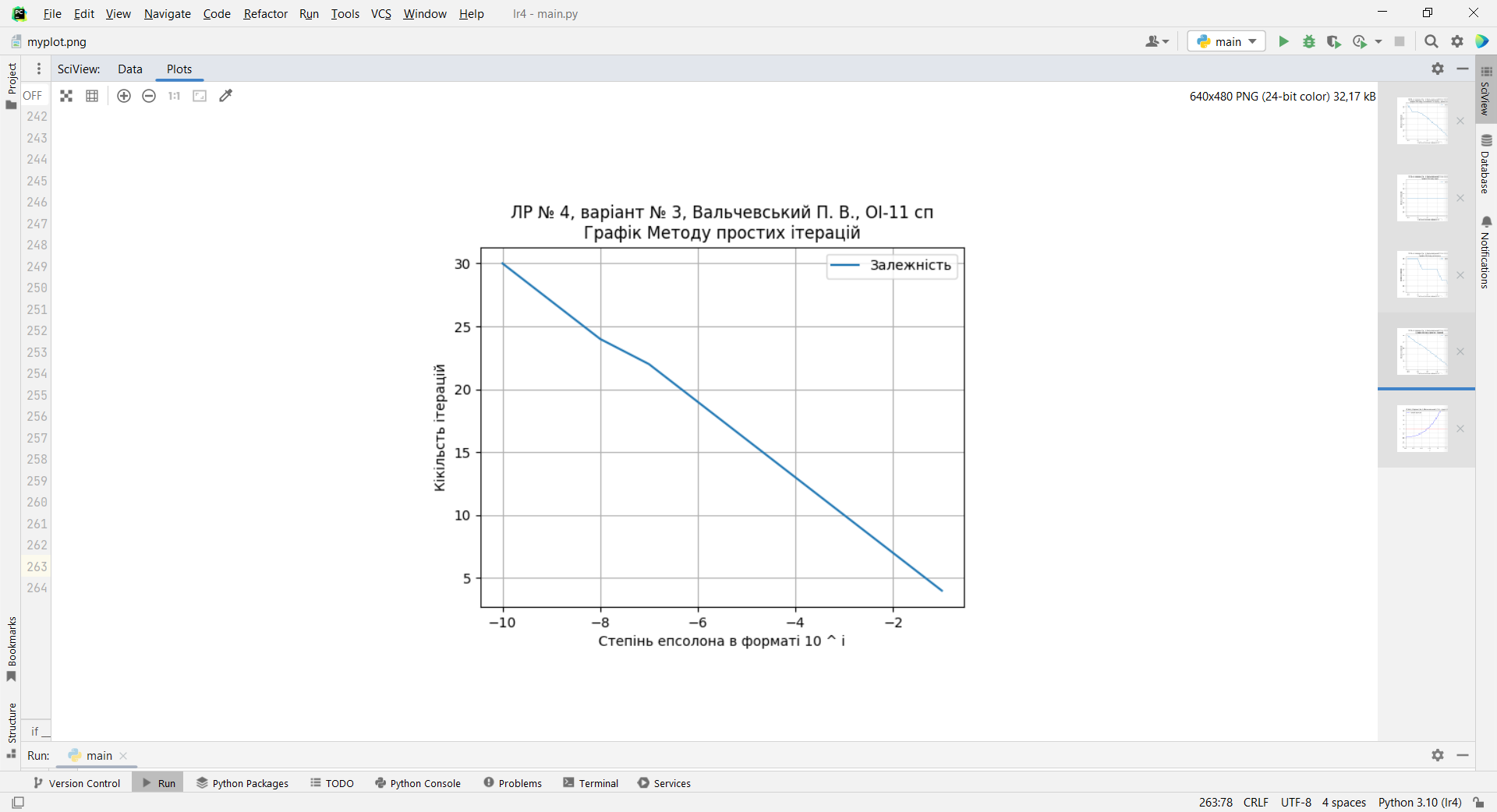


Рис. 9 Графік залежності для Методу простої ітерації (додаткове вікно програми).

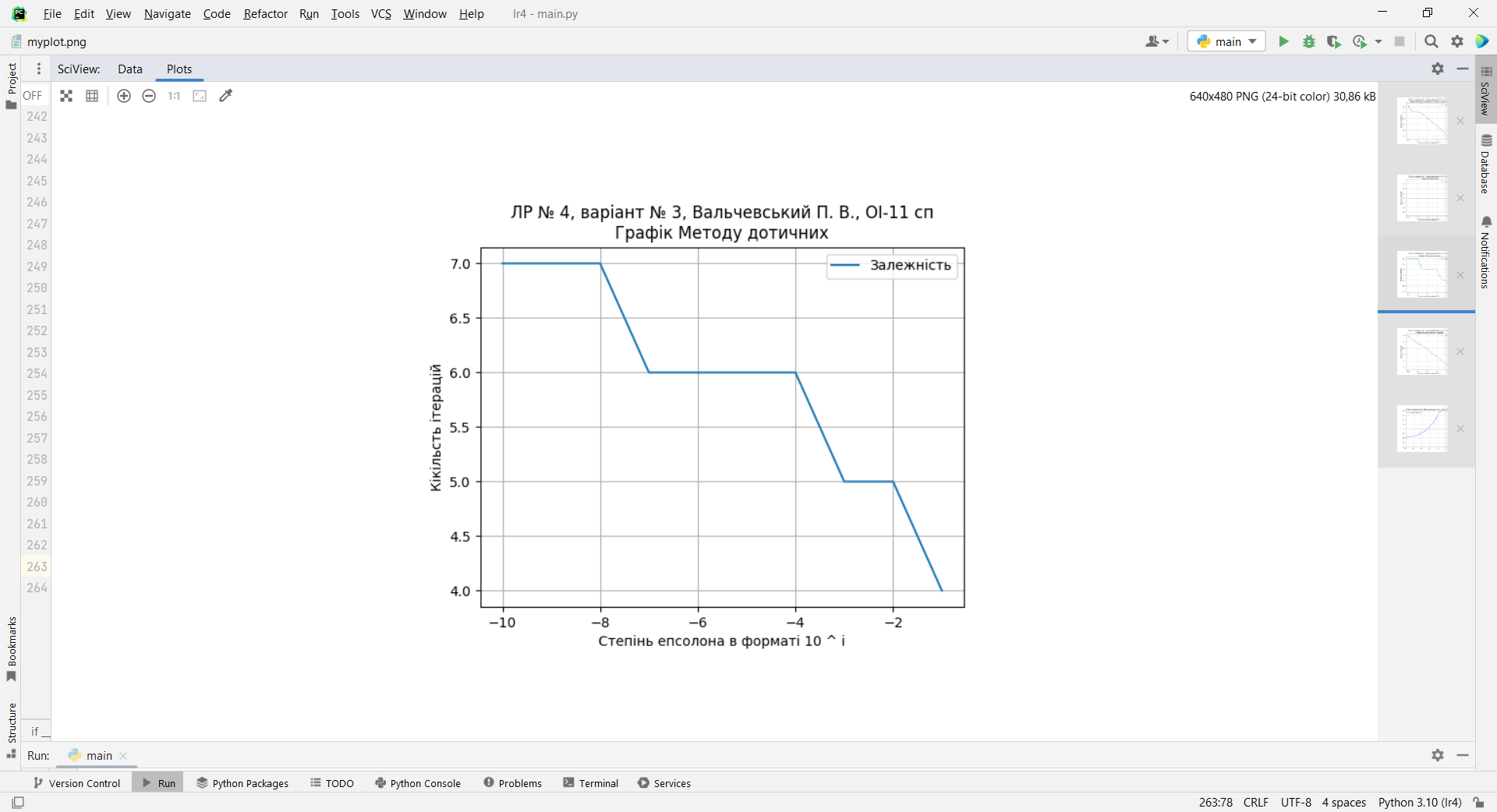


Рис. 10 Графік залежності для Методу дотичних (додаткове вікно програми).

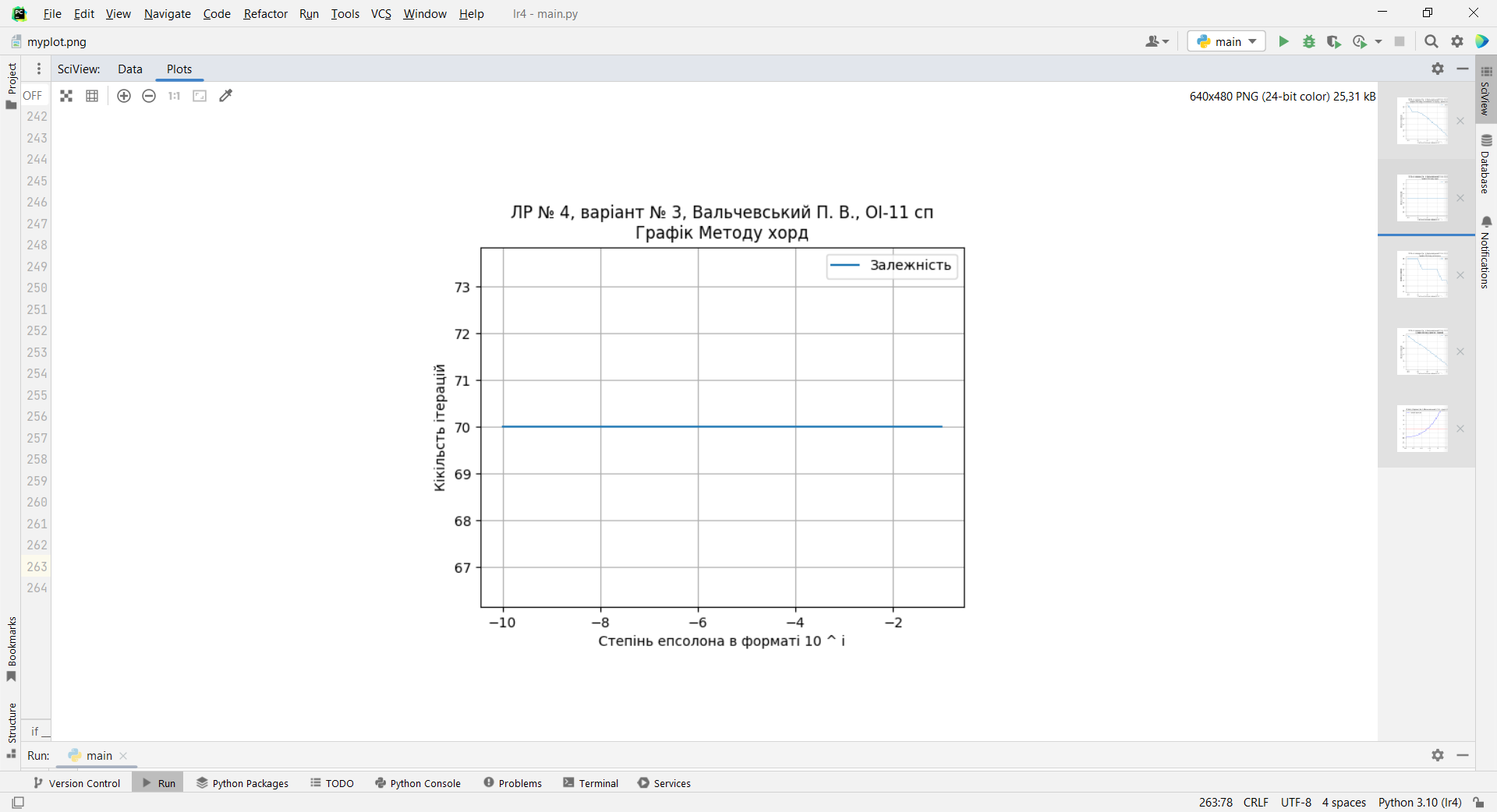


Рис. 11 Графік залежності для Методу хорд (додаткове вікно програми).

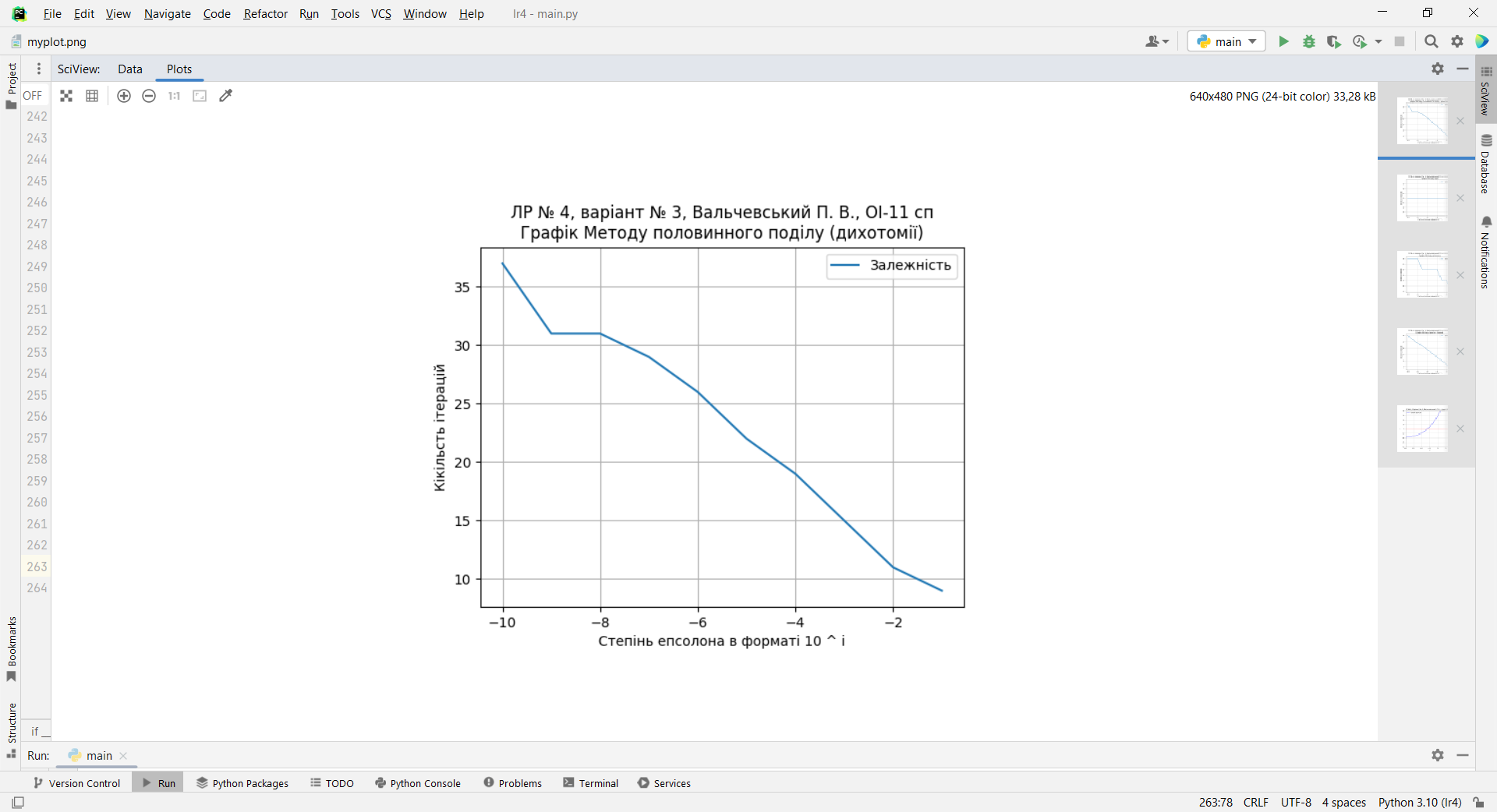


Рис. 12 Графік залежності для Методу половинного поділу (додаткове вікно програми).

1. **Висновки (з аналізом результату)**

Згідно результатів та порівняння з результатами розв’язаних вручну під час виконання програми – алгоритм виконується правильно та без помилок (з урахуванням виняткових ситуацій).

Реалізовуються умови для обчислення приблизного значення кореня рівняння на обраному мною проміжку [5; 8] при усіх методах. Усі знайдені корені від кожного методу відрізняються дробовими частинами (починаючи після заданої точності 10-5). Для перевірки й обчислення реального кореня та побудови графіку функції на вище згаданому boпроміжку з цим коренем було використано додаткові бібліотеки, які мають відповідний пояснювальний коментар.

Кожний метод має відповідну таблицю з списком усіх ітерацій та умов для подальших ітерацій. У кінці буде таблиця з усіма результатами кожного методу. Зовнішній вигляд було створено за допомогою бібліотеки, яка має відповідний коментар.

Для кожного методу було побудовано графік залежності від кількості ітерацій до епсолона (заданої точності) – використовувана бібліотека теж має коментар.

Загалом увесь код має відповідні коментарі та для кожного методу було створену власну блок схему.

Програма була відлагоджена й виконана у програмному середовищі PyCharm 2023.2.1 за допомогою мови програмування Python 3.10.4.