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 № 6

з дисципліни

“Чисельні методи”

на тему:

***«Ітерполяційна схема Ейткена»***

Виконав

студент групи *ОІ-11 сп*

*Вальчевський П. В.*

Викладач:

*Сенета М. Я.*

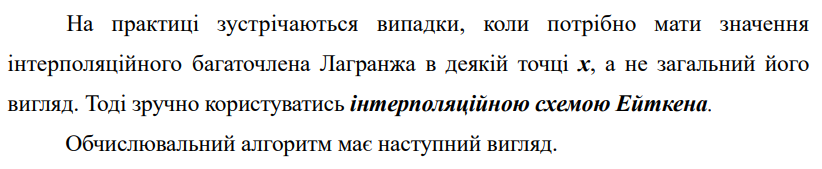
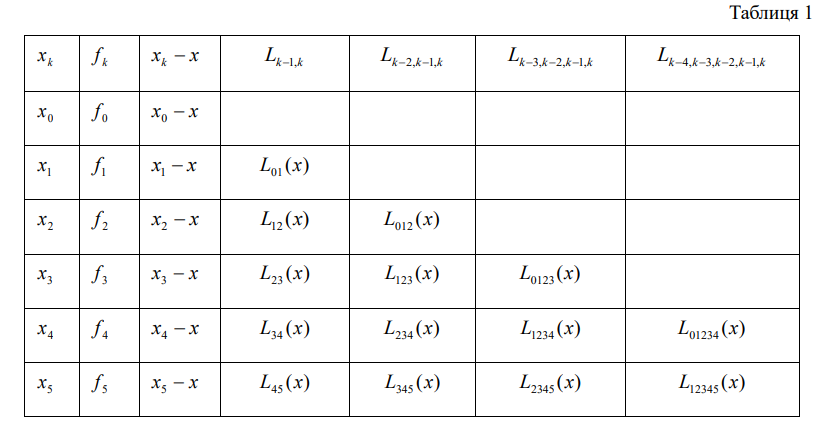
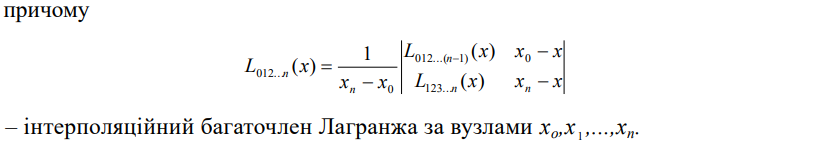
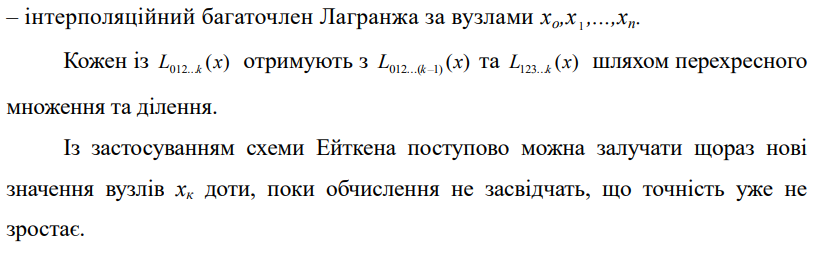
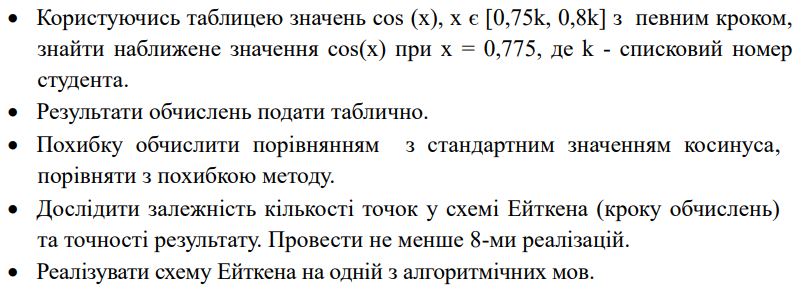
Львів – 2023

## Лабораторна робота № 6

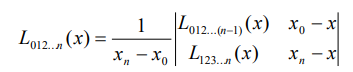
*Тема роботи:*  **Інтерполяційна схема Ейткена.**

*Мета роботи:* ***засвоїти теоретичний матеріал з теми апроксимації функцій, набути практичні навички знаходження наближених значень функцій.***

***Порядок виконання роботи***

1. **Основні теоретичні відомості**
2. ****
3. ****
4. ****
5. ****
6. **Номер варіанту – 3.**
7. **Умова завдання**
8. 
9. **Алгоримт розв’язання завдання**

* Алгоритм розв’язання інтерполяційною схемою Ейткена:
  1. Визначити точність, крок та межі х і у.
  2. Обрахувати усі значення х і у та зробити перші два стовпці у таблиці.
  3. Далі обрахувати третій стовпець за допомогою формули xn-x, де х шукане значення.
  4. Далі обчислюємо інтерполяційні багаточлени Лангранджа у вузлах х, доки обчислення не засвідачать, що точність не зростає. Многочлен можна зробити за формулою:



* Щодо програмної реалізації слід:
  1. Спроектувати приблизний вигляд програмної системи;
  2. Обрати підходящу мову програмування для реалізації поставленої задачі (Python);
  3. Реалізувати програмну систему на обраній мові:
     + Створити методи виводу;
     + Зобразити графіки;
     + Забезпечити виконання потрібних розрахунків для роботи системи;
  4. Провести тестування розробленої програмної системи та виправити помилки виявлені під час тестування;
  5. Сформувати програму виконання для використання створеної програмної системи.
  6. Провести тестування програми виконання та виправити помилки при їх наявності.

**Виконане завдання вручну**

Складаю таблицю для обчислення з кроком h = 0.01 і значенням х = 0,775 й перевіряю доки задачана точність не зростає з формули у теоретичній вказівці.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Xn | Xn-X | Yn | L1 | L2 | L3 |
| 0,753 | -0,022 | 0,72964 |  |  |  |
| 0,763 | -0,012 | 0,72277 | 0,71453 |  |  |
| 0,773 | -0,002 | 0,71582 | 0,71512 | 0,71518 |  |
| 0,783 | 0,008 | 0,7088 | 0,71067 | 0,71186 | 0,71275 |
| 0,793 | 0,018 | 0,70171 | 0,7049 | 0,7075 | 0,70946 |
| 0,803 | 0,028 | 0,69455 | 0,69856 | 0,70211 | 0,70513 |
| 0,813 | 0,038 | 0,68732 | 0,6919 | 0,69612 | 0,69991 |
| 0,823 | 0,048 | 0,68002 | 0,68503 | 0,68974 | 0,69411 |

Отже, знайдене значення рівне 0,71275 при кроці h=0,01, а табличне cos(0,775) = 0,7144. Похибка рівна 0,00165.

1. **Код програмної реалізації**

*Файл main.py*

from math import cos, fabs, log10 *# Математичні функції для обрахунку.*from tabulate import tabulate *# Створення таблиці в консолі.*import matplotlib.pyplot as plt *# Малювання графіку залежності.  
  
# Додати 1 стовпець (значення аргумента фукнції).*def appendX(xStart=float, xEnd=float, xStep=float, xFraction=int) -> list:  
 listX = []  
 xCounter = xStart  
 while True:  
 listX.append(round(xCounter, xFraction))  
 xCounter += xStep  
 if xCounter > xEnd:  
 return listX  
  
*# Додати 2 стовпець (табличні значення функції).*def appendFx(listX=list, xFraction=int) -> list:  
 return [round(cos(x), xFraction) for x in listX]  
  
*# Додати 3 стовпець (дельта: xn - x).*def appendDelta(listX=list, xValue=float, xFraction=int) -> list:  
 return [round(listX[i] - xValue, xFraction) for i in range(len(listX))]  
  
*# Повернення першого многочлена Лангранджа.*def getFirstLn(listFx=list, xValue=float) -> float:  
 for i in range(len(listFx)):  
 if (xValue > listFx[i]):  
 return listFx[i - 1]  
  
*# Долучення многочленів Лангранджа, списку ітерацій, кількості ітерацій, знайдене значення.*def appendColumns(listBaseData=list, xValue=float, xFraction=int, eps=float) -> list:  
 listStr = []  
 counterItearation = 0  
 listLn = []  
 saveLn = getFirstLn(listBaseData[-1], xValue)  
 for i in range(len(listBaseData[0]) - 2):  
 listIteration = []  
 listStrIteration = []  
 for j in range(i + 1):  
 listIteration.append(" " \* 15)  
 for j in range(i + 1, len(listBaseData[-1])):  
 x0 = listBaseData[0][0]  
 xN = listBaseData[0][j]  
 xDelta0 = listBaseData[1][0]  
 xDeltaN = listBaseData[1][j]  
 xDelta = xN - x0  
 L0 = listBaseData[-1][j - 1]  
 Ln = listBaseData[-1][j]  
 Lnew = (L0 \* xDeltaN - Ln \* xDelta0) / xDelta  
 LnewStr = f"{round(Lnew, xFraction)} = ({round(L0, xFraction)} \* {round(xDeltaN, xFraction)} - {round(Ln, xFraction)} \* {round(xDelta0, xFraction)}) / ({round(xN, xFraction)} - {round(x0, xFraction)})"  
 listStrIteration.append(LnewStr)  
 listIteration.append(round(Lnew, xFraction))  
 counterItearation += 1  
 if j == i + 1:  
 listLn.append(listIteration[-1])  
 epsPow = int(-log10(eps))  
 if i > 0 and fabs(round(listLn[-1], epsPow) - round(listLn[i - 1], epsPow)) < eps:  
 saveLn = listLn[i - 1]  
 break  
 listBaseData.append(listIteration)  
 listStr.append(listStrIteration)  
 listBaseData.append(listStr)  
 listBaseData.append(counterItearation)  
 listBaseData.append(saveLn)  
  
*# Створити контент для таблиці (перевернути многочлени Лангранджа у транспортованому вигляді).*def createTableContent(dataMethod=list) -> list:  
 lengthContent = len(dataMethod) - 3  
 dataContent = []  
 for i in range(lengthContent):  
 dataContent.append(dataMethod[i].copy())  
 return [[row[i] for row in dataContent] for i in range(len(dataContent[0]))]  
  
*# Повернення таблиці методу.*def getTableIterationOfMethod(dataMethod=list) -> str:  
 nameTable = "Таблиця алгоритма Ейткена:"  
 headersTable = [  
 "x",  
 "xn - x",  
 "F(x)"  
 ]  
 strLn = "L01"  
 counterLn = 2  
 for i in range(3, len(dataMethod) - 3):  
 headersTable.append(strLn + "n")  
 strLn += str(counterLn)  
 counterLn += 1  
 tableContent = createTableContent(dataMethod)  
 strTable = tabulate(tableContent, headersTable, tablefmt="pretty")  
 return f"{nameTable}\n{strTable}"  
  
*# Повернення списку обчислення кожного многочлена Лангранджа.*def getListIterationCalculateLn(dataMethod=list) -> str:  
 nameListLn = []  
 strLn = "L01"  
 counterLn = 2  
 for i in range(3, len(dataMethod) - 3):  
 nameListLn.append(strLn + "n")  
 strLn += str(counterLn)  
 counterLn += 1  
 strReturn = f"Обчислення кожного многочлена Лангранджа:\n"  
 counter = 0  
 for i in dataMethod[len(dataMethod) - 3]:  
 strReturn += f"\tОбрахунок {nameListLn[counter]}:\n"  
 counterEquals = 1  
 for j in i:  
 strReturn += f"\t\t{counterEquals}) {j}\n"  
 counterEquals += 1  
 counter += 1  
 return strReturn  
  
*# Кількість ітерацій при заданій точності (кроку).*def getCountInteration(xStart=float, xEnd=float, xStep=float, xFraction=int) -> int:  
 calcData = []  
 calcData.append(appendX(xStart, xEnd, xStep, xFraction))  
 calcData.append(appendDelta(calcData[0], xValue, xFraction))  
 calcData.append(appendFx(calcData[0], xFraction))  
 appendColumns(calcData, xValue, xFraction, xStep)  
 return calcData[len(calcData) - 2]  
  
*# Отримати похибку результату.*def getDeltaXandResult(xValue=float, xResult=float) -> float:  
 return fabs(cos(xValue) - xResult)  
  
*# Результат при заданій точності (кроку).*def getResultX(xStart=float, xEnd=float, xStep=float, xFraction=int) -> int:  
 calcData = []  
 calcData.append(appendX(xStart, xEnd, xStep, xFraction))  
 calcData.append(appendDelta(calcData[0], xValue, xFraction))  
 calcData.append(appendFx(calcData[0], xFraction))  
 appendColumns(calcData, xValue, xFraction, xStep)  
 return round(calcData[-1], xFraction)  
  
*# Демонстрація графіку залежності кількості ітерацій та точності (кроку).*def showGraphicMethodIterationAndEps(xStart=float, xEnd=float, xFraction=int, stepStart=float, stepEnd=float, stepDivision=int) -> None:  
 *# Значення для дефолту, щоб уникнути нескінченного циклу або довгого виконання програми.* if stepDivision <= 1 or not (stepStart <= 10 \*\* -2 and stepEnd >= 10 \*\* -4 and stepStart > stepEnd):  
 stepStart = 10 \*\* -2  
 stepEnd = 10 \*\* -4  
 listStep = []  
 counterStep = stepStart  
 while counterStep >= stepEnd:  
 listStep.append(counterStep)  
 counterStep /= stepDivision  
 countIterationInStep = [getCountInteration(xStart, xEnd, xStep, xFraction) for xStep in listStep]  
 fig, ax = plt.subplots(figsize=(8, 8)) *# Зміна розміру вікна.* plt.plot(listStep, countIterationInStep, label='Залежність')  
 plt.xlabel("Точність (крок)")  
 plt.ylabel("Кількість ітерацій")  
 plt.title(f"ЛР № 6, варіант № 3, Вальчевський П. В., ОІ-11 сп\nГрафік залежності кількості ітерацій та точності (кроку)\nКрок є в діапазоні: [{stepEnd}; {stepStart}] та має множник для зміни {stepDivision}")  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
*# Демонстрація графіку залежності похибки та точності (кроку).*def showGraphicMethodDeltaAndEps(xStart=float, xEnd=float, xValue=float, xFraction=int, stepStart=float, stepEnd=float, stepDivision=int) -> None:  
 *# Значення для дефолту, щоб уникнути нескінченного циклу або довгого виконання програми.* if stepDivision <= 1 or not (stepStart <= 10 \*\* -2 and stepEnd >= 10 \*\* -4 and stepStart > stepEnd):  
 stepStart = 10 \*\* -2  
 stepEnd = 10 \*\* -4  
 listStep = []  
 counterStep = stepStart  
 while counterStep >= stepEnd:  
 listStep.append(counterStep)  
 counterStep /= stepDivision  
  
 calcDelta = [getDeltaXandResult(xValue, getResultX(xStart, xEnd, xStep, xFraction)) for xStep in listStep]  
 fig, ax = plt.subplots(figsize=(8, 8)) *# Зміна розміру вікна.* plt.plot(listStep, calcDelta, label='Залежність')  
 plt.xlabel("Точність (крок)")  
 plt.ylabel("Похибка")  
 plt.title(f"ЛР № 6, варіант № 3, Вальчевський П. В., ОІ-11 сп\nГрафік залежності похибки та точності (кроку)\nКрок є в діапазоні: [{stepEnd}; {stepStart}] та має множник для зміни {stepDivision}")  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
*# Програма виконання.*if \_\_name\_\_ == '\_\_main\_\_':  
 dataMethod = []  
 xValue, xStart, xEnd, xStep, xFraction, xFractionGraphic = 0.775, 0.753, 0.83, 0.002, 5, 15  
 stepStart, stepEnd, stepDivision = 10 \*\* -2, 10 \*\* -4, 2  
 print("Програму розробив Вальчевський П. В., студент групи ОІ-11 сп для ЛР № 6, варіанту № 3 з дисципліни Чисельні методи.")  
 print(f"\t\*Проміжок [{xStart}; {xEnd}]; крок або точність (h): {xStep}; шукане значення х: {xValue}; заокруглення до {xFraction} чисел після коми; функція: F(x) = cos(x).")  
 print("\t\*Графіки функцій є в інших вікнах програми.")  
 dataMethod.append(appendX(xStart, xEnd, xStep, xFraction))  
 dataMethod.append(appendDelta(dataMethod[0], xValue, xFraction))  
 dataMethod.append(appendFx(dataMethod[0], xFraction))  
 appendColumns(dataMethod, xValue, xFraction, xStep)  
 print(getTableIterationOfMethod(dataMethod))  
 print(f"Знайдене значення: {dataMethod[-1]}")  
 print(f"Табличне значення: {cos(xValue)}")  
 print(f"Похибка: {getDeltaXandResult(xValue, dataMethod[-1])}")  
 print(f"Кількість ітерацій (кількість обрахунків): {dataMethod[len(dataMethod) - 2]}")  
 print(getListIterationCalculateLn(dataMethod))  
 showGraphicMethodIterationAndEps(xStart, xEnd, xFractionGraphic, stepStart, stepEnd, stepDivision)  
 showGraphicMethodDeltaAndEps(xStart, xEnd, xValue, xFractionGraphic, stepStart, stepEnd, stepDivision)

1. **Скріншот отриманого результату**

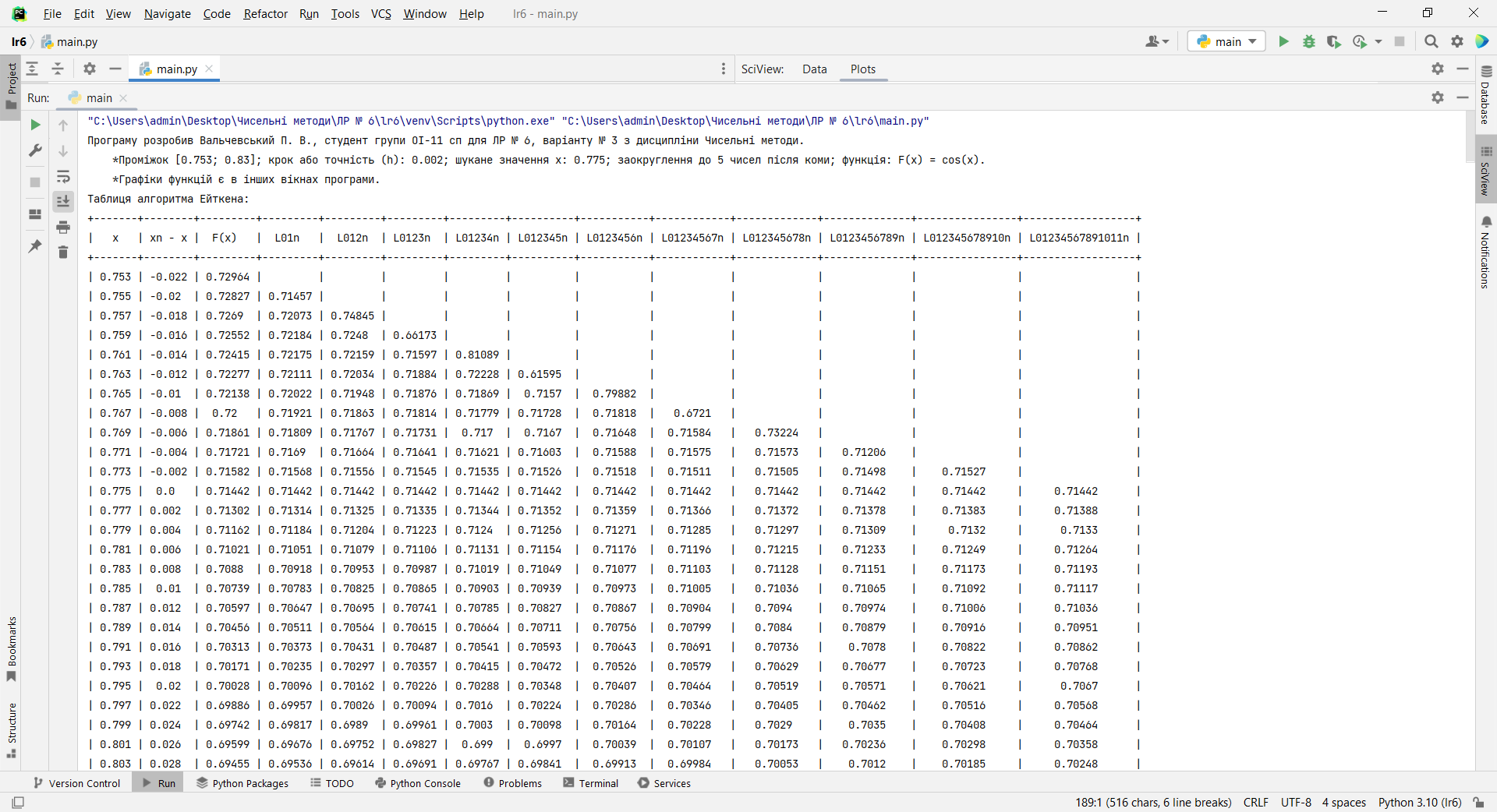


Рис. 1 Результат виконання у консолі.

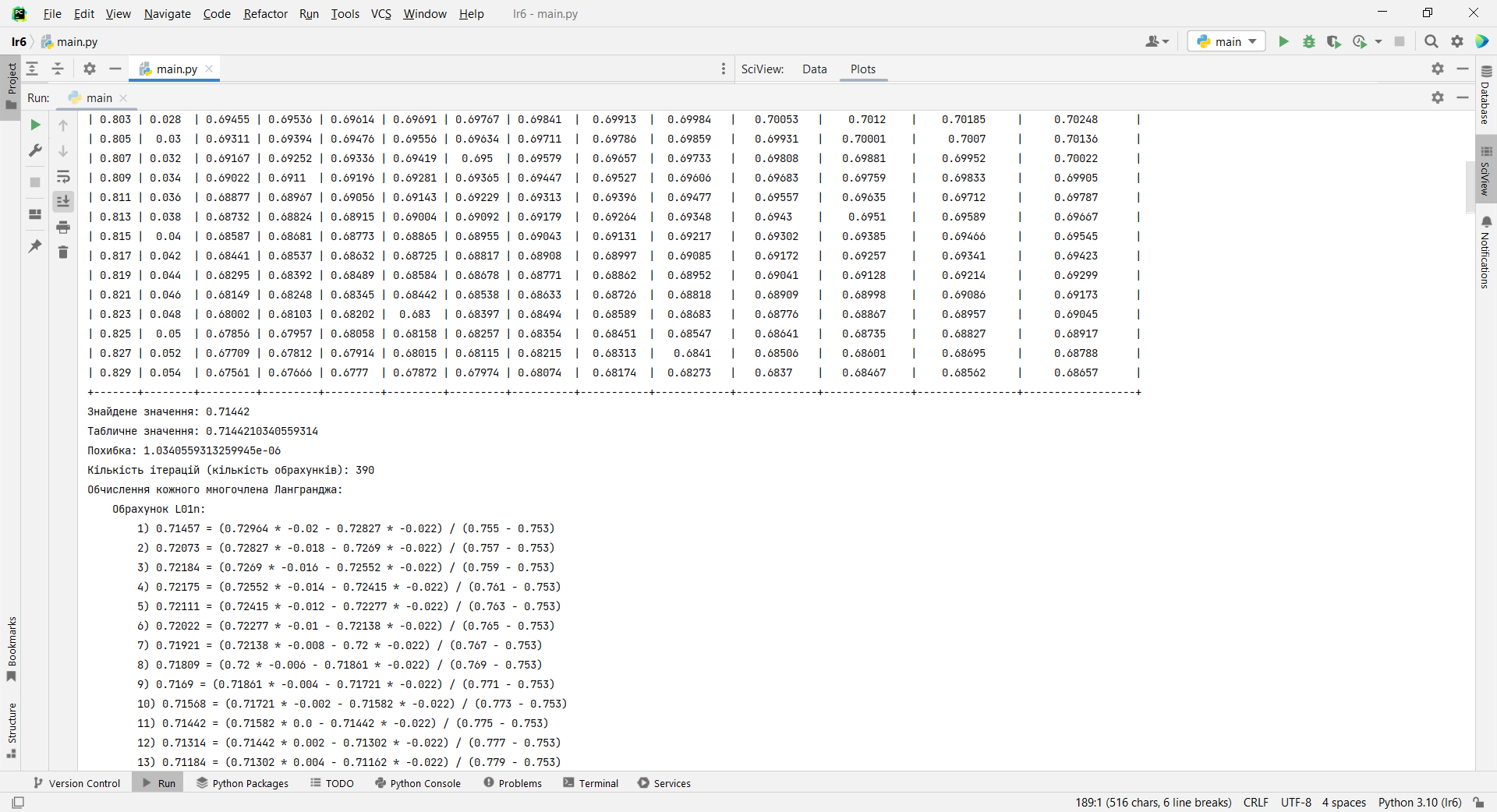


Рис. 2 Результат виконання у консолі (продовження).

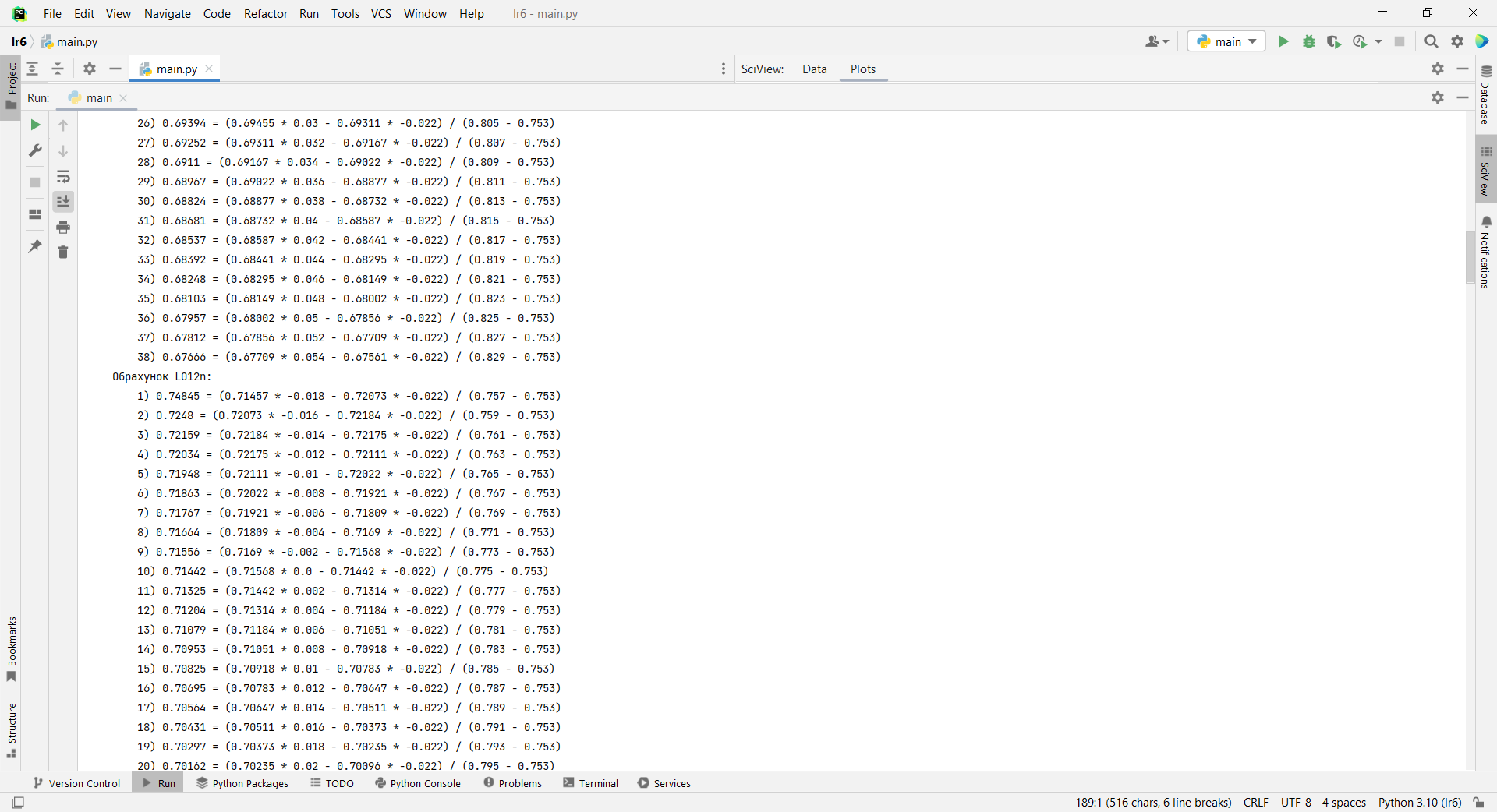


Рис. 3 Результат виконання у консолі (продовження).

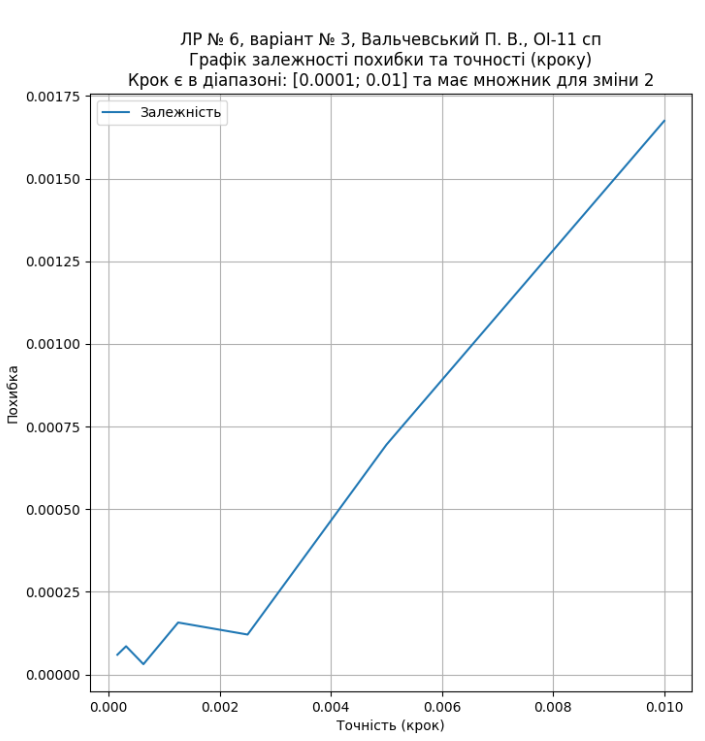
1. Подальший вивід обрахунку многочленів немає сенсу, оскільки формула буде коректно працювати (як зображено), як раніше наданих результатах.
2. 

Рис. 4 Графік залежності похибки та точності (кроку).

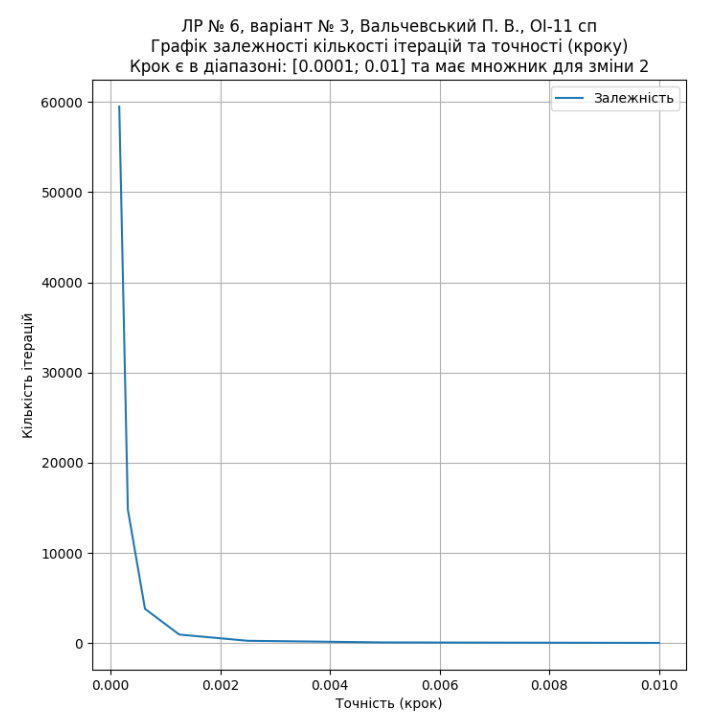


Рис. 5 Графік залежності кількості ітерацій та точності (кроку).

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

Згідно результатів та порівняння отриманих під час виконання програми – алгоритм виконується правильно та без помилок.

Реалізовується алгоритм обчислення інтерполяції Ейткена та йде вивід подальших розрахунків у вигляді таблиці.

Було побудовано графік залежності від кількості ітерацій до заданої точності (кроку) та похибки до кроку – використовувана бібліотека теж має коментар у коді.

Загалом увесь код має відповідні коментарі до кожної ключової чи визначної дії, яка впливає на подальші розрахунки.

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