НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

ЛАБОРАТОРНА РОБОТА №1

з дисципліни «Проектування та аналіз обчислювальних алгоритмів»

Варіант № 38

Виконав:
Студент ІІ курсу
Групи КІ-32
Присяжнюк Владислав
Прийняв:
Тимошенко Ю. О.

Київ 2024

Варіант 38

Завдання:

x=−8.0,−7.5,−7.0,−6.5,−6.0,−5.5,−5.0,−4.5,−4.0,−3.5,−3.0,−2.5,−2.0,
−1.5,−1.0,−0.5,0.0,0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.
0,7.5,8.0

y=2.5,2.0,1.5,1.0,0.0,−1.0,−1.5,−2.0,−1.5,−1.0,0.0,1.0,2.0,3.0,3.5,3.0,2.5,1.5
,1.0,1.5,2.5,3.0,4.0,5.0,6.0,6.5,6.0,5.5,5.0,4.5,4.0,3.5,3.0

Застосувати метод дихотомії, метод золотого перерізу, метод ньоютона.

Хід Роботи:

Спочатку побудуємо таблицю значень.

X	Y
-8.0	2.5
-7.5	2.0
-7.0	1.5
-6.5	1.0
-6.0	0.0
-5.5	-1.0
-5.0	-1.5
-4.5	-2.0
-4.0	-1.5
-3.5	-1.0
-3.0	0.0
-2.5	1.0
-2.0	2.0
-1.5	3.0
-1.0	3.5
-0.5	3.0
0.0	2.5
0.5	1.5

1.0	1.0
1.5	1.5
2.0	2.5
2.5	3.0
3.0	4.0
3.5	5.0
4.0	6.0
4.5	6.5
5.0	6.0
5.5	5.5
6.0	5.0
6.5	4.5
7.0	4.0
7.5	3.5
8.0	3.0

За допомогою мови програмування Python та бібліотеки для візуалізації даних matplotlib побудуємо графік.

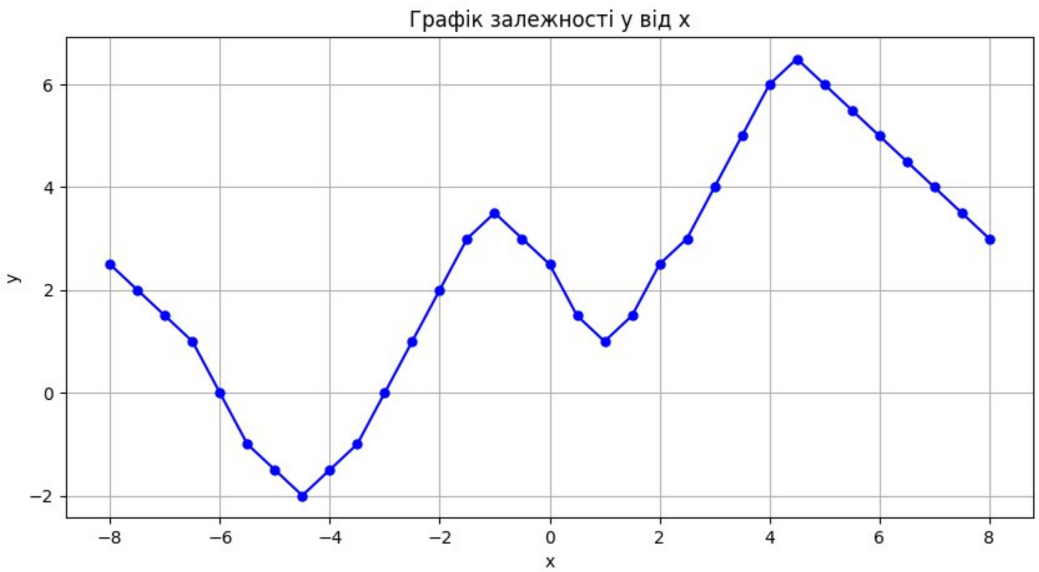


Рис 1.1 – Побудований графік бібліотекою matplotlib

Використаємо поліноміальну регресію для знаходження полінома певного степеня, щоб описати залежність між x та y у тих координатах, які не покривають надані дані.

Для цього використаємо мову програмування Python, та бібліотеки NumPy, MatPlotLib, SkLearn (для готових моделей поліноміальної та лінійної регресії).

Спочатку напишемо програму яка буде генерувати поліноміальну ознаку для поточного степеня, обчислювати коефіцієнт детермінації R^2, та зберегати найкращий степень для порівняння.

Код розробленої програми на мові Python:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Дані
x = np.array([-8.0, -7.5, -7.0, -6.5, -6.0, -5.5, -5.0, -4.5, -4.0, -3.5,
              -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0,
              2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0]).reshape(-1, 1)

y = np.array([2.5, 2.0, 1.5, 1.0, 0.0, -1.0, -1.5, -2.0, -1.5, -1.0,
              0.0, 1.0, 2.0, 3.0, 3.5, 3.0, 2.5, 1.5, 1.0, 1.5, 2.5,
              3.0, 4.0, 5.0, 6.0, 6.5, 6.0, 5.5, 5.0, 4.5, 4.0, 3.5, 3.0])

# Перебір степенів полінома від 1 до 10
best_degree = 1
best_r2 = -np.inf  # Мінімальне значення R^2 для початку
r2_scores = []

for degree in range(1, 11):
```

```
    # Генерація поліноміальних ознак для поточного степеня
    poly = PolynomialFeatures(degree=degree)
    x_poly = poly.fit_transform(x)

    # Побудова моделі та навчання
    model = LinearRegression()
    model.fit(x_poly, y)

    # Прогнозування та обчислення R^2
    y_pred = model.predict(x_poly)
    r2 = r2_score(y, y_pred)
    r2_scores.append(r2)

    print(f"Степінь: {degree}, R^2: {r2}")

    # Збереження найкращого степеня
    if r2 > best_r2:
        best_r2 = r2
        best_degree = degree


# Побудова графіка R^2 для кожного степеня
plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), r2_scores, marker='o', linestyle='-')
plt.title("Коефіцієнт детермінації R^2 для різних степенів полінома")
plt.xlabel("Степінь полінома")
plt.ylabel("R^2")
plt.grid(True)
plt.savefig('best_degree.png')
```

```
~/Documents python PAOA1_BestDegree.py
Степінь: 1, R^2: 0.5267126247050544
Степінь: 2, R^2: 0.5353775746751903
Степінь: 3, R^2: 0.7825626590815891
Степінь: 4, R^2: 0.7832763152678363
Степінь: 5, R^2: 0.8160047240173773
Степінь: 6, R^2: 0.8182255004055468
Степінь: 7, R^2: 0.9386082189546752
Степінь: 8, R^2: 0.938709744763925
Степінь: 9, R^2: 0.988642048707327
Степінь: 10, R^2: 0.9886679921003513
```

Рис 1.2 – Результат роботи розробленої програми

Тепер переведемо результати в графік для кращого розуміння.

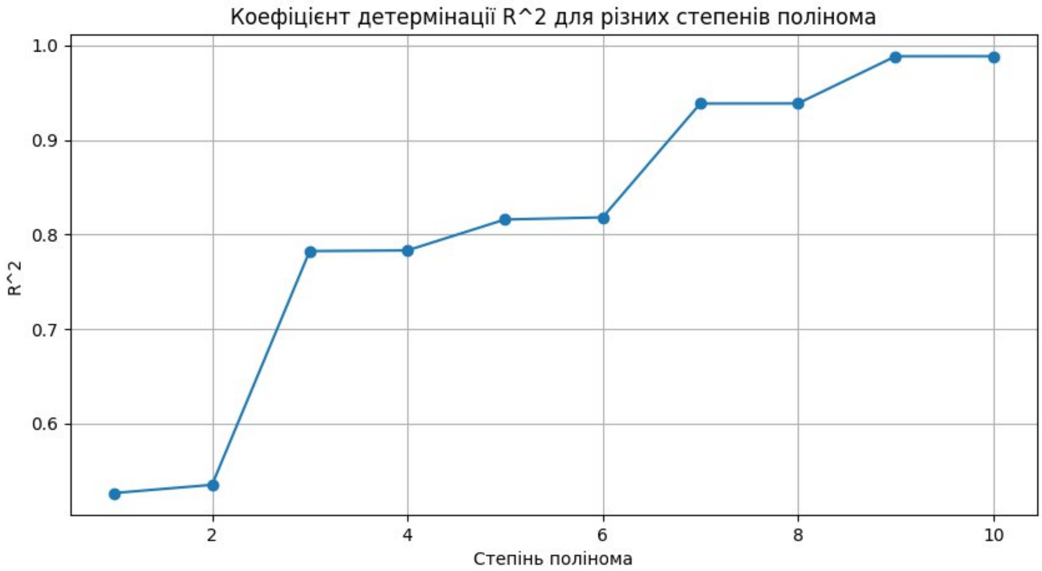


Рис 1.3 – Графік результатів роботи розробленої програми

Як видно на графіку коефіцієнт детермінації набуває найкращого рівня на 9 степені. Тож для наступних дій будемо використовувати саме його.

За допомогою ще однієї розробленої програми отримаємо коефіцієнти полінома та порівняємо на графіку його відповідність з наданими даними.

```
Поліноміальне рівняння (ступінь 9):
2.33 + (-0.94) * x^1 + (-0.05) * x^2 + (0.27) * x^3 + (0.00) * x^4 + (-0.01) * x^5 + (-0.00) *
 x^6 + (0.00) * x^7 + (0.00) * x^8 + (-0.00) * x^9
```

Рис 1.4 – Результат роботи програми

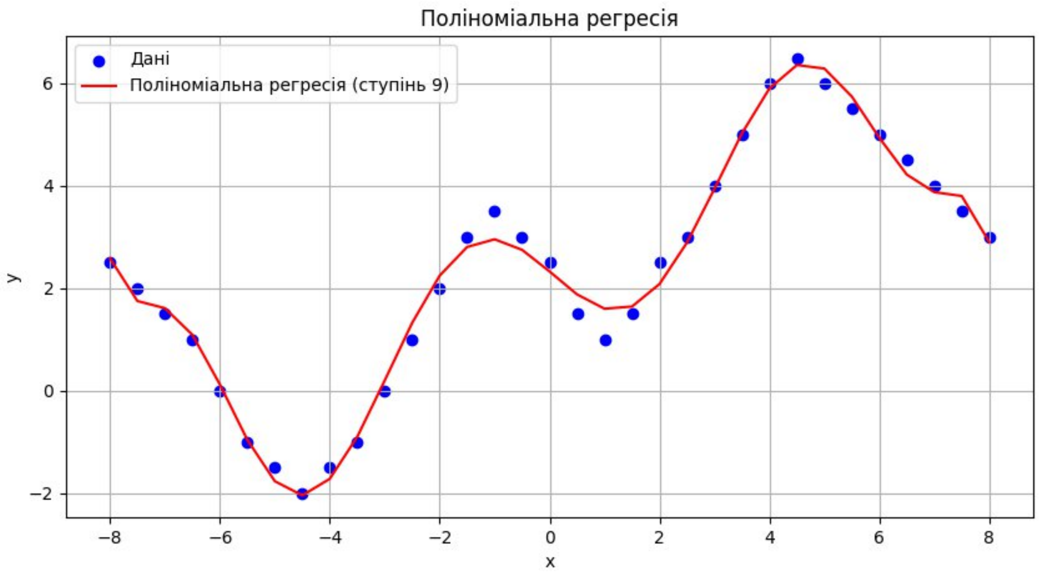


Рис 1.5 – Графік порівняння отриманого полінома з наданими даними

Отриманий поліном:

$$P(x) = 2.33 + (-0.94) \cdot x + (-0.05) \cdot x^2 + 0.27 \cdot x^3 + 0.00 \cdot x^4 - 0.01 \cdot x^5 - 0.00 \cdot x^6 + 0.00 \cdot x^7 + 0.00 \cdot x^8 - 0.00 \cdot x^9$$

Після отримання поліному можемо почати застосовувати перший метод, метод дихотомії.

Метод дихотомії

Метод дихотомії полягає в послідовному поділу інтервалу навпіл та вибору тієї його частини, де може знаходитися шуканий мінімум. Спочатку обирається початквий інтервал [a, b], дві внутрішні точки визначаються симетрично щодо середини інтервалу:

$$x_1 = a + \frac{b - a}{2} - \delta, \quad x_2 = a + \frac{b - a}{2} + \delta$$

де δ - мале значення, щоб уникнути точного ділення на половину.

Наступний крок це порівняння значень функції:

- Якщо $f(x_1) < f(x_2)$, це означає, що мінімум знаходиться в лівій частині, і новий інтервал буде [a, x_2]
- Якщо $f(x_2) < f(x_1)$, це означає, що мінімум знаходиться в лівій частині, і новий інтервал буде $[x_1, b]$

Процес повторюється, поки довжина інтервалу не стане меншою за ε.

Метод працює за принципом унімодальності, тобто якщо функція має лише один мінімум на інтервалі, тоді в кожній половині можна визначити, де знаходиться спад або підйом функції. Унімодальність — це властивість функції мати один локальний екстремум (мінімум або максимум) на певному інтервалі.

Вже зараз можна зрозуміти, що метод є ітераційним, отож головна функція програми міститиме ітерацію, яка триватиме до досягнення заданої точності. Давайте розробимо псевдокод для кращого розуміння алгоритму:

```
FUNCTION DichotomyMethod(f, a, b, tol):
   WHILE (b - a) / 2 > tol DO:
      mid = (a + b) / 2
      delta = tol / 2  # Малий зсув для обчислень

      x1 = mid - delta
      x2 = mid + delta

      IF f(x1) < f(x2) THEN:
         b = x2  # Мінімум лежить у лівій частині
      ELSE:
         a = x1  # Мінімум лежить у правій частині

   RETURN (a + b) / 2  # Наближене значення мінімуму
```

Після аналізу алгоритму та отримання загального уявлення про майбутній вигляд програми перейдемо до розробки програми та обробки результатів.

Готова програма на мові Python:

```
def dichotomy_method_min(f, a, b, tol=0.0001):
   """
```

```
    Метод дихотомії для знаходження мінімуму функції f на інтервалі [a, b].

    f - цільова функція
    a, b - початковий інтервал
    tol - точність
    """
    while (b - a) / 2 > tol:
        mid = (a + b) / 2
        delta = tol / 2  # Маленьке значення для порівняння

        # Дві точки, близькі до середини
        x1 = mid - delta
        x2 = mid + delta

        # Обчислення значень функції у цих точках
        f1 = f(x1)
        f2 = f(x2)

        # Вибір нового інтервалу
        if f1 < f2:
            b = x2  # Мінімум знаходиться в лівій частині
        else:
            a = x1  # Мінімум знаходиться в правій частині

    # Повертаємо наближене значення мінімуму
    return (a + b) / 2

# Поліном з вашого зображення
def polynomial(x):
    return (2.33 + (-0.94) * x + (-0.05) * x**2 + 0.27 * x**3 +
            0.00 * x**4 - 0.01 * x**5 - 0.00 * x**6 +
```

```
        0.00 * x**7 + 0.00 * x**8 - 0.00 * x**9)

# Застосування методу дихотомії для пошуку мінімуму
a, b = -5, -1  # Інтервал пошуку
min_x = dichotomy_method_min(polynomial, a, b)
print(f"Мінімум функції досягається в точці: x = {min_x: .4f}, f(x) =
{polynomial(min_x): .4f}")
```

Перевіримо роботоздатність розробленої програми, спочатку оберемо відносно великий інтеграл [-5, 4]:

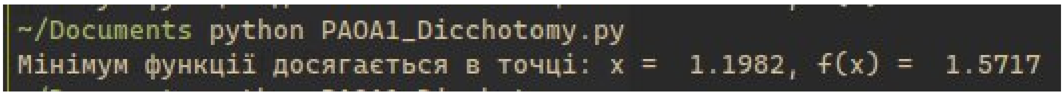


Рис 1.6 – Результат роботи програми с заданим інтервалом [-5, 4]

Тепер давайте перевіримо чи це правильний мінімум на за допомогою графіку полінома:

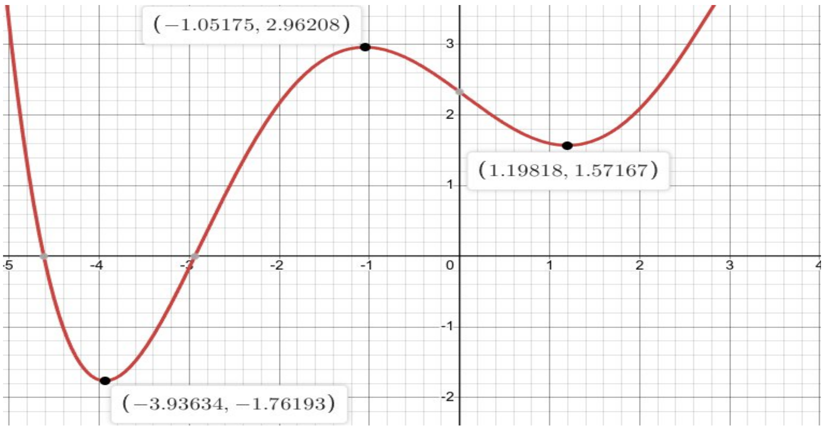


Рис 1.7 – Графік полінома побудований за допомогою інструмента desmos

Як зображено на графіку це і справді один з мінімумів полінома на цьому відрізку, Однак, якщо є кілька локальних мінімумів, метод дихотомії знайде лише той, який знаходиться ближче до початкової середини або в межах заданого інтервалу пошуку. Якщо є необхідність знаходження декількох локальних мінімумів то слід використотати інший метод.

Метод золотого перерізу

Метод золотого перерізу є більш ефективним варіантом пошуку мінімуму в порівнянні з методом дихотомії, оскільки він зменшує кількість необхідних обчислень функції, використовуючи співвідношення золотого перетину.

- Співвідношення золотого перетину використовується для вибору двох внутрішніх точок у межах інтервалу [a,b], щоб зменшити інтервал пошуку.
- Порівнюються значення функції в цих точках, і новий інтервал вибирається на основі того, де знаходиться мінімум.
- Цей процес триває, доки інтервал не зменшиться до значення меншого за задану точність.

Метод більш ефективний і швидший, особливо для знаходження мінімуму на довгих інтервалах. Основна відмінність від методу дихотомії полягає в тому, що на кожній ітерації обчислюються значення функції у двох внутрішніх точках інтервалу, які обираються за принципом золотого перетину φ (приблизно 0.618). Це гарантує, що інтервал зменшується оптимально, при цьому одна з точок залишається незмінною, а лише одна нова точка перераховується на кожній ітерації.
Код розробленої програми на мові Python:

```
import math


def golden_section_min(f, a, b, tol=0.0001):
    """
    Метод золотого перетину для знаходження мінімуму функції f на інтервалі
[a, b].

    f - цільова функція
    a, b - початковий інтервал
    tol - точність
    """
    golden_ratio = (math.sqrt(5) - 1) / 2  # Співвідношення золотого перетину

    # Початкові дві точки всередині інтервалу
    x1 = b - golden_ratio * (b - a)
    x2 = a + golden_ratio * (b - a)
```

```
    # Обчислення значень функції в цих точках
    f1 = f(x1)
    f2 = f(x2)

    while (b - a) > tol:
        if f1 < f2:
            b = x2
            x2 = x1
            f2 = f1
            x1 = b - golden_ratio * (b - a)
            f1 = f(x1)
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = a + golden_ratio * (b - a)
            f2 = f(x2)

    # Повертаємо наближене значення мінімуму
    return (a + b) / 2

# Поліном з вашого зображення
def polynomial(x):
    return (2.33 + (-0.94) * x + (-0.05) * x**2 + 0.27 * x**3 +
            0.00 * x**4 - 0.01 * x**5 - 0.00 * x**6 +
            0.00 * x**7 + 0.00 * x**8 - 0.00 * x**9)

# Застосування методу золотого перерізу для пошуку мінімуму
a, b = -5, 4  # Інтервал пошуку
min_x = golden_section_min(polynomial, a, b)
```

```
print(f"Мінімум функції досягається в точці: x = {min_x: .4f}, f(x) = {polynomial(min_x): .4f}")
```

Тепер обчислимо наш поліном на такому ж інтервалі [-5, 4]:

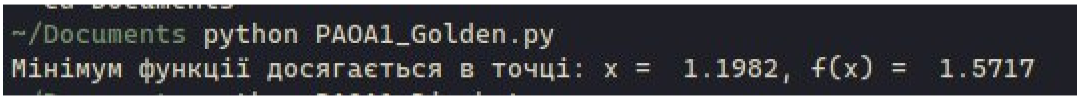


Рис 1.8 – Результат роботи програми з інтервалом [-5, 4]

Як видно с результату роботи програми метод знаходить таку саму точку мінімуму як і в попередньому методі.

Метод Ньютона

Метод Ньютона є ефективним чисельним методом для мінімізації поліномів та інших функцій. Його основна перевага — швидка збіжність, особливо в околі розв'язку, але він вимагає обчислення першої та другої похідних функції, що може бути складним для деяких функцій або поліномів.

Метод Ньютона використовується для чисельного розв'язання нелінійних рівнянь, тобто для знаходження коренів рівняння, а також для мінімізації та максимізації функцій.

Використовується ітеративний процес для знаходження кореня функції, де нове значення x_{n+1} обчислюється як:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ітерації зупиняються, коли значення функції стає досить малим, або коли перевищується максимальна кількість ітерацій.

Спочатку обрахуємо похідну полінома за допомогою мови Python та математичної бібліотеки SymPy:

```
import sympy as sp
x = sp.symbols('x')

f = (2.33 + (-0.94) * x + (-0.05) * x**2 + 0.27 * x**3 +
     0.00 * x**4 - 0.01 * x**5 - 0.00 * x**6 +
     0.00 * x**7 + 0.00 * x**8 - 0.00 * x**9)
```

```
f_prime = sp.diff(f, x)

print(f"Похідна полінома:",f_prime )
```

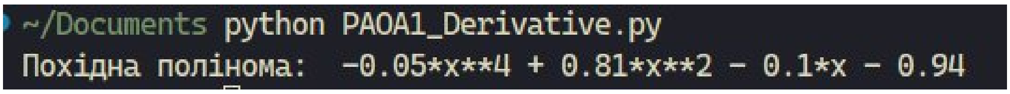


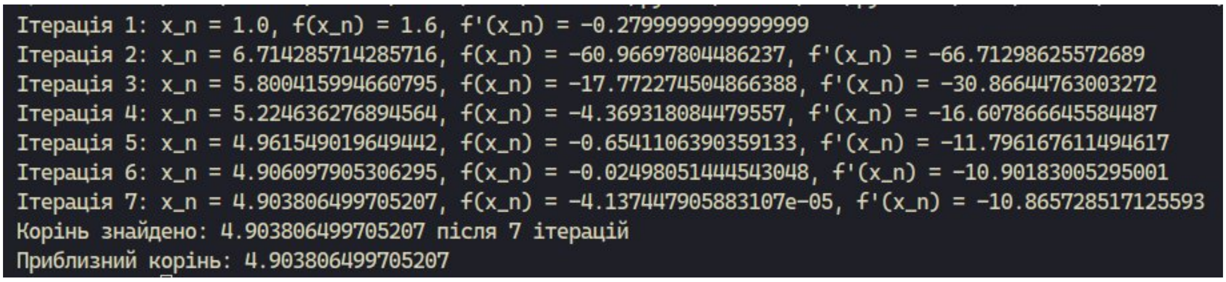
Рис 1.9 – Реузльат обрахунку похідної

Тепер розробимо код для метода Ньютона та підставимо нашу похідну.

```
import sympy as sp

# Визначаємо змінну x
x = sp.symbols('x')

# Визначаємо початкову функцію
f = 2.33 + (-0.94) * x + (-0.05) * x**2 + 0.27 * x**3 + 0.00 * x**4 - 0.01 * x**5

# Знаходимо похідну функції
f_prime = -0.05*x**4 + 0.81*x**2 - 0.1*x - 0.94

# Конвертуємо символічну функцію та її похідну в функції для числових розрахунків
f_func = sp.lambdify(x, f)
f_prime_func = sp.lambdify(x, f_prime)

# Реалізуємо метод Ньютона
def newtons_method(f, f_prime, x0, tol=0.01, max_iter=100):
    x_n = x0
    for n in range(max_iter):
        f_x_n = f(x_n)
        f_prime_x_n = f_prime(x_n)
```

```
        # Виведення поточної ітерації
        print(f"Ітерація {n+1}: x_n = {x_n}, f(x_n) = {f_x_n}, f'(x_n) = {f_prime_x_n}")

        if abs(f_x_n) < tol:
            print(f"Корінь знайдено: {x_n} після {n+1} ітерацій")
            return x_n

        if f_prime_x_n == 0:
            print("Нульова похідна. Розв'язок не знайдено.")
            return None

        x_n = x_n - f_x_n / f_prime_x_n

    print("Перевищено кількість ітерацій. Розв'язок не знайдено.")
    return None

# Початкова здогадка
x0 = 1.0

# Виклик методу Ньютона
root = newtons_method(f_func, f_prime_func, x0)
print(f"Приблизний корінь: {root}")
```

Початкове наближення було обрано у вигляді 1, тепер поглянемо на результат:



```
Ітерація 1: x_n = 1.0, f(x_n) = 1.6, f'(x_n) = -0.2799999999999999
Ітерація 2: x_n = 6.714285714285716, f(x_n) = -60.96697804486237, f'(x_n) = -66.71298625572689
Ітерація 3: x_n = 5.800415994660795, f(x_n) = -17.772274504866388, f'(x_n) = -30.86644763003272
Ітерація 4: x_n = 5.224636276894564, f(x_n) = -4.369318084479557, f'(x_n) = -16.607866645584487
Ітерація 5: x_n = 4.961549019649442, f(x_n) = -0.6541106390359133, f'(x_n) = -11.796167611494617
Ітерація 6: x_n = 4.906097905306295, f(x_n) = -0.02498051444543048, f'(x_n) = -10.90183005295001
Ітерація 7: x_n = 4.903806499705207, f(x_n) = -4.137447905883107e-05, f'(x_n) = -10.865728517125593
Корінь знайдено: 4.903806499705207 після 7 ітерацій
Приблизний корінь: 4.903806499705207
```

Рис 1.10 – Результат методу Ньютона

Як видно з результатів, метод доволі швидко збігся до числа 4, але досягнення точності в 0.01 зайняло 3 ітерації.

Висновки:

Метод дихотомії показав себе як надійний інструмент для пошуку мінімуму функції на заданому інтервалі. Ми побачили, що він ітераційно поділяє інтервал навпіл і дозволяє досить точно визначити область, де розташований мінімум. Однак цей метод може бути дещо повільним, особливо на великих інтервалах, через необхідність постійного скорочення інтервалу.

Метод золотого перерізу виявився більш оптимізованим варіантом пошуку мінімуму. Завдяки використанню співвідношення золотого перетину, цей метод дозволяє швидше знайти мінімум із меншою кількістю обчислень порівняно з методом дихотомії. Це робить його більш ефективним для знаходження мінімумів на довгих інтервалах.

Метод Ньютона, який ми використовували для пошуку кореня полінома, продемонстрував швидку збіжність. Цей метод є одним з найшвидших для знаходження коренів рівнянь, проте його основним недоліком є необхідність обчислення похідних. Ми побачили, що метод швидко наблизився до кореня, однак для досягнення необхідної точності знадобилося декілька ітерацій.

Загалом, робота дозволила глибше зрозуміти принципи роботи різних чисельних методів.

Використані джерела:

- patrickwalls.github.io/mathematicalpython/root-finding/newton/
- en.wikipedia.org/wiki/Golden-section_search
- encyclopediaofmath.org/index.php?title=Dichotomy_method&oldid=46648
- numpy.org/doc/2.1/
- docs.scipy.org/doc/scipy/