**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №4

з дисципліни «Методи Оптимізації та Дослідження Операцій» на тему:

«Методи оптимізації з використанням похідних»

**Виконав**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли**

Викладач Л. Ю. Дейнега

2024

Методи оптимізації з використанням похідних

Мета роботи

Вивчити одновимірні методи оптимізації з використанням похідних

Постановка задачі

Розробити программну реалізацію одновимірних методів оптимізації з використанням похідних

Функція:

Інтервал:

Результати виконання

Код програми

from rich.console import Console

from scipy.misc import derivative

from rich.traceback import install

from scipy.optimize import minimize\_scalar

install()

console = Console()

def f(x):

    return (x - 2) \*\* 2

def derivative(f=f, x=0, step=0.0001):

    return (f(x + step) - f(x)) / step

def newton(f=f, start=-1, end=3, step=0.1, epsilon=1e-6):

    X\_N = start

    while 1:

        F\_Prime = derivative(f, X\_N)

        F\_Double\_Prime = derivative(lambda x: derivative(f, x), X\_N)

        if abs(F\_Prime) < epsilon:

            break

        if F\_Double\_Prime == 0:

            break

        X\_N1 = X\_N - F\_Prime / F\_Double\_Prime

        if abs(X\_N1 - X\_N) < epsilon:

            break

        X\_N = X\_N1

    return X\_N

def bolzani(f=f, start=-1, end=3, step=0.1, epsilon=1e-6):

    Midpoint = (start + end) / 2

    while (end - start) > epsilon:

        Midpoint = (start + end) / 2

        Left\_Derivative = derivative(f, start)

        Right\_Derivative = derivative(f, end)

        Mid\_Derivative = derivative(f, Midpoint)

        if abs(Mid\_Derivative) < epsilon:

            return Midpoint

        if Left\_Derivative > 0 and Mid\_Derivative < 0:

            end = Midpoint

        elif Mid\_Derivative > 0 and Right\_Derivative < 0:

            start = Midpoint

        else:

            if abs(Left\_Derivative) < abs(Right\_Derivative):

                end = Midpoint

            else:

                start = Midpoint

    return Midpoint

def chord(f=f, start=-1, end=3, step=0.1, epsilon=1e-6):

    x\_n = (start + end) / 2

    while abs(f(x\_n)) > epsilon:

        derivative\_value = derivative(f, x\_n)

        if derivative\_value == 0:

            break

        x\_n = x\_n - f(x\_n) / derivative\_value

    return x\_n

def cube(f=f, start=-1, end=3, step=0.1, epsilon=1e-6):

    import numpy as np

    x\_values = np.array([start, end])

    y\_values = np.array([f(start), f(end)])

    derivative\_at\_start = derivative(f, start)

    A = np.array([[start\*\*2, start, 1], [end\*\*2, end, 1], [2 \* start, 1, 0]])

    b = np.array([f(start), f(end), derivative\_at\_start])

    coeffs = np.linalg.solve(A, b)

    a, b, \_ = coeffs

    vertex\_x = -b / (2 \* a)

    if start <= vertex\_x <= end:

        x\_star = vertex\_x

    else:

        x\_star = start if f(start) < f(end) else end

    return x\_star

def getRes(f=f, start=-1, end=3, step=0.1, epsilon=1e-6):

    methods = {

        "Ньютона-Рафсона": {"function": newton, "result": None},

        "Больцано": {"function": bolzani, "result": None},

        "Січних": {"function": chord, "result": None},

        "Кубічної апроксимації": {"function": cube, "result": None},

    }

    for name, method in methods.items():

        method["result"] = method["function"](f, start, end, step, epsilon)

        console.print(f"[bold]{name}[/]: {method['result']:.2f}")

    return methods

def checkCorrect(results):

    correct = minimize\_scalar(f, bounds=(-1, 3)).x

    for name, method in results.items():

        if f"{method['result']:.2f}" == f"{correct:.2f}":

            console.print(

                f"[green bold]Знайдено правильну відповідь![/green bold] {name}"

            )

        else:

            console.print(f"[red bold]Не збігається![/red bold] {name}")

def main() -> None:

    console.print()

    res = getRes()

    console.print()

    checkCorrect(res)

    console.print()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Результати роботи програми

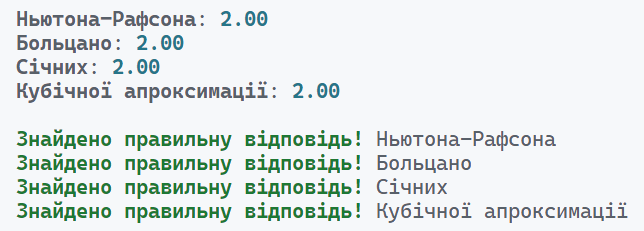


Рисунок 1.1 – Результати розрахунку мінімумів по завершенню роботи програми

Висновки

Таким чином, ми вивчили одновимірні методи оптимізації з використанням похідних

Контрольні питання

Розглянуті раніше методи оптимізації засновані на припущенні про унімодальність й у ряді випадків про безперервність досліджуваної цільової функції. Виконання якої додаткової умови необхідно для реалізації методів оптимізації з використанням похідних?

Умовою, необхідною для реалізації методів оптимізації з використанням похідних, є припущення про диференційованість. Похідна цільової функції повинна існувати і бути неперервною в точці, що нас цікавить, аби ці методи працювали коректно.

Опишіть пошукову процедуру, що реалізує метод Нъютона-Рафсона

Метод Ньютона-Рафсона - це процедура пошуку, яка реалізує використання числових похідних для знаходження коренів функції.

Процедура виглядає наступним чином:

1. Вибираємо початкове значення кореня .

2. Обчислити похідну функції в точці .

3. Якщо дорівнює нулю, завершити цикл.

4. Обчислити наступне наближення для кореня, .

5. Якщо зміна від попередньої ітерації мала, то завершити цикл.

6. Встановити у значення і перейти до кроку 2.

Опишіть схему пошуку з використанням методу середньої точки (пошуку Больцано)

Метод середньої точки, також відомий як пошук Больцано, - це алгоритм пошуку, який передбачає багаторазове розбиття поточного інтервалу на частини, поки не буде знайдено корінь.

Процедура виглядає наступним чином:

1. Вибрати початковий інтервал , який містить корінь.

2. Обчислити медіану інтервалу, .

3. Якщо , то завершити цикл.

4. Якщо знак відрізняється від знаку , то встановити . Інакше встановити .

5. Якщо інтервал скоротився до певного допустимого значення, завершити роботу циклу.

6. Встановити нову медіану як і перейти до кроку 2.