**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №5

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

«Порівняння методів одновимірного пошуку»

**Виконав**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли**

Викладач Л. Ю. Дейнега

2024

Порівняння методів одновимірного пошуку

Мета роботи

Опанувати методику проведення й аналізу чисельних експериментів з метою вибору ефективних стратегій пошуку оптимуму

Постановка задачі

- Провести аналіз ефективності заданих методів пошуку для рішення завдання мінімізації досліджуваної функції.

- Для оцінки ефективності порівнюваних методів оптимізації використати наступні характеристики:

- час, витрачений на одержання рішення;

- кількість обчислень функції (або її похідній), необхідних для досягнення кінцевого результату;

- точність рішення, що вимірюється як відносна (у відсотках) помилка оцінювання координати точки істинного мінімуму;

- чутливість досліджуваних методів до змін параметрів збіжності

Функція:

Інтервал:

Результати виконання

Код програми

from rich.console import Console

from rich.traceback import install

install()

console = Console()

import time

def f(x):

    return (x - 2) \*\* 2

def df(f, x, step=0.0001):

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

def golden(f, a, b, epsilon, step=0.1, calls=0):

    ratio = (5\*\*0.5 - 1) / 2

    c = b - ratio \* (b - a)

    d = a + ratio \* (b - a)

    while abs(c - d) > epsilon:

        if f(c) < f(d):

            b = d

        else:

            a = c

        calls += 2

        c = b - ratio \* (b - a)

        d = a + ratio \* (b - a)

    return (a + b) / 2, calls

def bisection(f, a, b, epsilon, step=0.1, calls=0):

    L = b - a

    while L > epsilon:

        x1 = a + L / 4

        xm = (a + b) / 2

        x2 = b - L / 4

        if f(x1) > f(xm):

            calls += 4

            if f(xm) < f(x2):

                a = x1

                b = x2

            else:

                a = xm

        else:

            b = xm

        L = b - a

    return (a + b) / 2, calls

def powell(f, a, b, epsilon, step=0.1, calls=0):

    def quadraticApproximation(p1, p2, p3, f):

        v1, v2, v3 = f(p1), f(p2), f(p3)

        return (

            0.5

            \* ((v2 - v1) \* (p3\*\*2 - p1\*\*2) - (v3 - v1) \* (p2\*\*2 - p1\*\*2))

            / ((v2 - v1) \* (p3 - p1) - (v3 - v1) \* (p2 - p1))

        )

    while True:

        p2 = a + step

        v1, v2 = f(a), f(p2)

        calls += 2

        p3 = a + 2 \* step if v1 > v2 else a - step

        v3 = f(p3)

        calls += 1

        vMin, pMin = min((v1, a), (v2, p2), (v3, p3))

        pPrime = quadraticApproximation(a, p2, p3, f)

        calls += 4

        if abs(vMin - f(pPrime)) < epsilon and abs(pMin - pPrime) < epsilon:

            return pMin, calls

        a, p2, p3 = sorted(

            [pMin, pPrime, pMin + step if pMin == pPrime else pMin - step]

        )

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

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

def newton(f, a, b, epsilon, step=0.1, calls=0):

    X\_N = a

    while 1:

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

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

        calls += 2

        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, calls

def bolzani(f, a=-1, b=3, epsilon=1e-6, step=0.1, calls=0):

    Midpoint = (a + b) / 2

    while (b - a) > epsilon:

        Midpoint = (a + b) / 2

        Left\_Derivative = derivative(f, a)

        Right\_Derivative = derivative(f, b)

        Mid\_Derivative = derivative(f, Midpoint)

        calls += 3

        if abs(Mid\_Derivative) < epsilon:

            return Midpoint, calls

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

            b = Midpoint

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

            a = Midpoint

        else:

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

                b = Midpoint

            else:

                a = Midpoint

    return Midpoint, calls

def hordination(f, a, b, epsilon, step=0.1, calls=0):

    xn = (a + b) / 2

    while abs(f(xn)) > epsilon:

        calls += 1

        der = df(f, xn)

        calls += 1

        if der == 0:

            break

        xn = xn - f(xn) / der

        calls += 1

    return xn, calls

def cube(f, a, b, epsilon, step=0.1, calls=0):

    import numpy as np

    x\_values = np.array([a, b])

    y\_values = np.array([f(a), f(b)])

    derivative\_at\_a = derivative(f, a)

    calls += 5

    A = np.array([[a\*\*2, a, 1], [b\*\*2, b, 1], [2 \* a, 1, 0]])

    b = np.array([f(a), f(b), derivative\_at\_a])

    calls += 4

    coeffs = np.linalg.solve(A, b)

    calls += 1

    a, b, \_ = coeffs

    vertex\_x = -b / (2 \* a)

    if a <= vertex\_x <= b:

        x\_star = vertex\_x

    else:

        x\_star = a if f(a) < f(b) else b

        calls += 2

    return x\_star, calls

methods = [

    {

        "name": "golden",

        "function": golden,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "bisection",

        "function": bisection,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "powell",

        "function": powell,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "newton",

        "function": newton,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "bolzani",

        "function": bolzani,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "hordination",

        "function": hordination,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

    {

        "name": "cube",

        "function": cube,

        "result": None,

        "time": 0,

        "calls": 0,

        "accuracy": 0,

        "sensitivity": 0,

    },

]

a = -1

b = 3

epsilon = 1e-6

step = 0.1

for method in methods:

    perfectRes = 2.0

    start = time.perf\_counter()

    method["result"], functionCalls = method["function"](f, a, b, epsilon)

    end = time.perf\_counter()

    method["time"] = end - start

    method["calls"] = functionCalls

    method["accuracy"] = abs(method["result"] - perfectRes)

    # Change precision for sensitivity

    Changed\_Precision = 1e-7

    resChanged, calls = method["function"](f, a, b, Changed\_Precision, step)

    method["sensitivity"] = abs(method["result"] - resChanged)

for method in methods:

    console.print(

        f"{method['name']}: {method['result']:.7f}, time: {method['time']:.7f}, calls: {method['calls']}, accuracy: {method['accuracy']:.7f}, sensitivity: {method['sensitivity']:.7f}"

    )

Результати роботи програми

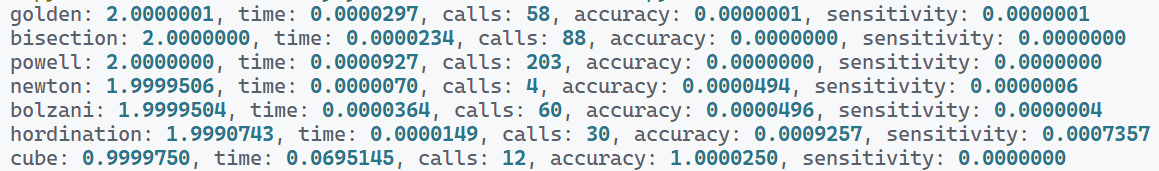


Рисунок 1.1 – Результати розрахунку мінімумів по завершенню роботи програми

Висновки

Таким чином, ми опанували методику проведення й аналізу чисельних експериментів з метою вибору ефективних стратегій пошуку оптимуму

Контрольні питання

Властивості функцій однієї змінної

Функції однієї змінної мають декілька властивостей. Вони можуть бути сталими, зростаючими або спадаючими. Вони також можуть бути неперервними або переривчастими, диференційованими або недиференційованими. Поняття функції включає в себе арифметичні операції над функціями, межі, односторонні межі та монотонні функції.

Критерії оптимальності в одновимірних оптимізаційних задачах

Одновимірна оптимізація передбачає пошук максимуму або мінімуму функції. Методи розв'язання цих задач включають метод виключення, метод інтерполяції та метод прямого пошуку коренів. Мета полягає в тому, щоб знайти точку, яка мінімізує або максимізує функцію в заданому інтервалі.

Ідентифікація оптимумів у випадку функції однієї змінної. Пошук глобального оптимуму

Оптимуми функції можна визначити як точки, в яких похідна функції дорівнює нулю або не існує. Ці точки можуть бути локальними (мінімум або максимум в певній області) або глобальними (мінімум або максимум у всій області визначення функції). Глобальна оптимізація передбачає перебір всього вхідного простору для пошуку глобального оптимуму.