数理工学実験レポート

第 6 章（連続最適化）

学籍番号 1029366161　　中塚一瑳

2025 年 12 月 15 日

目次

はじめに

今回は連続最適化の様々な手法を用いて、与えられた関数の最小値を求める課題に取り組む。

1　課題 15：

2　課題 15

本課題では，多項式

$$f(x) = x^3 + 2x^2 - 5x - 6 \tag{1}$$

の零点を数値的に求める．まず関数のグラフを描画して零点の存在を確認し，その後，二分法およびニュートン法を用いて零点を計算する．

2.1　原理と方法

2.1.1　二分法

二分法は，区間 $[a, b]$ において $f(a)$ と $f(b)$ の符号が異なるとき，その区間内に零点が存在することを利用した反復法である．中点 $c = (a + b)/2$ を取り，$f(c)$ の符号に応じて零点を含む半区間に更新する操作を繰り返すことで，区間幅を徐々に縮小し零点へ収束させる．零点を挟む区間が与えられれば必ず収束するが，収束速度は比較的遅い．

2.1.2　ニュートン法

ニュートン法は，関数をある点 x_k の周りで一次近似し，その接線と x 軸の交点を次の近似値とする方法である．反復公式は

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{2}$$

で与えられる．一般に収束は速いが，初期値の選び方によっては収束しない場合がある．本課題で用いた導関数は

$$f'(x) = 3x^2 + 4x - 5 \tag{3}$$

である．

2.2　実験方法

Python を用いて $x \in [-10, 10]$ の範囲で $f(x)$ を描画し，グラフから零点が $x \approx -3, -1, 2$ 付近に存在することを確認した．二分法では，それぞれの零点を挟む区間として

$$[-4, -2],\ [-2, 0],\ [1, 3]$$

を与えた．ニュートン法では初期値として

$$x_0 = -2.5,\ -0.5,\ 1.5$$

を用いた．停止条件は $|f(x)| \leq 10^{-10}$ とした．

2.3 結果

2.3.1 関数のグラフ

図 1: $f(x) = x^3 + 2x^2 - 5x - 6$ のグラフ

2.3.2 零点

数値計算によって得られた零点を表 1 に示す．

表 1: 二分法およびニュートン法で求めた零点

手法	近似解 x	残差 $f(x)$
二分法	-3.000	0
二分法	-1.000	0
二分法	2.000	0
ニュートン法	-3.000	0
ニュートン法	-1.000	0
ニュートン法	2.000	1.421×10^{-14}

2.4 考察

$f(x) = 0$ は因数分解により

$$f(x) = (x+3)(x+1)(x-2)$$

と書け，解析解は $x = -3, -1, 2$ である．数値計算によって得られた零点はこれらと一致しており，手法が正しく実装されていることが確認できる．ニュートン法において残差が完全に 0 とならない場合があるのは，浮動小数点演算誤差によるものである．

3　課題 16：

結論

付録 A　ソースコード

コード作成、レポート作成の一部に GitHub Copilot を使用した。

A.1　課題 15 のコード

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

# --------------------------
# 課題15: 関数の定義
# f(x) = x^3 + 2x^2 - 5x - 6
# --------------------------

def f(x):
    return x**3 + 2*x**2 - 5*x - 6

def df(x):
    """f(x) の導関数: f'(x) = 3x^2 + 4x - 5"""
    return 3*x**2 + 4*x - 5


# --------------------------
# (a) グラフ描画
# --------------------------

def plot_function():
    x = np.linspace(-10, 10, 2000)
    y = f(x)

    plt.figure()
    plt.plot(x, y)
    plt.axhline(0, color="black", linewidth=0.8)  # x軸
    plt.axvline(0, color="black", linewidth=0.8)  # y軸
    plt.xlim(-10, 10)
    plt.ylim(-10, 10)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.title("Graph of f(x) = x^3 + 2x^2 - 5x - 6")
    plt.grid(True)
    plt.show()
    plt.savefig("task15.png")


# --------------------------
# (b) 二分法
```

```python
# ------------------------

def bisection(f, a, b, eps=1e-10, max_iter=1000):
    """[a, b] で二分法により f(x) = 0 の解を求める.
       f(a) と f(b) の符号は異なることが前提。
    """
    fa = f(a)
    fb = f(b)
    if fa * fb > 0:
        raise ValueError("f(a) と f(b) の符号が同じです: a={}, b={}".format(a, b))

    for _ in range(max_iter):
        c = 0.5 * (a + b)
        fc = f(c)

        if abs(fc) <= eps or 0.5 * (b - a) < eps:
            return c

        # 符号でどちらの区間を残すか決める
        if fa * fc < 0:
            b = c
            fb = fc
        else:
            a = c
            fa = fc

    # 最大反復に達した場合
    return 0.5 * (a + b)


def solve_with_bisection():
    # グラフから零点が -3, -1, 2 付近にあることが分かるので
    # それを挟む区間を手で指定する
    intervals = [
        (-4.0, -2.0),  # -3 付近
        (-2.0,  0.0),  # -1 付近
        ( 1.0,  3.0),  #  2 付近
    ]

    roots = []
    for (a, b) in intervals:
        r = bisection(f, a, b)
        roots.append(r)
    return roots


# ------------------------
# (c) ニュートン法
# ------------------------

def newton(f, df, x0, eps=1e-10, max_iter=1000):
    """ニュートン法: f(x) = 0 の解を初期値 x0 から探索."""
    x = x0
    for _ in range(max_iter):
        fx = f(x)
        dfx = df(x)

        if abs(fx) <= eps:
```

```
            return x

        if dfx == 0:
            # 導関数が 0 になると更新できない
            raise ZeroDivisionError("f'(x) = 0 となったため打ち切り (x={})".format(x))

        x = x - fx / dfx

    return x  # 収束しなかった場合は最後の値を返す


def solve_with_newton():
    # グラフから零点が -3, -1, 2 付近にあることを利用
    initial_points = [-2.5, -0.5, 1.5]
    roots = []
    for x0 in initial_points:
        r = newton(f, df, x0)
        roots.append(r)
    return roots


# -------------------------
# メイン
# -------------------------

def main():
    # (a) グラフ描画
    plot_function()

    # (b) 二分法
    bisection_roots = solve_with_bisection()
    print("Bisection method roots:")
    for r in bisection_roots:
        print("  x ≈ {:.10f}, f(x) ≈ {:.3e}".format(r, f(r)))

    # (c) ニュートン法
    newton_roots = solve_with_newton()
    print("\nNewton method roots:")
    for r in newton_roots:
        print("  x0 -> root ≈ {:.10f}, f(x) ≈ {:.3e}".format(r, f(r)))


if __name__ == "__main__":
    main()
```

参考文献

参考文献

[1] 数理工学実験（2025 年度配布資料）.