

数理工学実験レポート

第 6 章（連続最適化）

学籍番号 1029366161 中塚一瑛

2025 年 12 月 15 日

目次

1	課題 15：ニュートン法および二分法による多項式の零点計算	2
1.1	原理と方法	2
1.2	実験方法	2
1.3	結果	3
1.4	考察	3
2	課題 16：最急降下法およびニュートン法による停留点計算	4
2.1	原理と方法	4
2.2	実装	4
2.3	結果	4
2.4	考察	5
3	課題 17：勾配およびヘッセ行列の解析的導出と実装との対応	5
3.1	微分結果	6
3.2	コードとの対応	6
4	課題 18：最急降下法およびニュートン法による 2 変数関数の最小化	7
4.1	原理と方法	7
4.2	実験方法	7
4.3	結果	7
4.4	考察	8
付録 A	ソースコード	8
A.1	課題 15 のコード	8
A.2	課題 16 のコード	11
A.3	課題 17 のコード	14
A.4	課題 18 のコード	15

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：ニュートン法および二分法による多項式の零点計算

本課題では、多項式

$$f(x) = x^3 + 2x^2 - 5x - 6 \quad (1)$$

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。

1.1 原理と方法

1.1.1 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。

1.1.2 ニュートン法

ニュートン法は、関数がある点 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

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

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

$$f'(x) = 3x^2 + 4x - 5 \quad (3)$$

である。

1.2 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。

1.3 結果

1.3.1 関数のグラフ

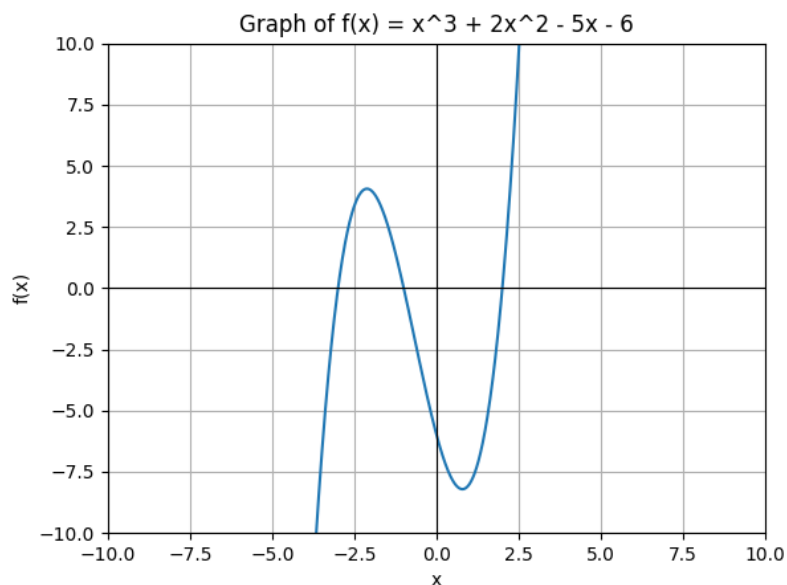


図 1: $f(x) = x^3 + 2x^2 - 5x - 6$ のグラフ

1.3.2 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}

1.4 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.

2 課題 16：最急降下法およびニュートン法による停留点計算

2.1 原理と方法

本課題では、次の関数

$$f(x) := \frac{1}{3}x^3 - x^2 - 3x + \frac{5}{3} \quad (4)$$

の停留点 ($f'(x) = 0$ を満たす点) を、(a) 最急降下法、(b) ニュートン法により数値的に求める。まず微分は

$$f'(x) = x^2 - 2x - 3 = (x - 3)(x + 1), \quad (5)$$

$$f''(x) = 2x - 2 \quad (6)$$

である。従って停留点は解析的には $x = -1, 3$ の 2 点である。

2.1.1 (a) 最急降下法

1 次元では勾配 $\nabla f(x)$ は $f'(x)$ に一致するため、最急降下法は

$$x_{k+1} = x_k - t_k f'(x_k), \quad t_k = \frac{1}{k+1} \quad (7)$$

で与えられる。初期点は $x_0 = 1/2$ とする。停止判定は $|f'(x_k)| \leq \varepsilon$ ($\varepsilon = 10^{-8}$) とし、最大反復回数も設ける。

2.1.2 (b) ニュートン法

停留点探索 ($f'(x) = 0$ の零点探索) としてのニュートン法は

$$x_{k+1} = x_k - t_k \frac{f'(x_k)}{f''(x_k)}, \quad t_k = 1 \quad (8)$$

で与えられる。初期点は $x_0 = 5$ とし、(a) と同様に $|f'(x_k)| \leq \varepsilon$ を停止判定とした。なお、本問題では $f''(x) = 0$ (すなわち $x = 1$) で更新が不能となるため、実装では $f''(x_k) = 0$ の場合を例外として扱う。

2.2 実装

Python により f, f', f'' をそれぞれ関数として実装し、(a) と (b) の更新式をそのまま反復した。各反復で $(x_k, f(x_k), |f'(x_k)|)$ を履歴として保存し、収束挙動の可視化に用いた。

2.3 結果

(a) 最急降下法 ($x_0 = 0.5, t_k = 1/(k+1)$) および (b) ニュートン法 ($x_0 = 5, t_k = 1$) の結果を表 2 に示す。両手法とも停留点 $x^* = 3$ に収束した (もう一つの停留点 $x = -1$ には到達しなかった)。

両方法による反復点の推移を、関数 $y = f(x)$ 上に重ねて図 2 に示す。

表 2: 課題 16 の計算結果 ($\varepsilon = 10^{-8}$)

手法	初期値 x_0	近似解 x^*	$f(x^*)$	反復回数 k
最急降下法	0.5000	3.000	-7.333	193
ニュートン法	5.000	3.000	-7.333	5

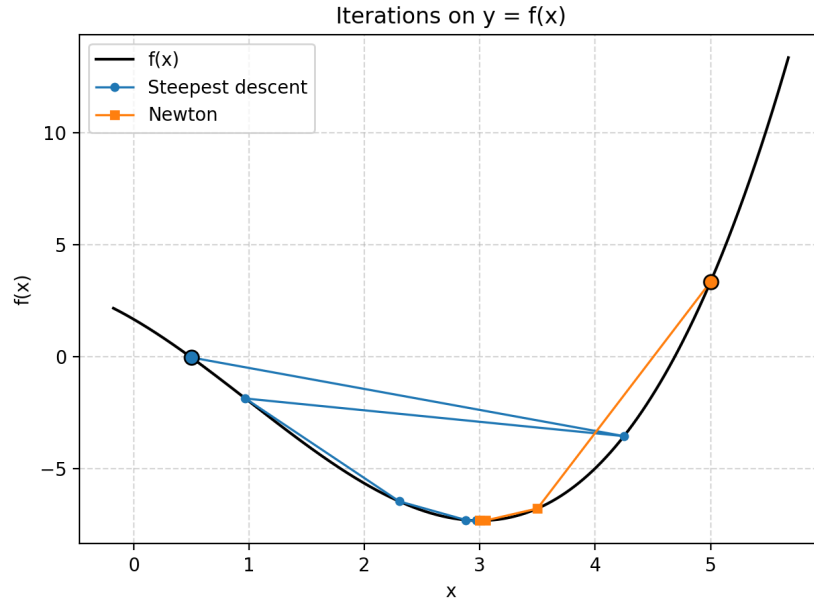


図 2: 関数 $y = f(x)$ 上における反復点の推移. 青丸は最急降下法, 橙四角はニュートン法による反復点を表す.

2.4 考察

図 2 より, 最急降下法では反復点が谷に沿って緩やかに移動しており, ステップサイズ $t_k = 1/(k+1)$ が減少することで後半の収束が遅くなっていることが視覚的に確認できる. 一方, ニュートン法では局所的な 2 次近似に基づき更新が行われるため, 停留点近傍では大きなジャンプを伴い, 極めて高速に収束している.

3 課題 17: 勾配およびヘッセ行列の解析的導出と実装との対応

本課題では, 次の関数

$$f(x_0, x_1) = x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3 \quad (9)$$

について, 勾配およびヘッセ行列を解析的に求め, それを実装したコードとの対応を示す.

3.1 微分結果

まず，各変数による 1 階微分は

$$\frac{\partial f}{\partial x_0} = 2x_0 + e^{x_0} - 2x_1, \quad (10)$$

$$\frac{\partial f}{\partial x_1} = 4x_1^3 + 2x_1 - 2x_0 \quad (11)$$

である．従って勾配ベクトルは

$$\nabla f(x) = \begin{pmatrix} 2x_0 + e^{x_0} - 2x_1 \\ 4x_1^3 + 2x_1 - 2x_0 \end{pmatrix} \quad (12)$$

と表される．

次に，2 階微分よりヘッセ行列は

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} \\ \frac{\partial^2 f}{\partial x_1 \partial x_0} & \frac{\partial^2 f}{\partial x_1^2} \end{pmatrix} = \begin{pmatrix} 2 + e^{x_0} & -2 \\ -2 & 12x_1^2 + 2 \end{pmatrix} \quad (13)$$

となる．

3.2 コードとの対応

上記の解析結果は，以下の Python コードとして実装されている．

- 目的関数

```
1 f = x0**2 + exp(x0) + x1**4 + x1**2 - 2*x0*x1 + 3
```

- 勾配

```
1 g0 = 2*x0 + exp(x0) - 2*x1
2 g1 = 4*x1**3 + 2*x1 - 2*x0
3 g = [g0, g1]
```

- ヘッセ行列

```
1 h00 = 2 + exp(x0)
2 h01 = -2
3 h10 = -2
4 h11 = 12*x1**2 + 2
5 H = [[h00, h01],
6      [h10, h11]]
```

これにより，関数値・勾配・ヘッセ行列が理論式と一致して計算されていることが確認できる．

4 課題 18：最急降下法およびニュートン法による 2 変数関数の最小化

4.1 原理と方法

本課題では、2 変数関数

$$f(x_0, x_1) = x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3 \quad (14)$$

を最小化する。勾配およびヘッセ行列は

$$\nabla f(x) = \begin{pmatrix} 2x_0 + e^{x_0} - 2x_1 \\ 4x_1^3 + 2x_1 - 2x_0 \end{pmatrix}, \quad (15)$$

$$\nabla^2 f(x) = \begin{pmatrix} 2 + e^{x_0} & -2 \\ -2 & 12x_1^2 + 2 \end{pmatrix} \quad (16)$$

である。更新は

$$x_{k+1} = x_k + t_k d_k \quad (17)$$

とし、方向 d_k は (a) 最急降下法 $d_k = -\nabla f(x_k)$, (b) ニュートン法 $\nabla^2 f(x_k) d_k = -\nabla f(x_k)$ の解として定める。

ステップ幅 t_k はバクトラック法 (Armijo 条件) で決定した。すなわち、 $t \leftarrow t_{\text{init}}$ から開始し、

$$f(x_k + td_k) \leq f(x_k) + \xi t \langle \nabla f(x_k), d_k \rangle \quad (18)$$

を満たすまで $t \leftarrow \rho t$ により縮小する。

4.2 実験方法

初期点は $x_0 = (1, 1)^T$ とし、共通パラメータを

$$\xi = 1.0 \times 10^{-4}, \quad \rho = 0.5, \quad t_{\text{init}} = 1 \quad (19)$$

とした。停止条件は $\|\nabla f(x_k)\| \leq 1.0 \times 10^{-6}$ とし、最大反復回数は 1000 とした。各反復で $(x_k, f(x_k))$ を保存し、 x_0 - x_1 平面上の等高線図に反復点列を重ねて可視化した (図 3)。

4.3 結果

(a) バクトラック法付き最急降下法および (b) バクトラック法付きニュートン法の収束結果を表 3 に示す。両手法は同一の解に収束し、ニュートン法の方が少ない反復回数で収束した。

table 3 より、ニュートン法は最急降下法に比べて訳 5 分の 1 の反復回数で収束している。

反復点列を等高線図上に重ねた結果を図 3 に示す。

表 3: 課題 18 の計算結果 (初期点 $x_0 = (1, 1)^T$)

手法	x_0^*	x_1^*	$f(x^*)$	反復回数 k
最急降下法 +BT	-0.7335	-0.4933	3.597	31
ニュートン法 +BT	-0.7335	-0.4933	3.597	6

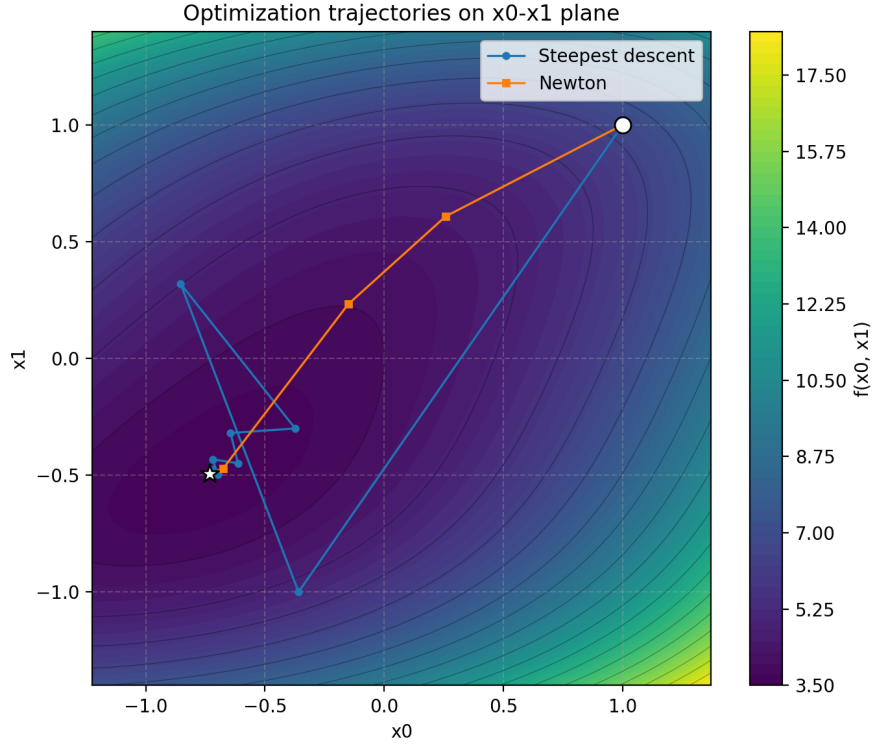


図 3: x_0-x_1 平面における等高線図と反復点列. \circ が最急降下法 +BT, \square がニュートン法 +BT の反復点列を表す.

4.4 考察

図 3 より, 最急降下法では反復点同士を結ぶ線分が等高線に対して直交しているのに対し, ニュートン法では直交していないものの, あたかも低い場所を知っているかのように, 等高線の谷に沿って効率的に移動している様子がわかる. これは, ニュートン法がヘッセ行列を用いて関数の曲率を考慮しているためであり, その結果として少ない反復回数で収束していると考えられる.

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


```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  # -----
8  # 課題15: 関数の定義
9  #  $f(x) = x^3 + 2x^2 - 5x - 6$ 
10 # -----
11
12 def f(x):
13     return x**3 + 2*x**2 - 5*x - 6
14
15 def df(x):
16     """ $f(x)$  の導関数:  $f'(x) = 3x^2 + 4x - 5$ """
17     return 3*x**2 + 4*x - 5
18
19 # -----
20 # (a) グラフ描画
21 # -----
22
23
24 def plot_function():
25     x = np.linspace(-10, 10, 2000)
26     y = f(x)
27
28     plt.figure()
29     plt.plot(x, y)
30     plt.axhline(0, color="black", linewidth=0.8) # x軸
31     plt.axvline(0, color="black", linewidth=0.8) # y軸
32     plt.xlim(-10, 10)
33     plt.ylim(-10, 10)
34     plt.xlabel("x")
35     plt.ylabel("f(x)")
36     plt.title("Graph of  $f(x) = x^3 + 2x^2 - 5x - 6$ ")
37     plt.grid(True)
38     plt.savefig("task15.png")
39     plt.show()
40
41 # -----
42 # (b) 二分法
43 # -----
44
45
46 def bisection(f, a, b, eps=1e-10, max_iter=1000):
47     """[a, b] で二分法により  $f(x) = 0$  の解を求める。
48      $f(a)$  と  $f(b)$  の符号は異なることが前提。
49     """
50     fa = f(a)
51     fb = f(b)
52     if fa * fb > 0:
53         raise ValueError("f(a) と f(b) の符号が同じです: a={}, b={}".format(a, b))
54
55     for _ in range(max_iter):
56         c = 0.5 * (a + b)
57         fc = f(c)

```

```

58
59     if abs(fc) <= eps or 0.5 * (b - a) < eps:
60         return c
61
62     # 符号でどちらの区間を残すか決める
63     if fa * fc < 0:
64         b = c
65         fb = fc
66     else:
67         a = c
68         fa = fc
69
70     # 最大反復に達した場合
71     return 0.5 * (a + b)
72
73
74 def solve_with_bisection():
75     # グラフから零点が -3, -1, 2 付近にあることが分かるので
76     # それを挟む区間を手で指定する
77     intervals = [
78         (-4.0, -2.0), # -3 付近
79         (-2.0, 0.0), # -1 付近
80         (1.0, 3.0), # 2 付近
81     ]
82
83     roots = []
84     for (a, b) in intervals:
85         r = bisection(f, a, b)
86         roots.append(r)
87     return roots
88
89
90 # -----
91 # (c) ニュートン法
92 # -----
93
94 def newton(f, df, x0, eps=1e-10, max_iter=1000):
95     """ニュートン法: f(x) = 0 の解を初期値 x0 から探索."""
96     x = x0
97     for _ in range(max_iter):
98         fx = f(x)
99         dfx = df(x)
100
101         if abs(fx) <= eps:
102             return x
103
104         if dfx == 0:
105             # 導関数が 0 になると更新できない
106             raise ZeroDivisionError("f'(x) = 0 となったため打ち切り (x={})".format(x))
107
108         x = x - fx / dfx
109
110     return x # 収束しなかった場合は最後の値を返す
111
112
113 def solve_with_newton():
114     # グラフから零点が -3, -1, 2 付近にあることを利用
115     initial_points = [-2.5, -0.5, 1.5]

```

```

116     roots = []
117     for x0 in initial_points:
118         r = newton(f, df, x0)
119         roots.append(r)
120     return roots
121
122
123 # -----
124 # メイン
125 # -----
126
127 def main():
128     # (a) グラフ描画
129     plot_function()
130
131     # (b) 二分法
132     bisection_roots = solve_with_bisection()
133     print("Bisection method roots:")
134     for r in bisection_roots:
135         print(" x ≈ {:.10f}, f(x) ≈ {:.3e}".format(r, f(r)))
136
137     # (c) ニュートン法
138     newton_roots = solve_with_newton()
139     print("\nNewton method roots:")
140     for r in newton_roots:
141         print(" x0 -> root ≈ {:.10f}, f(x) ≈ {:.3e}".format(r, f(r)))
142
143
144 if __name__ == "__main__":
145     main()

```

A.2 課題 16 のコード

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # 課題16
5  # f(x) = (1/3)x^3 - x^2 - 3x + 5/3 の停留点を求める
6  # (a) 最急降下法 (x0 = 1/2, tk = 1/(k+1))
7  # (b) ニュートン法 (x0 = 5, tk = 1)
8
9  from __future__ import annotations
10
11 from dataclasses import dataclass
12 from pathlib import Path
13
14
15 def f(x):
16     """目的関数 f(x) = 1/3 x^3 - x^2 - 3x + 5/3"""
17     return (1.0 / 3.0) * x**3 - x**2 - 3.0 * x + 5.0 / 3.0
18
19
20 def df(x):
21     """1階微分 f'(x) = x^2 - 2x - 3"""
22     return x**2 - 2.0 * x - 3.0
23

```

```

24
25 def d2f(x):
26     """2階微分 f''(x) = 2x - 2"""
27     return 2.0 * x - 2.0
28
29
30 @dataclass(frozen=True)
31 class IterationHistory:
32     x: list[float]
33     f: list[float]
34     grad_abs: list[float]
35
36
37 # -----
38 # (a) 最急降下法
39 # -----
40
41
42 def steepest_descent(x0, eps=1e-8, max_iter=1000, *, return_history=False):
43     """
44     最急降下法 (1次元版)
45      $x_{k+1} = x_k - t_k * f'(x_k)$ 
46      $t_k = 1 / (k+1)$ 
47     """
48     x = x0
49     history_x = [float(x)]
50     history_f = [float(f(x))]
51     history_grad_abs = [float(abs(df(x)))]
52
53     for k in range(max_iter):
54         g = df(x) # 勾配 (1次元なので単なる導関数)
55         if abs(g) <= eps:
56             # 停留点に到達したとみなす
57             if return_history:
58                 return x, k, IterationHistory(history_x, history_f, history_grad_abs)
59             return x, k
60
61         t_k = 1.0 / (k + 1) # 指定どおりのステップ幅
62         x = x - t_k * g
63
64         history_x.append(float(x))
65         history_f.append(float(f(x)))
66         history_grad_abs.append(float(abs(df(x))))
67
68     # 最大反復に到達した場合
69     if return_history:
70         return x, max_iter, IterationHistory(history_x, history_f, history_grad_abs)
71     return x, max_iter
72
73
74 # -----
75 # (b) ニュートン法
76 # -----
77
78
79 def newton_method(x0, eps=1e-8, max_iter=1000, *, return_history=False):
80     """
81     ニュートン法

```

```

82     x_{k+1} = x_k - t_k * f'(x_k) / f''(x_k)
83     ここでは t_k = 1
84     """
85     x = x0
86     history_x = [float(x)]
87     history_f = [float(f(x))]
88     history_grad_abs = [float(abs(df(x)))]
89
90     for k in range(max_iter):
91         g = df(x)
92         h = d2f(x)
93
94         if abs(g) <= eps:
95             # 停留点に到達したとみなす
96             if return_history:
97                 return x, k, IterationHistory(history_x, history_f, history_grad_abs)
98             return x, k
99
100         if h == 0.0:
101             raise ZeroDivisionError(f"f''(x) = 0 となったため更新できません (x = {x})")
102
103         t_k = 1.0 # 指定どおり常に1
104         x = x - t_k * g / h
105
106         history_x.append(float(x))
107         history_f.append(float(f(x)))
108         history_grad_abs.append(float(abs(df(x))))
109
110     if return_history:
111         return x, max_iter, IterationHistory(history_x, history_f, history_grad_abs)
112     return x, max_iter
113
114 # -----
115 # メイン
116 # -----
117
118
119
120 def main():
121     print("=== (a) 最急降下法 ===")
122     x0_sd = 0.5 # 初期点 x0 = 1/2
123     x_star_sd, it_sd, hist_sd = steepest_descent(x0_sd, return_history=True)
124     print("初期値 x0 = {:.6f}".format(x0_sd))
125     print("近似停留点 x* ? {:.10f}".format(x_star_sd))
126     print("f'(x*) ? {:.3e}".format(df(x_star_sd)))
127     print("反復回数 k =", it_sd)
128     print()
129
130     print("=== (b) ニュートン法 ===")
131     x0_nt = 5.0 # 初期点 x0 = 5
132     x_star_nt, it_nt, hist_nt = newton_method(x0_nt, return_history=True)
133     print("初期値 x0 = {:.6f}".format(x0_nt))
134     print("近似停留点 x* ? {:.10f}".format(x_star_nt))
135     print("f'(x*) ? {:.3e}".format(df(x_star_nt)))
136     print("反復回数 k =", it_nt)
137     print()
138

```

```

139 # y=f(x) 上で、反復点が収束していく様子を同一グラフにプロット
140 try:
141     import matplotlib.pyplot as plt
142 except ImportError:
143     print("matplotlib が見つからないため、プロットをスキップします。")
144     return
145
146 all_x = hist_sd.x + hist_nt.x
147 x_min = min(all_x)
148 x_max = max(all_x)
149 margin = 0.15 * (x_max - x_min) if x_max > x_min else 1.0
150 x_left = x_min - margin
151 x_right = x_max + margin
152
153 n = 600
154 xs = [x_left + (x_right - x_left) * i / n for i in range(n + 1)]
155 ys = [f(x) for x in xs]
156
157 plt.figure()
158 plt.plot(xs, ys, color="black", linewidth=1.5, label="f(x)")
159
160 plt.plot(hist_sd.x, hist_sd.f, marker="o", markersize=4, linewidth=1.2, label="
    Steepest descent")
161 plt.plot(hist_nt.x, hist_nt.f, marker="s", markersize=4, linewidth=1.2, label="
    Newton")
162
163 plt.scatter([hist_sd.x[0]], [hist_sd.f[0]], s=60, edgecolors="black", zorder=3)
164 plt.scatter([hist_nt.x[0]], [hist_nt.f[0]], s=60, edgecolors="black", zorder=3)
165
166 plt.xlabel("x")
167 plt.ylabel("f(x)")
168 plt.title("Iterations on y = f(x)")
169 plt.grid(True, linestyle="--", alpha=0.5)
170 plt.legend()
171
172 out_path = Path(__file__).with_name("task16.png")
173 plt.tight_layout()
174 plt.savefig(out_path, dpi=200)
175 print(f"プロットを保存しました: {out_path}")
176 import matplotlib
177
178 if "agg" not in matplotlib.get_backend().lower():
179     plt.show()
180
181
182 if __name__ == "__main__":
183     main()

```

A.3 課題 17 のコード

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import numpy as np
5
6 # f(x) = x0^2 + exp(x0) + x1^4 + x1^2 - 2 x0 x1 + 3

```

```

7
8 def evalf(x):
9     """
10     目的関数の値 f(x) を計算する
11     x: np.array shape (2,)
12     """
13     x0, x1 = x[0], x[1]
14     f = x0**2 + np.exp(x0) + x1**4 + x1**2 - 2.0*x0*x1 + 3.0
15     return f
16
17 def evalg(x):
18     """
19     勾配ベクトル  $\nabla f(x)$  を計算する
20      $\nabla f(x) = [ 2x_0 + \exp(x_0) - 2x_1,$ 
21                  $4x_1^3 + 2x_1 - 2x_0 ]$ 
22     """
23     x0, x1 = x[0], x[1]
24     g0 = 2.0*x0 + np.exp(x0) - 2.0*x1
25     g1 = 4.0*x1**3 + 2.0*x1 - 2.0*x0
26     g = np.array([g0, g1])
27     return g
28
29 def evalh(x):
30     """
31     ヘッセ行列  $\nabla^2 f(x)$  を計算する
32      $\nabla^2 f(x) =$ 
33          $\begin{bmatrix} 2 + \exp(x_0) & -2 \\ -2 & 12x_1^2 + 2 \end{bmatrix}$ 
34     """
35     x0, x1 = x[0], x[1]
36     h00 = 2.0 + np.exp(x0)
37     h01 = -2.0
38     h10 = -2.0
39     h11 = 12.0*x1**2 + 2.0
40     H = np.array([[h00, h01],
41                   [h10, h11]])
42     return H
43
44
45 def main():
46     # 動作確認用
47     x = np.array([0.3, 5.0])
48     f = evalf(x)
49     g = evalg(x)
50     H = evalh(x)
51
52     print("x =", x)
53     print("f(x) =", f)
54     print("g(x) =", g)
55     print("H(x) =\n", H)
56
57 if __name__ == "__main__":
58     main()

```

A.4 課題 18 のコード

```

1 #!/usr/bin/env python

```

```

2  # -*- coding: utf-8 -*-
3
4  from __future__ import annotations
5
6  from dataclasses import dataclass
7  from pathlib import Path
8
9  import numpy as np
10
11  # =====
12  # 課題17の設定
13  #  $f(x) = x_0^2 + \exp(x_0) + x_1^4 + x_1^2 - 2x_0x_1 + 3$ 
14  # =====
15
16
17  def evalf(x):
18      """
19      目的関数  $f(x)$  を計算する
20      x: np.array shape (2,)
21      """
22      x0, x1 = x[0], x[1]
23      f = x0**2 + np.exp(x0) + x1**4 + x1**2 - 2.0 * x0 * x1 + 3.0
24      return f
25
26
27  def evalg(x):
28      """
29      勾配ベクトル  $\nabla f(x)$  を計算する
30       $\nabla f(x) = [ 2x_0 + \exp(x_0) - 2x_1,$ 
31                   $4x_1^3 + 2x_1 - 2x_0 ]$ 
32      """
33      x0, x1 = x[0], x[1]
34      g0 = 2.0 * x0 + np.exp(x0) - 2.0 * x1
35      g1 = 4.0 * x1**3 + 2.0 * x1 - 2.0 * x0
36      return np.array([g0, g1])
37
38
39  def evalh(x):
40      """
41      ヘッセ行列  $\nabla^2 f(x)$  を計算する
42       $\nabla^2 f(x) =$ 
43           $\begin{bmatrix} 2 + \exp(x_0) & -2 \\ -2 & 12x_1^2 + 2 \end{bmatrix}$ 
44      """
45      x0, x1 = x[0], x[1]
46      h00 = 2.0 + np.exp(x0)
47      h01 = -2.0
48      h10 = -2.0
49      h11 = 12.0 * x1**2 + 2.0
50      return np.array([[h00, h01], [h10, h11]])
51
52
53
54  @dataclass(frozen=True)
55  class IterationHistory2D:
56      x: np.ndarray # shape (n, 2)
57      f: np.ndarray # shape (n,)
58
59

```



```

60 # =====
61 # バックトラック法（共通で使う）
62 # =====
63
64
65 def backtracking(xk, dk, evalf, evalg, t_init=1.0, rho=0.5, xi=1e-4):
66     """
67     バックトラック法（アルミホ条件）
68     与えられた点 xk と方向 dk に対してステップ幅 t を返す。
69     """
70     t = t_init
71     fk = evalf(xk)
72     gk = evalg(xk)
73     # アルミホ条件:  $f(x_k + t dk) \leq f(x_k) + \xi t \langle dk, gk \rangle$ 
74     while True:
75         x_new = xk + t * dk
76         if evalf(x_new) <= fk + xi * t * np.dot(dk, gk):
77             break
78         t *= rho
79     return t
80
81
82 # =====
83 # (a) バックトラック法付き最急降下法
84 # =====
85
86
87 def steepest_descent_bt(
88     x0,
89     evalf,
90     evalg,
91     eps=1e-6,
92     max_iter=1000,
93     xi=1e-4,
94     rho=0.5,
95     t_init=1.0,
96     *,
97     return_history=False,
98 ):
99     """
100     バックトラック法付き最急降下法
101      $x_{k+1} = x_k + t_k d_k$ ,  $d_k = -\nabla f(x_k)$ 
102     """
103     x = x0.copy()
104     history_x = [x.copy()]
105     history_f = [float(evalf(x))]
106
107     for k in range(max_iter):
108         g = evalg(x)
109         norm_g = np.linalg.norm(g)
110         if norm_g <= eps:
111             if return_history:
112                 hist = IterationHistory2D(np.stack(history_x, axis=0), np.array(
113                     history_f))
114                 return x, evalf(x), k, hist
115             return x, evalf(x), k
116
117     dk = -g

```

```

117         tk = backtracking(x, dk, evalf, evalg, t_init=t_init, rho=rho, xi=xi)
118         x = x + tk * dk
119
120         history_x.append(x.copy())
121         history_f.append(float(evalf(x)))
122
123     if return_history:
124         hist = IterationHistory2D(np.stack(history_x, axis=0), np.array(history_f))
125         return x, evalf(x), max_iter, hist
126     return x, evalf(x), max_iter
127
128
129 # =====
130 # (b) バックトラック法付きニュートン法
131 # =====
132
133
134 def newton_bt(
135     x0,
136     evalf,
137     evalg,
138     evalh,
139     eps=1e-6,
140     max_iter=1000,
141     xi=1e-4,
142     rho=0.5,
143     t_init=1.0,
144     *,
145     return_history=False,
146 ):
147     """
148     バックトラック法付きニュートン法
149     d_k はヘッセ行列を用いて  $\nabla^2 f(x_k)$  d_k =  $-\nabla f(x_k)$  を解く
150     """
151     x = x0.copy()
152     history_x = [x.copy()]
153     history_f = [float(evalf(x))]
154
155     for k in range(max_iter):
156         g = evalg(x)
157         norm_g = np.linalg.norm(g)
158         if norm_g <= eps:
159             if return_history:
160                 hist = IterationHistory2D(np.stack(history_x, axis=0), np.array(
161                     history_f))
162                 return x, evalf(x), k, hist
163             return x, evalf(x), k
164
165         H = evalh(x)
166         # ニュートン方向を解く: H d = -g
167         try:
168             dk = np.linalg.solve(H, -g)
169         except np.linalg.LinAlgError:
170             break
171
172         tk = backtracking(x, dk, evalf, evalg, t_init=t_init, rho=rho, xi=xi)
173         x = x + tk * dk

```

```

174         history_x.append(x.copy())
175         history_f.append(float(evalf(x)))
176
177     if return_history:
178         hist = IterationHistory2D(np.stack(history_x, axis=0), np.array(history_f))
179         return x, evalf(x), max_iter, hist
180     return x, evalf(x), max_iter
181
182
183 # =====
184 # メイン
185 # =====
186
187
188 def main():
189     # 共通パラメータ (課題文指定)
190     xi = 1e-4
191     rho = 0.5
192     t_init = 1.0
193     x0 = np.array([1.0, 1.0]) # 初期点 (1, 1)^T
194
195     print("初期点 x0 =", x0)
196
197     # (a) 最急降下法
198     print("\n=== (a) バックトラック法付き最急降下法 ===")
199     x_sd, f_sd, k_sd, hist_sd = steepest_descent_bt(
200         x0,
201         evalf,
202         evalg,
203         eps=1e-6,
204         max_iter=1000,
205         xi=xi,
206         rho=rho,
207         t_init=t_init,
208         return_history=True,
209     )
210     print("最適解近似 x* ?", x_sd)
211     print("最適値近似 f(x*) ?", f_sd)
212     print("反復回数 =", k_sd)
213
214     # (b) ニュートン法
215     print("\n=== (b) バックトラック法付きニュートン法 ===")
216     x_nt, f_nt, k_nt, hist_nt = newton_bt(
217         x0,
218         evalf,
219         evalg,
220         evalh,
221         eps=1e-6,
222         max_iter=1000,
223         xi=xi,
224         rho=rho,
225         t_init=t_init,
226         return_history=True,
227     )
228     print("最適解近似 x* ?", x_nt)
229     print("最適値近似 f(x*) ?", f_nt)
230     print("反復回数 =", k_nt)
231

```

```

232 # x0-x1 平面に f の値を色で表示し、両手法の反復点の遷移を同一グラフに重ねる
233 try:
234     import matplotlib.pyplot as plt
235 except ImportError:
236     print("matplotlib が見つからないため、プロットをスキップします。")
237     return
238
239 all_points = np.vstack([hist_sd.x, hist_nt.x])
240 x0_min, x1_min = np.min(all_points, axis=0)
241 x0_max, x1_max = np.max(all_points, axis=0)
242
243 margin0 = 0.2 * (x0_max - x0_min) if x0_max > x0_min else 1.0
244 margin1 = 0.2 * (x1_max - x1_min) if x1_max > x1_min else 1.0
245 x0_left, x0_right = x0_min - margin0, x0_max + margin0
246 x1_bottom, x1_top = x1_min - margin1, x1_max + margin1
247
248 n = 250
249 grid_x0 = np.linspace(x0_left, x0_right, n)
250 grid_x1 = np.linspace(x1_bottom, x1_top, n)
251 X0, X1 = np.meshgrid(grid_x0, grid_x1)
252 F = X0**2 + np.exp(X0) + X1**4 + X1**2 - 2.0 * X0 * X1 + 3.0
253
254 plt.figure(figsize=(7, 6))
255 cf = plt.contourf(X0, X1, F, levels=60, cmap="viridis")
256 plt.contour(X0, X1, F, levels=20, colors="k", linewidths=0.4, alpha=0.35)
257 cbar = plt.colorbar(cf)
258 cbar.set_label("f(x0, x1)")
259
260 plt.plot(hist_sd.x[:, 0], hist_sd.x[:, 1], "-o", markersize=3.5, linewidth=1.2,
261         label="Steepest descent")
262 plt.plot(hist_nt.x[:, 0], hist_nt.x[:, 1], "-s", markersize=3.5, linewidth=1.2,
263         label="Newton")
264
265 plt.scatter([hist_sd.x[0, 0]], [hist_sd.x[0, 1]], s=80, c="white", edgecolors="
266             black", zorder=4)
267 plt.scatter([hist_nt.x[0, 0]], [hist_nt.x[0, 1]], s=80, c="white", edgecolors="
268             black", zorder=4)
269 plt.scatter([hist_sd.x[-1, 0]], [hist_sd.x[-1, 1]], s=110, marker="*", c="white",
270             edgecolors="black", zorder=5)
271 plt.scatter([hist_nt.x[-1, 0]], [hist_nt.x[-1, 1]], s=110, marker="*", c="white",
272             edgecolors="black", zorder=5)
273
274 plt.xlabel("x0")
275 plt.ylabel("x1")
276 plt.title("Optimization trajectories on x0-x1 plane")
277 plt.grid(True, linestyle="--", alpha=0.3)
278 plt.legend()
279
280 out_path = Path(__file__).with_name("task18.png")
281 plt.tight_layout()
282 plt.savefig(out_path, dpi=200)
283 print(f"プロットを保存しました: {out_path}")
284
285 import matplotlib
286
287 if "agg" not in matplotlib.get_backend().lower():
288     plt.show()
289

```

```
284 |  
285 | if __name__ == "__main__":  
286 |     main()
```

参考文献

参考文献

- [1] 数理工学実験（2025 年度配布資料）.