数理工学実験
テーマ:連続最適化

2回生　田中風帆 (1029321151)
実施場所:自宅

実施:2021年11月29日
提出:2021年12月12日

目 次

図 目 次

表 目 次

第 I 部
概要

このレポートでは、与えられた関数 f:$\mathbb{R}^n \to \mathbb{R}$ の零点および最小値または最大値を与える点 $x \in \mathbb{R}^n$ を求める手法を学び、Python を用いた演習問題への回答を通してその手法を理解する。課題 1 における Python の導入に始まり、関数の零点を求める二分法とニュートン法についての課題を解く。また、降下法を導入し、最急降下法とニュートン法についての課題を解く。さらに直線探索を用いたニュートン法および最適化手法の収束性について学び、最後にはニュートン法に修正を加えた手法を導入したのち総合的な演習課題を解く。

第 II 部
課題 1

ここでは Python のインストールを行う。私のパソコンには Python3.9.7、Numpy、Matplotlib がインストールされているためこの課題は省略する。

第 III 部
課題 2

ここでは、次の 2 次関数 f:$\mathbb{R} \to \mathbb{R}$ の描画および二種類の手法による零点の計算を行った。

$$f(x) = x^3 + 2x^2 - 5x - 6 \tag{0.1}$$

(a) では関数 f を $-10 \leq x \leq 10$、$-10 \leq y \leq 10$ の範囲で描画した。(b) では関数 f の零点を二分法を用いて求めた。ただし、二分法を用いる際に必要となる二つの異なる初期点は、(a) で描画した関数 f の概形を参考にして決定した。(c) では関数 f の零点をニュートン法を用いて求めた。ただし初期点は (a) で描画した関数 f の概形を参考にすることで、零点の近くに設定した。

1　準備

課題 2 を解くのに必要な前提知識および手法と、その理論をまとめる。

1.1　関数の零点探索

与えられた関数 f:$\mathbb{R}^n \to \mathbb{R}$ に対して $f(x) = 0$ を満たす点 x のことを関数 f の零点という。本問の (b),(c) で求めるのはこれである。関数の零点を求める手法のうち、二分法とニュートン法について述べる。

1.1.1　二分法

以下で与えられるアルゴリズムが二分法である。

二分法

手順 1:$f(a) < 0, 0 \leq f(b)$ を満たす初期点 a, b を選び、終了条件 $\epsilon > 0$ を決定する。
手順 2：a と b の中間点 $c = \frac{a+b}{2}$ を求める。仮に c が終了条件 $|f(c)| \leq \epsilon$ を満たしていたら c を解とし、終了する。
手順 3：もし $f(c) < 0$ であれば a に c を代入し、$0 \leq f(c)$ であれば b に c を代入して手順 2 に戻る。

二分法は関数 f が $\mathbb{R}$ 上で連続であれば必ず収束する。手順 1 の初期条件を満たす a, b を選べば関数 f の零点は a と b の間に存在することから、a と b の間隔を反復を経るたびに半分にしていけば、a と b の中間点である c が f の零点に近づくことは明らかである。特に f が単調増加関数または単調減少関数の時はただ一つの解に必ず収束する。これは $f(a) < 0, 0 \leq f(b)$ であることから、中間値の定理を適用することで証明できる。ネット上のページのいくつかに証明の詳細が記載されている [2]。逆に、f が単調でない場合は零点が複数存在する可能性があるため、全ての零点を求めるには複数種類の初期点 a, b を用いてアルゴリズムを実行する必要がある (本問の (b) でもこれを行う)。
二分法はアルゴリズムが簡単であり、初期点 a, b の間に零点が存在すれば必ず収束するという点で優れているが、収束が遅いという欠点がある。次節で述べるニュートン法はより高速なアルゴリズムである。

1.1.2 ニュートン法

以下で与えられるアルゴリズムがニュートン法である。

ニュートン法

手順 1:初期点 x_0 を選び、終了条件 ϵ を決める。$k = 0$ とする。
手順 2：ニュートン方程式

$$f'(x_k)\Delta x_k = -f(x_k)$$

を解き、Δx_k を求める。
手順 3：点列を $x_{k+1} = x_k + \Delta x_k$ により更新し、もし $|f(x_{k+1})| \leq \epsilon$ を満たしていれば x_{k+1} を解として終了する。
手順 4：$k \leftarrow k + 1$ として手順 2 に戻る。

ニュートン法のアルゴリズムは、点 x_k の周辺で関数 f を一次近似し、接線の方程式 $y = f'(x_k)(x - x_k) + f(x_k)$ と x 軸の交点を求め、更新点とするこ

とを表す。すなわち、$f'(x_k)(x-x_k)+f(x_k)=0$ を解き、$x=x_k-\frac{f(x_k)}{f'(x_k)}$ としている。一般的に収束は速い。実際、ニュートン法は 2 次収束である。証明はネット上にいくつか掲載されている [3]。しかし、解の近くに初期点を選ばなければ収束しないという欠点がある。また、その解の導出法より、少なくとも f が微分可能でなければならない。

2　課題 2 の回答

以下の問題を解くためのコードは全て Python で実装した。

2.1　(a)

Python の Matplotlib を用い、式 (0.1) で定義される関数 f を $-10 \leq x \leq 10$、$-10 \leq y \leq 10$ の範囲で描画した。ただし x を幅 0.1 の区間に離散化して描画した。結果は以下のようになった。

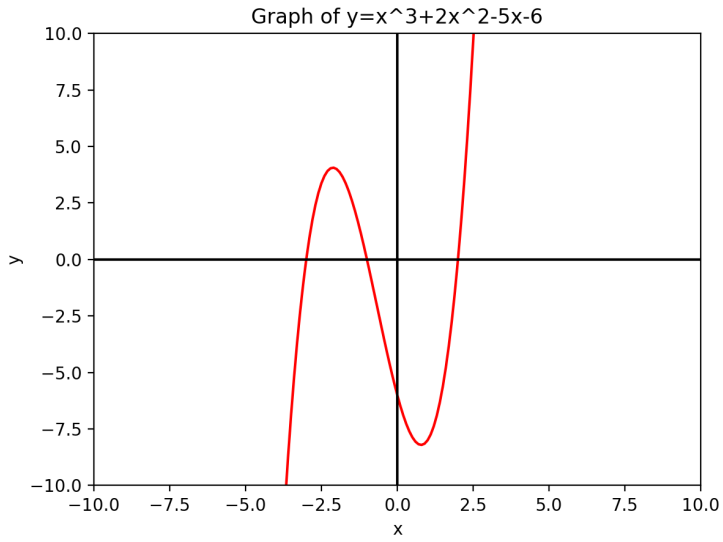

図 1: 式 (0.1) で定義される関数 f のグラフ. 横軸が x で縦軸が y である. 零点が 3 つ存在することがわかる.

2.2　(b)

関数 f の零点を二分法を用いて求めた。グラフより、零点が $x=-3,-1,2$ のあたりに存在することがわかる。よって、初期点の組 (a,b) として、この 3

点に近くでありかつ $f(a) < 0, 0 \leq f(b)$ を満たす点 (-4.0, -2.5)、(-2.5,-0.5)、(1.5,3.0) を採用した。また、手順 1 において $\epsilon = 10^{-6}$ とした。結果は以下のようになった。なお、(c) で行うニュートン法との比較のため、収束までの反復回数も結果の表に含める。

表 1: 二分法を用いて求めた零点. 左から初期点の組、零点、収束までの反復回数となっている.

初期点 (a,b)	零点	反復回数
$(-4.0, -2.5)$	-3.0000000596046448	23
$(-2.5, -0.5)$	-1.0	1
$(1.5, 3.0)$	2.0000000596046448	23

2.3 (c)

関数 f の零点をニュートン法を用いて求めた。グラフより、零点が $x = -3, -1, 2$ のあたりに存在することがわかる。よって、零点に近い初期点として-3.5,0.5,1.5 を採用した。手順 2 においては、$-f(x_k)$ を $f'(x_k)$ で割り Δx_k を求めた。また、手順 1 において $\epsilon = 10^{-6}$ とした。結果は以下のようになった。

表 2: ニュートン法を用いて求めた零点. 左から初期点、零点、収束までの反復回数となっている.

初期点	零点	反復回数
-3.5	-3.0000000013109727	4
0.5	-1.0	1
1.5	2.0000000406383567	4

3 この課題の考察とまとめ

式 (0.1) を因数分解すると $f(x) = (x-2)(x+3)(x+1)$ となることから、正確な解は $x = -3, -1, 2$ である。これと (b)、(c) より、どちらの手法でもほぼ正確に零点を求められていることがわかった。また、反復回数の比較より、確かに二分法よりニュートン法の方が収束が速いことが確認できた。これは準備の章で述べた理論が正しいことを数値的に示している。

4 コード

4.1 (a) のコード

コード 1: 課題 2(a) のコード

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def f(x):
5      return x**3 + 2.0*(x**2) - 5.0*x - 6.0 #f(x) を定義
6
7  #x の定義域
8  x = np.arange(-10, 10, 0.1)
9  y = f(x)
10 plt.plot(x, y, 'red')
11 #プロットの範囲
12 plt.xlim(-10, 10)
13 plt.ylim(-10, 10)
14 plt.axhline(0, c='black')
15 plt.axvline(0, c='black')
16 #ラベルに名前をつける
17 plt.xlabel('x')
18 plt.ylabel('y')
19 #グラフの名前
20 plt.title('Graph␣of␣y=x^3+2x^2-5x-6')
21 plt.show()
```

4.2 (b) のコード

コード 2: 課題 2(b) のコード

```
1  def f(x):
2      return x**3 + 2.0*(x**2) - 5.0*x - 6.0
3
4  epsilon = 0.1**6 #終了条件
5  a_and_b = [[-4.0,-2.5],[ -2.5,0.5],[1.5,3.0]] #初期点
6
7  for i in range(3):
8      a = a_and_b[i][0] #a の値
9      b = a_and_b[i][1] #b の値
10     c = 5.0 #f(c) がイプシロンより大きくなる初期値
11     k = 0
12
13     #反復
14     while abs(f(c)) > epsilon:
15         c = (a + b) / 2.0
16         if f(c) < 0:
17             a = c
18         elif f(c) >= 0:
19             b = c
20         k += 1
21
22     #解の出力
23     print(c, k)
```

4.3 (c) のコード

コード 3: 課題 2(c) のコード

```
def f(x):
    return x**3 + 2.0*(x**2) - 5.0*x - 6.0

def dfdx(x): #f の微分
    return 3*(x**2) + 4*x - 5.0

x0_list = [-3.5, -0.5, 1.5] #x の初期値
epsilon = 0.1 ** 6 #イプシロン
for i in range(3):
    x = x0_list[i]
    k = 0
    while abs(f(x)) > epsilon:
        dx = -f(x) / dfdx(x) #ニュートン方程式を解く
        x += dx
        k += 1
    print(x, k)
```

第 IV 部
課題 3

ここでは、関数 $f : \mathbb{R} \to \mathbb{R}$ を以下のように定義し、課題に回答した。

$$f(x) = \frac{1}{3}x^3 - x^2 - 3x + \frac{5}{3} \tag{4.1}$$

(a) では最急降下法を実装し、関数 f の停留点を求めた。(b) ではニュートン法を実装し、関数 f の停留点を求めた。コードは全て Python で記述した。

5 準備

課題 3 を解くのに必要な前提知識および手法と、その理論をまとめる。

5.1 停留点

関数 f が微分可能である時、$\nabla f(x) = (\frac{\partial f(x)}{\partial [x]_0}, ..., \frac{\partial f(x)}{\partial [x]_{n-1}})^T = 0$ を満たす点 $x \in \mathbb{R}^n$ のことを f の停留点という (ただし $[x]_k$ はベクトル x の k 成分)。本問では関数 (4.1) の停留点を、最急降下法およびニュートン法を用いて求めた。

5.2 降下法

降下法とは、$f(x^0) > f(x^1) > ...$ となる点列 $\{x^k\}$ を生成する手法である。この点列は $x^{k+1} = x^k + t_k d^k$ で与えられる (ただし d^k は探索方向、t_k はステップサイズである。詳細はすぐに述べる)。本問で扱う最急降下法およびニュートン法は降下法の例である。降下法においては、d^k は次の条件を満たすと仮定する。

$$\langle d^k, \nabla f(x^k) \rangle < 0$$

ただし、$\langle , \rangle$ は内積を表す。このような条件を満たすベクトル d^k が降下方向である。降下方向を適切に決れば関数値は減少する。ただし、進みすぎて増加するのを防ぐため、$f(x^k) > f(x^{k+1})$ となるように t_k をうまく設定する必要がある。この t_k をうまく決める方法が直線探索法である。直線探索法には厳密直線探索法やバックトラック法などいくつかの種類がある。詳細は課題 4,5 の章で述べる。

降下法のアルゴリズムは以下である (本レポートで用いる降下法のアルゴリズム共通である)。なお、課題 3 では手順 4 において直線探索法は用いず、(a)、(b) それぞれにおいて定められた方法でステップサイズを決定した。

降下法

手順 1:初期点 x^0、終了条件を $\epsilon > 0$ を決める。$k = 0$ とする。
手順 2:もし x^k が終了条件 $\|\nabla f(x^k)\| \leq \epsilon$ を満たしていれば x^k を解として終了する。
手順 3:

$$\langle d^k, \nabla f(x^k) \rangle < 0$$

を満たす降下方向 d^k を定める。
手順 4:バックトラック法を用いてステップサイズ t_k を決定する。
手順 5:点 x^k を $x^{k+1} = x^k + t_k d^k$ と更新し、$k \leftarrow k + 1$ として手順 2 に戻る。

ただし、反復回数が十分大きい時は解が求まらないと判断するため、通常は許容できる最大反復回数を設定する (ただし、このレポートでは最適解の計算が求められていることから、解が求まることは前提となっていると判断し、最大反復回数は設定しなかった)。また、問題の次元 n が大きくなるにつれて計算誤差が大きくなることを考慮し、この誤差を許容するため $\epsilon = n \times 10^{-6}$ のように設定する。

また、この降下法によって生成される点列 $\{x_k\}$ に対して以下の定理が成り立つことより、うまく降下方向を選べば停留点を求めることができると言える。最急降下法とニュートン法はこの定理における降下方向に関する仮定を満たす (理由は後述する)。

定理 1

点列 $\{x_k\}$ および $\{d_k\}$ はそれぞれ降下法により生成される有界列とし、$\{d_k\}$ は降下方向の列であると仮定した時、ある正の定数 γ が存在し、全ての $k \in \mathbb{N} \cup \{0\}$ に対して

$$\langle d^k, \nabla f(x^k) \rangle < -\gamma \|\nabla f(x^k)\|^2$$

が成り立つならば点列 $\{x_k\}$ の任意の集積点 x^* は関数 f の停留点となり $\nabla f(x^*) = 0$ が成り立つ。

5.2.1 最急降下法

最急降下法は、点 x^k が与えられた時に降下方向として

$$d^k = -\nabla f(x^k)$$

を用いる手法である。$\nabla f(x^k) = 0$ の時 x^k はすでに f の停留点となっているため、$\nabla f(x^k) \neq 0$ とする。すると

$$\langle d^k, \nabla f(x^k) \rangle = -\|\nabla f(x^k)\|^2 < 0$$

が成り立つため d^k は降下方向 (関数 f を減少させる) であり、上で紹介した定理 1 を満たすことがわかる (上記の等号により、定理 1 内での $\gamma = 1$ の存在が確認できるため)。本問の (a) を含め、最急降下法を用いて解く課題ではこの手順を踏んだ。

5.2.2 ニュートン法

ニュートン法は、降下方向として

$$d^k = -\nabla^2 f(x^k)^{-1} \nabla f(x^k)$$

を用いる降下法である。$\nabla^2 f(x)$ は x における f の Hesse 行列を表しており、

$$\begin{pmatrix} \frac{\partial^2 f(x)}{\partial [x]_0^2} & \frac{\partial^2 f(x)}{\partial [x]_0 [x]_1} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_0 [x]_{n-1}} \\ \frac{\partial^2 f(x)}{\partial [x]_1 [x]_0} & \frac{\partial^2 f(x)}{\partial [x]_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_1 [x]_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial [x]_{n-1} [x]_0} & \frac{\partial^2 f(x)}{\partial [x]_{n-1} [x]_1} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_{n-1}^2} \end{pmatrix} \tag{5.1}$$

で表せる。$\{\nabla^2 f(x^k)\}$ が有界であり、かつ一様正定値であるとする。これはある $\mu > 0$ が存在し、全ての $k \in \mathbb{N} \cup \{0\}$ および $v \in \mathbb{R}^n$ に対して

$$\frac{\|v\|^2}{\mu} \leq \langle \nabla f(x^k)^{-1} v, v \rangle \leq \mu \|v\|^2 \tag{5.2}$$

が成立することと同値。この時、全ての k に対して $\nabla f(x^k) \neq 0$ であれば

$$\langle d^k, \nabla f(x^k) \rangle = -\langle \nabla^2 f(x^k)^{-1} \nabla f(x^k), \nabla f(x^k) \rangle \leq -\frac{1}{\mu} \|\nabla f(x^k)\|^2 < 0$$

となるため、d^k は降下方向 (関数 f を減少させる) となり、定理 1 の仮定を満たすと言える。本問の (b) を含め、ニュートン法を用いて解く課題ではこの手順を踏んだ。

5.3 ニュートン方向の修正

ニュートン法を用いるにあたって、$\{\nabla^2 f(x^k)\}$ が有界であり、かつ一様正定値であることが前提となっているがこれは大変厳しい条件である。よって以下に述べるような修正がよく使われる。

ニュートン方向の修正

各反復 k に対して、Hesse 行列 $\nabla^2 f(x^k)$ に単位行列 I の τ 倍を加え、次の方程式を考える。

$$(\nabla^2 f(x^k) + \tau I) d^k = -\nabla f(x^k) \tag{5.3}$$

ただし $\tau > 0$ は $\nabla^2 f(x^k) + \tau I$ が正定値であるようにとる。

ただし、課題 3 においては $\{\nabla^2 f(x^k)\}$ が有界であり、かつ一様正定値であることを前提として課題を解いた。以下、ニュートン法の修正について具体的に述べる。まず、前提として、対称行列 $M \in \mathbb{R}^{n \times n}$ が正定値であるとは以下を満たすことである。

$$\langle Mv, v \rangle > 0 \quad for \quad all \quad v \in \mathbb{R}^n \backslash \{0\}$$

また、対称行列 M が正定値であることは、その固有値 $\lambda_j(M)(j = 1, ..., n)$ が全て正であることと同値である (∗)。一様正定値の定義は (5.2) で与えられる。
$\nabla^2 f(x^k)$ が正定値ならば $\tau = 0$ とすれば良い。$\nabla^2 f(x^k)$ が正定値でない場合を考える。$\nabla^2 f(x^k) + \tau I$ の固有値が $\lambda_j(\nabla^2 f(x^k)) + \tau (j = 1, ..., n)$ となる。よって、τ として $\nabla^2 f(x^k)$ の最小固有値の絶対値に小さい定数 (δ と表記する) を加えた値を設定すれば、$\lambda_j(\nabla^2 f(x^k)) + \tau > 0 (j = 1, ..., n)$ となり、(∗) より $\tau > 0$ は $\nabla^2 f(x^k) + \tau I$ は正定値となる。よって、この τ を用いて、式 (5.3) から探索方向 d^k を決定すれば良い。
本レポートでは、課題 6 と 7 で使うニュートン法において、以上のような修正を行った。なお、修正において $\delta = 10^{-2}$ を共通して用いた。

6　課題 3 の回答

最急降下法およびニュートン法を用いて式 (4.1) で表される関数 $f : \mathbb{R} \to \mathbb{R}$ の停留点を求めた。この関数 f について、$\nabla f(x) = x^3/3 - x^2 - 3x + 5/3$ および $\nabla^2 f(x) = x^2 - 2x - 3$ が成立するため、これを利用した。さらに、一変数であるため $\nabla^2 f(x^k)^{-1}$ の値は $\frac{1}{\nabla^2 f(x^k)}$ となるので、これもニュートン法において利用した。コードは全て Python で記述した。解の存在を前提とし、最大反復回数は設定しなかった。

6.1　(a)

ここでは最急降下法を用い、式 (4.1) で表される関数 $f : \mathbb{R} \to \mathbb{R}$ の停留点を求めた。ただし、初期値は $x_0 = \frac{1}{2}$ とし、各反復 k においてステップサイズは $t_k = \frac{1}{1+k}$ とした (バックトラック法は使用していない)。反復の終了条件 ϵ は 10^{-6} とした。結果は以下である。なお、停留点であることの数値的な確認および後のニュートン法との比較のために、解となった x における f の勾配と収束までに要した反復回数 k も出力した。

表 3: 最急降下法を用いて求めた f の停留点. 左から順に停留点、勾配、反復回数となっている.

停留点	勾配	反復回数
2.999999750304564	-9.987816813605832e-07	62

6.2　(b)

ここでは最急降下法を用い、式 (4.1) で表される関数 $f : \mathbb{R} \to \mathbb{R}$ の停留点を求めた。ただし、初期値は $x_0 = 5$ とし、各反復 k においてステップサイズは $t_k = 1$ とした (バックトラック法は使用していない)。反復の終了条件 ϵ は 10^{-6} とした。結果は以下である。なお、停留点であることの数値的な確認および最急降下法との比較のために、解となった x における f の勾配と収束までに要した反復回数 k も出力した。

表 4: ニュートン法を用いて求めた f の停留点. 左から順に停留点、勾配、反復回数となっている.

停留点	勾配	反復回数
3.0000000929222947	3.716891878724482e-07	4

7　この課題の考察とまとめ

(a)、(b) どちらにおいても停留点はおおよそ 3 に収束した。$\frac{df(x)}{dx} = (x-3)(x+1)$ であるため $x = 3, 1$ が正確な停留点の値となることと、解として出力された x において勾配の値がほぼ 0 となっていることより、この結果は正しいと言える。1 ではなく 3 に収束したのはどちらの手法も d^k が降下方向 (関数 f を減少させる) 方向に x を更新するものであることと、f の極小値を与えるのが $x = 3$ であったことに起因すると考えられる ($x = 1$ は極大値を与える)。

両手法の反復回数を比較すると、ニュートン法の方が最急降下法よりも少ない回数で収束していることがわかる。この現象が起こる理由を以下に与える。x^* を求める問題の解とし、$\{x_k\}$ は x^* に収束するとする。この時、ある $a \in (0, 1)$ と $m_0 \in \mathbb{N}$ が存在して、

$$\|x^{k+1} - x^*\| \leq a\|x^k - x^*\|$$

が m_0 以上のすべての k に対して成立することを、点列 $\{x^k\}$ は解 x^* に一次収束するという。一方、

$$\lim_{k\to\infty} \frac{\|x^{k+1} - x^*\|}{x^k - x^*} < \infty$$

となる時、点列 $\{x^k\}$ は二次収束するという。これはある $b \in (0, 1)$ と $m_0 \in \mathbb{N}$ が存在して、

$$\|x^{k+1} - x^*\| \leq b\|x^k - x^*\|^2$$

が n_0 以上のすべての k に対して成立することを意味しており、$\|x^k - x^*\| < 1$ となったが最後、急速に収束する。

最急降下法はある仮定の元で一次収束するが、最小化する関数の一階微分しか用いていないため二次収束することは少ない。ニュートン法は関数の Hesse 行列を用いているため、ある仮定の元で二次収束する。よってニュートン法の方が収束が速くなる。

本問での実験結果を見ても、以上の理論が正しいことが確認できる。

8　コード

8.1　(a) のコード

コード 4: 課題 3(a) のコード

```
1  def f(x):#関数 f
2      return (x ** 3)/3.0 - x ** 2 - 3.0*x + 5.0/3.0
3
4  def dfdx(x):#f の微分
5      return x ** 2 - 2.0*x - 3.0
```

```
6
7  epsilon = 0.1 ** 6 #イプシロン
8  x = 1.0 / 2.0 #x の初期値
9  k = 0 #繰り返し回数
10
11 while(abs(dfdx(x)) > epsilon):
12     d = -dfdx(x)
13     t = 1.0/(k+1) #ステップサイズ
14     x += (t * d) #更新
15     k += 1
16
17 print(x)
18 print(dfdx(x))
19 print(k)
```

8.2 (b) のコード

コード 5: 課題 3(b) のコード

```
1  def f(x):#関数 f
2      return (x ** 3)/3.0 - x ** 2 - 3.0*x + 5.0/3.0
3
4  def dfdx(x):#f の微分
5      return x ** 2 - 2.0*x - 3.0
6
7  def d2fdx2(x): #f の二階微分
8      return 2.0*x - 2.0
9
10 epsilon = 0.1 ** 6 #イプシロン
11 x = 5.0 #x の初期値
12 k = 0 #繰り返し回数
13
14 while(abs(dfdx(x)) > epsilon):
15     d = -1.0/d2fdx2(x) * dfdx(x)
16     t = 1.0 #ステップサイズ
17     x += (t * d) #更新
18     k += 1
19
20 print(x)
21 print(dfdx(x))
22 print(k)
```

第 V 部
課題 4

ここでは、次の 2 次元の最適化問題を考えた。

$$\begin{cases} minimize \quad f(x) = x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3 \\ subject \quad to \quad x = (x_0, x_1)^T \in \mathbb{R}^2 \end{cases} \tag{8.1}$$

(a) では x が与えられた時 $f(x)$ を出力する関数を作成し、(b) では x が与えられた時勾配ベクトル $\nabla f(x)$ を出力する関数を作成し、(c) では x が与えられた時 Hesse 行列 $\nabla^2 f(x)$ を出力する関数を作成した。

9 準備

課題 4 を解くのに必要な前提知識や手法およびその理論をまとめる。

$x = (x_0, ..., x_{n-1})^T$ を n 次元のベクトルとすると、$f(x)$ の勾配ベクトルは $(\frac{\partial f(x)}{\partial x_0}, ..., \frac{\partial f(x)}{\partial x_{n-1}})^T$ とおける。また、関数 f の Hesse 行列は式 (5.1) で表される。

10 課題 4 の回答

10.1 (a)

式 (8.1) を出力する関数を作成した。作成した関数は以下のコードで表される。引数を x とし、最適化問題 (8.1) の f に従って出力を行う。

コード 6: 課題 4(a) のコード

```
1 import numpy as np
2
3 def f(x):#関数 f(x)を定義
4     x0 = x[0]
5     x1 = x[1]
6     return x0 ** 2 + np.exp(x0) + x1 ** 4 + x1 ** 2 - 2.0*x0*x1 + 3.0
```

10.2 (b)

最適化問題 (8.1) における f の勾配ベクトルを出力する関数を作成した。本問においては x は二次元であるため、第一成分が $\frac{\partial f(x)}{\partial x_0} = 2x_0 + exp(x_0) - 2x_1$、第二成分が $\frac{\partial f(x)}{\partial x_1} = 4x_1^3 + 2x_1 - 2x_0$ となるようなベクトルを構成し、出力する。作成した関数は以下のコードで表される。

コード 7: 課題 4(b) のコード

```
1 import numpy as np
2
3 def grad(x):#f(x) の勾配ベクトルを返す
4     x0 = x[0]
5     x1 = x[1]
6     dfdx0 = 2.0 * x0 + np.exp(x0) - 2.0*x1 #x0 に関して微分
7     dfdx1 = 4.0 * (x1 ** 3) + 2.0*x1 - 2.0*x0 #x1 に関して微分
8     return np.array([dfdx0, dfdx1])
```

10.3 (c)

最適化問題 (8.1) の f の Hesse 行列を出力する関数を作成した。本問においては x は二次元であるため Hesse 行列は以下のように表される。成分はそれぞれ、1 行 1 列成分が $\frac{\partial^2 f(x)}{\partial x_0 x_0} = 2e^{x_0}$、1 行 2 列成分が $\frac{\partial^2 f(x)}{\partial x_0 x_1} = -2$、2 行 1 列成分が $\frac{\partial^2 f(x)}{\partial x_1 x_0} = -2$、2 行 2 列成分が $\frac{\partial^2 f(x)}{\partial x_1 x_1} = 12x_1 + 2$ である。

$$\begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_0 x_0} & \frac{\partial^2 f(x)}{\partial x_0 x_1} \\ \frac{\partial^2 f(x)}{\partial x_1 x_0} & \frac{\partial^2 f(x)}{\partial x_1 x_1} \end{pmatrix} \tag{10.1}$$

作成した関数は以下のコードで表される。

コード 8: 課題 4(c) のコード

```
1  import numpy as np
2
3  def hesse(x):#f(x) の hesse 行列を返す
4      x0 = x[0]
5      x1 = x[1]
6      x00 = 2.0 + np.exp(x0) #d^2f/dx0^2
7      x01 = -2.0 #d^2f/ dx0dx1
8      x10 = -2.0 #d^2f / dx1x0
9      x11 = 12.0 * x1 + 2.0 #d^2f / dx1x1
10     return np.array([[x00, x01],[x10, x11]])
```

11 この課題の考察とまとめ

以上の 3 問により、最適化問題 (8.1) における関数 $f(x)$、$f(x)$ の勾配ベクトルと Hesse 行列の定義を行なった。課題 5 において、これを用いて関数 $f(x)$ の停留点を求めるため、考察は次の部にまわす。

第 VI 部
課題 5

ここでは、バックトラック法を利用した最急降下法およびニュートン法を用いて最適化問題 (8.1) を解いた。解の存在を前提とし、最大反復回数は設定しなかった。

12 準備

課題 5 を解くために必要となる前提知識、手法およびその理論について述べる。

12.1 直線探索法

先述したとおり、直線探索法はステップサイズ $t_k(0 < t_k)$ を決定するための手法である。点 x_k を x_{k+1} に更新する時、$f(x_{k+1}) < f(x_k)$ を満たすように行う。厳密直線探索法やバックトラック法などの種類がある。厳密直線探索法は、半直線 $x + t\Delta x(t > 0)$ 上の f を最小化する t を選択するもので、

$$t_k = argmin_{0 \leq s} \quad f(x^k + s\Delta x^k)$$

で表される。
本問で用いるのはバックトラック法である。以下にバックトラック法をまとめる。
バックトラック法のアルゴリズムは以下で表される。

バックトラック法

手順 1:$\xi \in (0,1), \rho \in (0,1)$、初期ステップサイズ $\bar{t}$ を選択し、$t = \bar{t}$ とする。
手順 2:次式を満たすまで、$t \leftarrow \rho t$ を繰り返す。

$$f(x^k + td^k) \leq \quad f(x^k) + \xi t\langle d^k, \nabla f(x^k)\rangle$$

手順 3:$t_k = t$ として終了する。

バックトラック法の意味を以下に述べる。
$\phi(t) = f(x^k + td^k)$ について考える。$t = 0$ における ϕ の接線は $y = f(x^k) + t\langle d^k, \nabla f(x^k)\rangle$ である。手順 2 においてこの不等式を満たすステップサイズを選ぶことは、ϕ の接線の傾きを緩めて得られる直線 $y = f(x^k) + \xi t\langle d^k, \nabla f(x^k)\rangle$ よりも小さい関数値を与える t の区間からステップサイズを選ぶことに対応する。これはバックトラック法が、反復後に関数値を小さくするような t_k を与えるアルゴリズムとして適切に機能していることを意味する。

13 課題 5 の回答

Python を用い、バックトラック法を用いた最急降下法およびニュートン法により最適化問題 8.1 を解いた。最急降下法およびニュートン法は停留点を求める手法であり、その点が定義域において関数 f の最小値を与えるかどうかは保証しない。しかし、以下のグラフからわかるように今回与えられた関数 f は停留点を一つしか持っておらず、さらにその点において最小値をとる凸な関数であると予想されるため、最急降下法およびニュートン法を用いて得られた解は本問の題意に適する解であると言える。

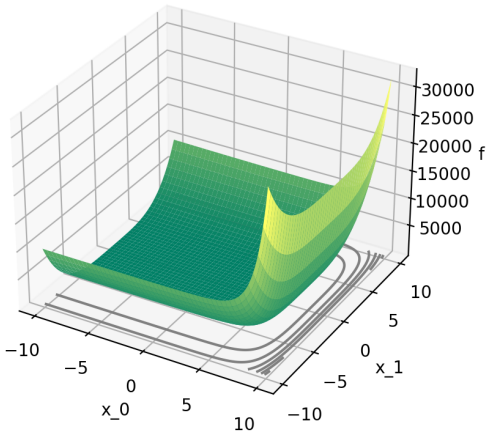

図 2: 最適化問題 8.1 における関数 f のグラフ.2 つの横軸が x_0, x_1 を表し、縦軸が f を表す. 停留点が一つしかない凸なグラフであり、停留点が f に最小値を与えることがわかる.

13.1 (a)

ここではバックトラック法を用いた最急降下法を実装し、最適化問題 8.1 を解いた。ただし、初期点は $x^0 = (1,1)^T$ とし、バックトラック法における $\xi, \rho, \bar{t}$ はそれぞれ $\xi = 10^{-4}, \rho = 0.5, \bar{t} = 1$ とした。収束判定に用いる ϵ は 10^{-6} とした。関数 f および f の勾配ベクトルを返す関数は課題 4 で作成したものを用いた。
結果は以下である。これより、最適解は $(-0.73345152, -0.49332751)^T$、最適値は 3.597138024959682 となった。なお、本当に停留点を求められているかどうかの確認、および (b) のニュートン法との比較のため、解として出力された停留点における勾配ベクトルの値と、収束までに要した反復回数と実行時間を出力した。表の上から順に最適解、最適値、勾配ベクトル、反復回数、実行時間である。

13.2 (b)

ここではバックトラック法を用いたニュートン法を実装し、最適化問題 8.1 を解いた。ただし、初期点は $x^0 = (1,1)^T$ とし、バックトラック法における

表 5: 最急降下法を用いて最適化問題を解いた結果. 上から順に最適解、最適値、勾配ベクトル、反復回数、実行時間となっている.

最適解	$(-0.73345152, -0.49332751)^T$
最適値	3.597138024959682
勾配ベクトル	$(5.09815666e-07, -4.35599102e-07)^T$
反復回数	31
実行時間	0.0016918182373046875(s)

$\xi, \rho, \bar{t}$ はそれぞれ $\xi = 10^{-4}, \rho = 0.5, \bar{t} = 1$ とした。収束判定に用いる ϵ は 10^{-6} とした。なお、ここでも課題 3 と同様、$\{\nabla^2 f(x^k)\}$ が有界であり、かつ一様正定値であることを前提として課題を解いた。
関数 f および f の勾配ベクトル、Hesse 行列を返す関数は課題 4 で作成したものを用いた。ニュートン法 (課題 3 の準備を参照) にて降下方向を決める際、

$$d^k = -\nabla^2 f(x^k)^{-1} \nabla f(x^k)$$

を計算するために Hesse 行列の逆行列を求める必要がある。今回は Hesse 行列が 2×2 行列となるため、

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$$

の逆行列として

$$\frac{1}{a_{00}a_{11} - a_{01}a_{10}} \begin{pmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{pmatrix}$$

を返す関数を定義した。なお、後の課題 7 のように、Python の numpy で提供される数値計算関数 $linalg.inv(matrix)$ を用いても同様の解を得ることができる。さらに、同じく降下方向を決定する際に行列とベクトルの掛け算を行う必要が出てくる。本問ではそのための関数を定義したが、後の課題 7 などのように Python の numpy で提供される数値計算関数 $dot(matrix, matrix)$ を用いることでも同様の解が得られる。
結果は以下である。これより、最適解は $(-0.73345172, -0.4933275)^T$、最適値は 3.597138024959629 となることがわかった。なお、本当に停留点を求められているかどうかの確認、および (a) の最急降下法との比較のため、解として出力された停留点における勾配ベクトルの値と、収束までに要した反復回数と実行時間を出力した。表の上から順に停留点、勾配ベクトル、反復回数、実行時間である。

表 6: ニュートン法を用いて最適化問題を解いた結果. 上から順に最適解、最適値、勾配ベクトル、反復回数、実行時間となっている.

最適解	$(-0.73345172, -0.4933275)^T$
最適値	3.597138024959629
勾配ベクトル	$(9.43689571e-14, -3.61755070e-12)^T$
反復回数	6
実行時間	0.00033783912658691406(s)

14 この課題の考察とまとめ

(a)、(b) どちらにおいても解として出力された停留点における勾配ベクトルはほぼ (0,0) となっているため、実装したアルゴリズムは意図した動作をしていることが確認できる。ただし、ニュートン法を用いた (b) の結果の方が最急降下法を用いた (a) よりも勾配ベクトルの値がより 0 に近いことから、本問の場合はニュートン法の方がより正確な値を求められていることがわかる。さらに、反復回数を比較するとニュートン法で 6 回、最急降下法で 31 回であることから、ニュートン法の収束の速さを確認することができる。この現象を説明する理論は課題 3 の考察に記しているとおりである。
また、収束までに最急降下法ではニュートン法の約 5 倍の時間がかかった。ニュートン法における反復回数が最急降下法の約 $\frac{1}{5}$ であることから、一回の反復にかかる時間は両者でさほど変わりないと結論づけることができる。ニュートン法では Hesse 行列の計算を行いニュートン方程式 $\nabla^2 f(x^k)d^k \ -\nabla f(x^k)$ を解かなければならないため、一般に一回の反復にかかる時間は長くなる。しかし本問では Hesse 行列が 2×2 行列であったことからそこまでの計算負荷がかからず、最急降下法と同程度の計算時間で一回の反復が可能だったのではないかと考察する。

15 コード

15.1 関数 f を図示するコード

コード 9: 関数 f を図示するコード

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5 fig = plt.figure()
6 ax = fig.add_subplot(111, projection="3d")
7
8 #範囲の設定
9 x_range = np.linspace(−10, 10, 1000)
```

```
y_range = np.linspace(−10, 10, 1000)
x, y = np.meshgrid(x_range, y_range)

#関数 f を記述
z = x**2+np.exp(x)+y**4+y**2−2.0*x*y+3.0

ax.plot_surface(x, y, z, cmap = "summer")
ax.contour(x, y, z, colors = "gray", offset = −1) #底面に等高線を描画

ax.set_xlabel("x_0")
ax.set_ylabel("x_1")
ax.set_zlabel("f")

plt.show()
```

15.2 (a) のコード

コード 10: (a) のコード

```
import numpy as np
import time

def f(x):#関数 f(x)を定義
    x0 = x[0]
    x1 = x[1]
    return x0 ** 2 + np.exp(x0) + x1 ** 4 + x1 ** 2 − 2.0*x0*x1 + 3.0

def grad(x):#f(x) の勾配ベクトルを返す
    x0 = x[0]
    x1 = x[1]
    dfdx0 = 2.0 * x0 + np.exp(x0) − 2.0*x1 #x0 に関して微分
    dfdx1 = 4.0 * (x1 ** 3) + 2.0*x1 − 2.0*x0 #x1 に関して微分
    return np.array([dfdx0, dfdx1])


def backtrack(x, d):#backtrack 法
    #パラメータ
    ita = 10.0 ** (−4)
    rou = 0.5
    t_kari = 1.0
    #反復
    while f(x + t_kari*d) > f(x) + ita * t_kari * np.sum(d*grad(x)):
        t_kari = rou * t_kari
    return t_kari

def norm(x): #ベクトルのノルムを計算
    return np.sqrt(np.sum(x ** 2))

epsilon = 0.1 ** 6 #イプシロン
x = np.array([1.0, 1.0]) #x の初期値
k = 0 #繰り返し回数

t1 = time.time()
while(norm(grad(x)) > epsilon):
    d = −grad(x)
    t = backtrack(x, d) #ステップサイズ
    x += (t * d) #更新
    k += 1
t2 = time.time()
```

```
41
42 print(x)
43 print(grad(x))
44 print(k)
45 print(t2−t1)
```

15.3 (b) のコード

コード 11: (b) のコード

```
1  import numpy as np
2  import time
3
4  def f(x):#関数 f(x)を定義
5      x0 = x[0]
6      x1 = x[1]
7      return x0 ** 2 + np.exp(x0) + x1 ** 4 + x1 ** 2 − 2.0*x0*x1 + 3.0
8
9  def grad(x):#f(x) の勾配ベクトルを返す
10     x0 = x[0]
11     x1 = x[1]
12     dfdx0 = 2.0 * x0 + np.exp(x0) − 2.0*x1 #x0 に関して微分
13     dfdx1 = 4.0 * (x1 ** 3) + 2.0*x1 − 2.0*x0 #x1 に関して微分
14     return np.array([dfdx0, dfdx1])
15
16 def hesse(x):#f(x) の hesse 行列を返す
17     x0 = x[0]
18     x1 = x[1]
19     x00 = 2.0 + np.exp(x0) #d^2f/dx0^2
20     x01 = −2.0 #d^2f/ dx0dx1
21     x10 = −2.0 #d^2f / dx1x0
22     x11 = 12.0 * (x1**2) + 2.0 #d^2f / dx1x1
23     return np.array([[x00, x01],[x10, x11]])
24
25
26 def backtrack(x, d):#backtrack 法
27     #パラメータ
28     ita = 10.0 ** (−4)
29     rou = 0.5
30     t_kari = 1.0
31     #反復
32     while f(x + t_kari*d) > f(x) + ita * t_kari * np.sum(d*grad(x)):
33         t_kari = rou * t_kari
34     return t_kari
35
36 def norm(x): #ベクトルのノルムを計算
37     return np.sqrt(np.sum(x ** 2))
38
39 def inverse(x):#行列 x の逆行列を計算
40     x00 = x[0][0]
41     x01 = x[0][1]
42     x10 = x[1][0]
43     x11 = x[1][1]
44     a00 = x11
45     a01 = −x01
46     a10 = −x10
47     a11 = x00
48     return np.array([[a00, a01],[a10, a11]]) / (a00*a11 − a01*a10)
49
```

```
50 def mat_cross_vec(A, b):#行列 A とベクトル b の掛け算
51     vec = []
52     length = len(b)
53     for i in range(length):
54         vec.append(np.sum(A[i]*b))
55     return np.array(vec)
56
57 epsilon = 0.1 ** 6 #イプシロン
58 x = np.array([1.0, 1.0]) #x の初期値
59 k = 0 #繰り返し回数
60
61 #反復
62 t1 = time.time()
63 while(norm(grad(x)) > epsilon):
64     d = -mat_cross_vec(inverse(hesse(x)),grad(x))
65     t = backtrack(x, d) #ステップサイズ
66     x += (t * d) #更新
67     k += 1
68 t2 = time.time()
69
70 print(x)
71 print(grad(x))
72 print(k)
73 print(t2 - t1)
```

第 VII 部

課題 6

ここでは最適化問題

$$\begin{cases} minimize \quad f(x) = \sum_{i=0}^{2} f_i(x)^2 \\ subject \quad to \quad x \in \mathbb{R}^2 \end{cases} \tag{15.1}$$

を、最急降下法および修正を加えたニュートン法で解いた。ただし、$f_i(x) = y_i - [x]_0(1 - [x]_1^{i+1})(i = 0, 1, 2)$ であり、$y_0 = 1.5, y_1 = 2.25, y_2 = 2.625$ である。$[x]_0, [x]_1$ はそれぞれ x ベクトルの第一成分、第二成分を表す。以下これらは x_0, x_1 と表記する。ニュートン法の修正の具体的な内容については課題 3 にて説明したとおりである。

16　準備

本問を解くのに必要となる前提知識および手法とその理論は、これまでの課題の準備にて説明している。よって本項目は省略する。

17　課題 6 の回答

ここでは、最適化問題 15.1 を最急降下法および修正を加えたニュートン法を用いて解いた。ただし、$f_i(x) = y_i - x_0(1 - x_1^{i+1})(i = 0, 1, 2)$ であり、$y_0 = 1.5, y_1 = 2.25, y_2 = 2.625$ である。初期点は $x^0 = [2, 0]^T$ を用い、バックトラック法における $\xi, \rho, \bar{t}$ はそれぞれ $\xi = 10^{-4}, \rho = 0.5, \bar{t} = 1$ と設定した。また、収束判定に用いる ϵ は $\epsilon = 10^{-6}$ とした。この課題では修正したニュートン法を用いた。ニュートン方向の修正に使用する τ(課題 3 におけるニュートン方向の修正の項目を参照のこと) を求める過程にて、最小固有値の絶対値に加算する値 δ は 10^{-2} とした。解の存在を前提とし、最大反復回数は設定しなかった。
まず、課題 5 と同様に、f の停留点を求めればそれが最適解となっていること、すなわち f が凸な関数であることを確認した。f を $-10 \leq x \leq 10, -10 \leq y \leq 10$ の範囲で図示した結果が以下である。この図より、関数 f は凸であることが予想され、最適解を求めるためには停留点を求めることが必要かつ十分であると判断した。

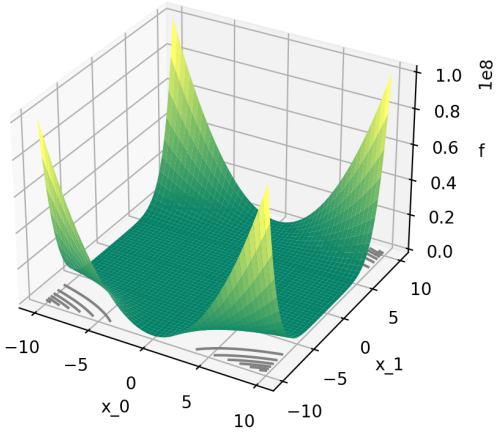

図 3: 最適化問題 15.1 における関数 f のグラフ.2 つの横軸が x_0, x_1 を表し、縦軸が f を表す. 停留点が一つしかない凸なグラフであり、停留点が f に最小値を与えることがわかる.

本問において、関数 f の勾配と Hesse 行列はそれぞれ次のようになる。

$$\begin{cases} \nabla f(x) = 2\sum_{i=0}^{2} f_i(x)\nabla f_i(x) \\ \nabla^2 f(x) = 2\sum_{i=0}^{2}(f_i(x)\nabla^2 f_i(x) + \nabla f_i(x)\nabla f_i(x)^T) \end{cases}$$

以下、これを示す。
$\nabla f(x)$ は以下のように変形できる。

$$\nabla f(x) = \nabla \sum_{i=0}^{2} f_i(x^2) = \sum_{i=0}^{2} \nabla (f_i(x)^2) = \sum_{i=0}^{2} 2 f_i(x)\nabla f_i(x)$$

$\nabla^2 f(x)$ は以下のように変形できる。

$$\begin{aligned} \nabla^2 f(x) = \nabla(\nabla f(x)) = \\ 2\nabla \sum_{i=0}^{2} f_i(x)\nabla f_i(x) \\ = 2\sum_{i=0}^{2} \nabla (f_i(x)\nabla f_i(x)) \\ = 2\sum_{i=0}^{2}(\nabla f_i(x)\nabla f_i(x)^T + f_i(x)\nabla^2 f_i(x)) \end{aligned}$$

以上より示された。ここで、

$$\nabla f_i(x) = \begin{pmatrix} \frac{\partial f_i(x)}{\partial x_0} \\ \frac{\partial f_i(x)}{\partial x_1} \end{pmatrix}$$

であり、

$$\nabla^2 f_i(x) = \begin{pmatrix} \frac{\partial^2 f_i(x)}{\partial x_0 \partial x_0} & \frac{\partial^2 f_i(x)}{\partial x_0 \partial x_1} \\ \frac{\partial^2 f_i(x)}{\partial x_1 \partial x_0} & \frac{\partial^2 f_i(x)}{\partial x_1 \partial x_1} \end{pmatrix}$$

である。ただし、$f_i(x) = y_i - [x]_0(1 - [x]_1^{i+1})(i = 0, 1, 2)$ なので、

$$\begin{cases} \frac{\partial f_i(x)}{\partial x_0} = x_1^{i+1} - 1 \\ \frac{\partial f_i(x)}{\partial x_1} = (i+1)x_0 x_1^i \\ \frac{\partial^2 f_i(x)}{\partial x_0 \partial x_0} = 0 \\ \frac{\partial^2 f_i(x)}{\partial x_0 \partial x_1} = (i+1)x_1^i \\ \frac{\partial^2 f_i(x)}{\partial x_1 \partial x_0} = (i+1)x_1^i \\ \frac{\partial^2 f_i(x)}{\partial x_1 \partial x_1} = i(i+1)x_0 x_i^{i-1}(i \neq 0) \\ \frac{\partial^2 f_i(x)}{\partial x_1 \partial x_1} = 0(i = 0) \end{cases}$$

である。$i(i+1)x_0 x_i^{i-1}$ の値は $i = 0$ においては定義できないため、この場合のみ $f_0(x) = y_0 - x_0(1 - x_1)$ を x_1 で二階微分した値である 0 を用いて表記

した。
コード内においては、行列の固有値を求めるのに numpy の $linalg.eig(matrix)$ 関数を、行列とベクトルの掛け算に numpy の $dot(matrix, matrix)$ 関数を、逆行列の計算に numpy の $linalg.inv(matrix)$ 関数を用いた。
以上の事項を用いて計算した結果は以下の表の通りとなった。ただし、上の表が最急降下法を用いた結果、下の表がニュートン法を用いた結果となっている。それぞれの表の上の項目から順に、停留点 (解)、停留点における f の勾配ベクトル、収束までに要した反復回数、収束までに要した時間、停留点における f の値 (最適値) である。

表 7: 最急降下法を用いて最適化問題 (15.1) を解いた結果. 上の項目から順に停留点 (解)、停留点における f の勾配ベクトル、収束までに要した反復回数、収束までに要した時間、停留点における f の値 (最適値) を表す.

停留点	(2.99999781, 0.49999946)
勾配ベクトル	(-7.86377584e-07 , 3.46495487e-07)
反復回数	791
時間	0.0898580551147461
最適値	7.676324891043213e-13

表 8: ニュートン法を用いて最適化問題 (15.1) を解いた結果. 上の項目から順に停留点 (解)、停留点における f の勾配ベクトル、収束までに要した反復回数、収束までに要した時間、停留点における f の値 (最適値) を表す.

停留点	(2.99999794, 0.49999949)
勾配ベクトル	(-6.21551290e-07 ,-1.55117534e-07)
反復回数	23
時間	0.004372835159301758
最適値	6.806553025312737e-13

18　この課題の考察とまとめ

以上の結果からわかったことを述べる。
まず、どちらの方法においても最適解はおよそ (3,0.5) であること、最適値はおよそ 0 であることがわかった。最適解として出力されている x における勾配ベクトルもほぼ (0,0) となっていることから、確かに最適解を求められていることが確認できた。
収束までに要した反復回数を二つの方法で比較すると、ニュートン法は最急降下法のおよそ $\frac{1}{34}$ の回数で収束していることがわかった。これはニュートン

法の収束の速さを数値的に裏付けている (理論は課題 3 を参照)。また、収束までにかかった時間を反復回数で割ることにより、一回の反復にかかった時間を求めることができる。これを行うことにより、一回の反復にかかる時間はニュートン法が最急降下法のおよそ 1.67 倍であることがわかった。これは課題 5 とは違う結果 (課題 5 ではほぼ変わらなかった) である。課題 5 と同じく Hesse 行列は 2 次であるから、逆行列の計算においてはそこまで計算時間に差はつかないと考えられる。よって、計算時間が両者の間で異なる原因の一つとして、Hesse 行列の導出に必要な手続きの多さがあるのではないかと考察する。

19 コード

19.1 関数 f を図示するコード

コード 12: 関数 f を図示するコード

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

#範囲の設定
x_range = np.linspace(−10, 10, 1000)
y_range = np.linspace(−10, 10, 1000)
x, y = np.meshgrid(x_range, y_range)

#関数 f を記述
z = (1.5−x∗(1.0−y))∗∗2 + (2.25−x∗(1.0−y∗∗2))∗∗2 + (2.625−x∗(1.0−y
    ∗∗3))∗∗2

ax.plot_surface(x, y, z, cmap = "summer")
ax.contour(x, y, z, colors = "gray", offset = −1) #底面に等高線を描画

ax.set_xlabel("x_0")
ax.set_ylabel("x_1")
ax.set_zlabel("f")

plt.show()
```

19.2 課題 6 のコード

コード 13: 課題 6 のコード

```
import numpy as np
import time

y = np.array([1.5, 2.25, 2.625])

```

```
6  def f(x):#関数 f(x)を定義
7      f = 0.0
8      x0 = x[0]
9      x1 = x[1]
10     for i in range(3):
11         f += (y[i]−x0∗(1.0−x1∗∗(i+1)))∗∗2
12     return f
13
14 def grad(x):#f(x) の勾配ベクトルを返す
15     fi = []
16     x0 = x[0]
17     x1 = x[1]
18     sum_0 = 0.0
19     sum_1 = 0.0
20     for i in range(3):
21         fi.append(y[i]−x0∗(1.0−x1∗∗(i+1)))
22     for i in range(3):
23         dfdx0 = x1∗∗(i+1)−1#x0 に関して微分
24         dfdx1 = (i+1)∗x0∗(x1∗∗i) #x1 に関して微分
25         sum_0 += (dfdx0∗fi[i])
26         sum_1 += (dfdx1∗fi[i])
27
28     return np.array([sum_0∗2, sum_1∗2])
29
30 def hesse(x):#f(x) の hesse 行列を返す
31     x0 = x[0]
32     x1 = x[1]
33     ans = np.array([[0.0, 0.0],[0.0, 0.0]])
34
35     for i in range(3):
36         fi = y[i] − x0∗(1.0 − x1∗∗(i+1))
37         dfidx0 = x1∗∗(i+1)−1#x0 に関して微分
38         dfidx1 = (i+1)∗x0∗(x1∗∗i) #x1 に関して微分
39         d2fidx0x0 = 0.0
40         d2fidx0x1 = (i+1)∗(x1∗∗i)
41         d2fidx1dx0 = (i+1)∗(x1∗∗i)
42         if i == 0:
43             d2fidx1dx1 = 0.0
44         else:
45             d2fidx1dx1 = i ∗ (i+1) ∗ x0 ∗ (x1 ∗∗ (i−1))
46
47         A = np.array([[0.0, 0.0],[0.0, 0.0]])
48         B = np.array([[0.0, 0.0],[0.0, 0.0]])
49
50         A[0][0] = fi ∗ d2fidx0x0
51         A[0][1] = fi ∗ d2fidx0x1
52         A[1][0] = fi ∗ d2fidx1dx0
53         A[1][1] = fi ∗ d2fidx1dx1
54         B[0][0] = dfidx0∗dfidx0
55         B[0][1] = dfidx0∗dfidx1
56         B[1][0] = dfidx1∗dfidx0
57         B[1][1] = dfidx1∗dfidx1
58
59         ans += (A + B)∗2.0
60     return ans
61
62 def backtrack(x, d):#backtrack 法
63     #パラメータ
64     ita = 10.0 ∗∗ (−4)
65     rou = 0.5
66     t_kari = 1.0
67     #反復
68     while f(x + t_kari∗d) > f(x) + ita ∗ t_kari ∗ np.sum(d∗grad(x)):
```

```
69              t_kari = rou * t_kari
70          return t_kari
71
72  def norm(x): #ベクトルのノルムを計算
73          return np.sqrt(np.sum(x ** 2))
74
75  epsilon = 0.1 ** 6 #イプシロン
76
77  #最急降下法
78  x = np.array([2.0, 0.0]) #x の初期値
79  k = 0 #繰り返し回数
80
81  t1 = time.time()
82  while(norm(grad(x)) > epsilon):
83          d = -grad(x)
84          t = backtrack(x, d) #ステップサイズ
85          x += (t * d) #更新
86          k += 1
87  t2 = time.time()
88
89  print(x)
90  print(grad(x))
91  print(k)
92  print(t2 - t1)
93  print(f(x))
94
95  #ニュートン法
96  x = np.array([2.0, 0.0]) #x の初期値
97  k = 0 #繰り返し回数
98
99  t1 = time.time()
100 while(norm(grad(x)) > epsilon):
101         w,v = np.linalg.eig(hesse(x))
102         tau = abs(np.min(w)) + 10.0 ** (-2)
103         #修正ニュートン法
104         d = -np.dot(np.linalg.inv(hesse(x)+tau*np.eye(2)),grad(x))
105         t = backtrack(x, d) #ステップサイズ
106         x += (t * d) #更新
107         k += 1
108 t2 = time.time()
109
110 print(x)
111 print(grad(x))
112 print(k)
113 print(t2 - t1)
114 print(f(x))
```

第 VIII 部

課題 7

ここでは、Python を用いて最急降下法およびニュートン法を実装し、最適化問題

$$\begin{cases} minimize \quad f(x) = \frac{1}{2}\langle Ax, x\rangle \\ subject \quad to \quad x \in \mathbb{R}^n \end{cases} \tag{19.1}$$

を解いた。ただし、$A \in \mathbb{R}^n$ は [0,1] の範囲の乱数を各要素に持つ行列 $Z \in \mathbb{R}^{n \times n}$ を用いて $A = Z^T Z$ により定義される半正定値対称行列である。乱数は [0,1] の範囲の一様分布から生成した。これは numpy の提供する $random.rand(dimension, dimension)$ 関数を用いて実現した。初期点は要素が全て 1 のベクトルを用い、バックトラック法における $\xi, \rho, \bar{t}$ はそれぞれ $\xi = 10^{-4}, \rho = 0.5, \bar{t} = 1$ とした。この課題では修正したニュートン法 (詳細は課題 3 を参照) を用いた。ニュートン法の修正において τ に加える定数は 10^{-2} とした。また、収束条件に用いる ϵ は 10^{-6} とした。この時、$n = 2, 5, 10$ それぞれに対し 5 回計算を実行し、各手法と各 n に対して反復回数の平均を計算した。ただし、5 回の計算では、毎回 A を新たに生成した。解の存在を前提とし、最大反復回数は設定しなかった。

20 準備

この課題を解くのに必要な前提知識はこれまでの課題にて述べているため、このセクションは省略する。

21 課題の回答

f の勾配と Hesse 行列を求める。
勾配に関しては以下の式が成り立つ。

$$\begin{aligned}\nabla f(x) &= \nabla \frac{1}{2}\langle Ax, x\rangle = \frac{1}{2}\nabla\langle Ax, x\rangle \\ &= \frac{1}{2}\nabla((Ax)^T x) = \frac{1}{2}2Ax = Ax\end{aligned}$$

Hesse 行列に関しては以下の式が成り立つ。

$$\nabla^2 f(x) = \nabla(Ax) = A\nabla x = A$$

以上より両手法において、勾配、Hesse 行列としてそれぞれ Ax, A を用いた。結果は以下の表の通りとなった。上の表から順に、$n = 2$ の時の最急降下法の結果、$n = 2$ の時のニュートン法の結果、$n = 5$ の時の最急降下法の結果、$n = 5$ の時のニュートン法の結果 (以上 4 つの表には最適解および最適値を掲載)、$n = 10$ の時の最急降下法の最適解、$n = 10$ の時のニュートン法の最適解、$n = 10$ の時の最急降下法の最適値、$n = 10$ の時のニュートン法の最適値を表す。ただし、ベクトルは全て横ベクトルで表記している。

表 9: $n = 2$ の時の最急降下法の結果. 上から順に 1 回目から 5 回目まで の計算で得られた停留点 (最適解) と最適値の値を記している. 下の二つの項目は収束までに要した 反復回数 k の平均とかかった時間の平均である.

最急降下法 1 回目の計算	
停留点	(7.120226811033507e-05,-3.972621743646066e-05)
最適値	4.0694491181995643e-11
最急降下法 2 回目の計算	
停留点	(-4.331243529587426e-06,5.8993181000580425e-06)
最適値	3.093124340151268e-12
最急降下法 3 回目の計算	
停留点	(-1.7155785703470527e-05,3.7626746560188244e-05)
最適値	2.0341491741059613e-11
最急降下法 4 回目の計算	
停留点	(0.000196116901439849,-8.704827224549435e-05)
最適値	1.0692163852436396e-10
最急降下法 5 回目の計算	
停留点	(0.00095838603057631 51,-7.171536882464201e-05)
最適値	4.804676358203406e-10
$n = 2$ の時の最急降下法の k の平均	
1886.6	
$n = 2$ の時の最急降下法の平均時間	
0.039397144317626955	

表 10: $n = 2$ の時のニュートン法の結果. 上から順に 1 回目から 5 回目までの計算で得られた停留点 (最適解) と最適値 の値を記している. 下の二つの項目は収束までに要した反復回数 k の平均とかかった時間の平均である.

ニュートン法 1 回目の計算	
停留点	(-5.153285605303484e-06,5.384370034271967e-06)
最適値	2.4564854325973017e-12
ニュートン法 2 回目の計算	
停留点	(4.673802048739853e-05,-3.9798296357512804e-05)
最適値	2.3615328728356087e-11
ニュートン法 3 回目の計算	
停留点	(-0.00014686808449519443,0.0001631654068892077)
最適値	9.753470093168322e-11
ニュートン法 4 回目の計算	
停留点	(-1.6552493665020165e-05,3.3149826149430825e-05)
最適値	1.2207390128217195e-11
ニュートン法 5 回目の計算	
停留点	(0.00016491452442972594,-5.791682158408651e-05)
最適値	6.94789814351928e-11
$n = 2$ の時のニュートン法の k の平均	
20.4	
$n = 2$ の時のニュートン法の平均時間	
0.0015887737274169922	

表 11: $n = 5$ の時の最急降下法の結果. 上から順に 1 回目から 5 回目まで の計算で得られた停留点 (最適解) と最適値の値を記している. 下の二つの項目は収束までに要した 反復回数 k の平均とかかった時間の平均である.

最急降下法 1 回目の計算	
停留点	(1.4439277090815203e-05,-1.3685801169359159e-05, -9.86675608974041e-06,5.502585871750754e-06, 7.911870525107084e-06)
最適値	9.007652198214156e-12
最急降下法 2 回目の計算	
停留点	(0.00010261438242966822,1.7448545124434954e-05, 4.985306129756685e-05,-7.210047314932489e-05, -6.818078294873635e-05)
最適値	5.784281497021978e-11
最急降下法 3 回目の計算	
停留点	(-7.718331875688409e-06,-2.7879190054060714e-05, 2.1056986965171018e-05,8.02152842703497e-06, 2.9450182913760995e-05)
最適値	1.8689539626998104e-11
最急降下法 4 回目の計算	
停留点	(2.142206438617686e-05,-7.13098727910072e-07, 3.5177489043476792e-06,-2.0370739674408986e-05, -3.623293358121266e-06)
最適値	1.0431208014541875e-11
最急降下法 5 回目の計算	
停留点	(0.0003381035415972823,-0.0002199212662231622, 0.00013787977712927 2,2.9722854472449757e-05, -0.00024753606556511076)
最適値	2.3639935393786106e-10
$n = 5$ の時の最急降下法の k の平均	
2793.0	
$n = 5$ の時の最急降下法の平均時間	
0.11287097930908203	

表 12: $n = 5$ の時のニュートン法の結果. 上から順に 1 回目から 5 回目までの計算で得られた停留点 (最適解) と最適値 の値を記している. 下の二つの項目は収束までに要した反復回数 k の平均とかかった時間の平均である.

ニュートン法 1 回目の計算	
停留点	(9.056868412446562e-05,3.241910016229305e-05, -2.6150691932901463e-05,-6.333993209131262e-05, 7.299666883653399e-07)
最適値	4.3542368107133876e-11
ニュートン法 2 回目の計算	
停留点	(9.469892710492463e-06,-2.0223933698991586e-05, -3.2140067705312576e-05,2.2879184589996424e-05, 3.5331073103754825e-05)
最適値	2.097049361563807e-11
ニュートン法 3 回目の計算	
停留点	(0.027416128494099214,0.012259494323115485, -0.02833759318731096,0.028119929299079343, -0.021684379839984905)
最適値	2.7224764311022522e-08
ニュートン法 4 回目の計算	
停留点	(-0.0001833663585169933,-0.00023369190267998665, 0.00017226344081195755,-1.1129642143967834e-05, 0.000325324473554397)
最適値	2.1077107170504007e-10
ニュートン法 5 回目の計算	
停留点	(-2.4688379622204596e-06,1.6446003042172743e-05, -1.702150559722066e-06,-4.599642302135603e-06, -1.6697417293122095e-06)
最適値	5.690559612791889e-12
$n = 5$ の時のニュートン法の k の平均	
215.4	
$n = 5$ の時のニュートン法の平均時間	
0.020704889297485353	

表 13: $n = 10$ の時の最急降下法の結果. 上から順に 1 回目から 5 回目まで の計算で得られた停留点 (最適解) と最適値の値を記している. 下の二つの項目は収束までに要した 反復回数 k の平均とかかった時間の平均である.

最急降下法 1 回目の計算	
停留点	(-5.318810059990533e-06,-2.791948316799226e-06, -1.2757664917784455e-05,3.257327155719891e-06, -5.631061245695976e-06,1.316493965281553e-05, 7.732418246814598e-06,7.438720329918048e-06, 3.4968154144826613e-06,-1.9195078838738656e-06)
最急降下法 2 回目の計算	
停留点	(6.852456938678921e-06,1.3746153980838749e-05, -5.21906449553119e-06,5.933415077690986e-06, 6.269711830222009e-06,-5.497440735121528e-06, -5.61980440616482e-06,-9.117347327607416e-06, -4.799813000247715e-06,1.4022639835313279e-06)
最急降下法 3 回目の計算	
停留点	(0.00026330463001817816,-1.1651401432270404e-05, -6.880775890703885e-05,-5.054949594291168e-05, -6.174312113345679e-07,-0.0001469110435626079, 0.00013824139992160844,6.023459369357024e-05, 0.00011913467270304648,-0.0002440814251455534)
最急降下法 4 回目の計算	
停留点	(-6.978474520840145e-07,-1.5003183193911026e-05, 4.202742158576482e-06,-1.9813840686355066e-07, -1.709652891863741e-06,8.576817316147291e-06, 3.5194979944483134e-06,1.3405230062843963e-05, 8.985372345224248e-06,-1.0309238004259107e-05)
最急降下法 5 回目の計算	
停留点	(0.0009394798556473681,0.00017008782696242548, 0.0004051003065778582,0.0004068378280492621, -0.0002564781608999018,-0.000990795003117159, -0.00021679228814970065,-2.5285360231767483e-05, 0.00033226838931198893,-0.0005395496928007351)
$n = 10$ の時の最急降下法の k の平均	
36469.4	
$n = 10$ の時の最急降下法の平均時間	
2.2231071472167967	

表 14: $n = 10$ の時のニュートン法の結果. 上から順に 1 回目から 5 回目までの計算で得られた停留点 (最適解) と最適値 の値を記している. 下の二つの項目は収束までに要した反復回数 k の平均とかかった時間の平均である.

ニュートン法 1 回目の計算	
停留点	(-0.0001875790738563613,-0.00017046658802140054, 0.00021509643295454154,5.009047649334759e-05, 2.7154080732929952e-05,0.0001425396696008707, 2.1037641785413925e-05,-7.222444176027892e-05, -3.663089656845523e-05,7.12083832789994e-05)
ニュートン法 2 回目の計算	
停留点	(4.557059633168072e-05,0.00012439128342115672, -0.0002023919978213001,-2.855117206430577e-05, 3.3836244957095724e-05,0.00010818599387807715, 7.302451617391325e-05,1.5432644583208848e-05, 4.026789622462496e-05,-0.00012450139069480356)
ニュートン法 3 回目の計算	
停留点	(-2.5667493298164107e-05,-1.2752757945604592e-05, 1.3561648586760311e-05,-3.384386627215746e-05, -0.00013838418635750 21,5.8803843675338114e-05, 1.6973375460136162e-05,-3.5364324315388607e-05, 0.00014648714519490942,1.854127083400572e-05)
ニュートン法 4 回目の計算	
停留点	(8.344286393794692e-08,-8.14093494198105e-08, -2.248012418913383e-06,4.349157470505605e-07, -1.0048406705295434e-06,1.9059702939032643e-06, 2.5578964142312485e-06,2.815774635294018e-06, -2.612560364132198e-06,-1.5543179289420738e-06)
ニュートン法 5 回目の計算	
停留点	(1.2184837362089875e-05,-1.3241581284977766e-05, 6.7740025112528895e-06,-8.854215055799571e-06, -8.640604844505834e-06,-4.559470362085478e-06, 7.309784189293368e-06,-1.2861642776226503e-05, 1.9010399840464963e-05,9.554556737625375e-06)
$n = 10$ の時のニュートン法の k の平均	
23.8	
$n = 10$ の時のニュートン法の平均時間	
0.0023061275482177735	

表 15: $n = 10$ の時の最急降下法の結果. 上から順に 1 回目から 5 回目まで の計算で得られた最適値の値を記している.

最適値	9.508348691686043e-12
最適値	8.695152491719303e-12
最適値	1.6785452580109055e-10
最適値	9.686431965056228e-12
最適値	6.118032210590004e-10

表 16: $n = 10$ の時のニュートン法の結果. 上から順に 1 回目から 5 回目までの計算で得られた最適値の値を記している.

最適値	1.688743369554442e-10
最適値	1.4718128573851426e-10
最適値	9.277278695580676e-11
最適値	1.731173745671279e-12
最適値	1.1701450848239743e-11

22　この課題の考察とまとめ

以上の結果より、最適化問題 (19.1) の最適解はおよそ 0、最適値もおよそ 0 であることがわかった。この結果に対して、以下理論的な考察を与える。
まず、任意のベクトル x に対し、$f = \frac{1}{2}\langle Ax, x\rangle \geq 0$ であること (行列 A が半正定値行列であること) を示す。
$A = Z^T Z$ であるから、

$$\begin{aligned} &\langle Ax, x\rangle \\ &= \langle Z^T Zx, x\rangle \\ &= \langle Zx, Zx\rangle = \|Zx\|^2 \geq 0 \end{aligned}$$

よって $f = \frac{1}{2}\langle Ax, x\rangle \geq 0$ である。
また、$x = 0$ は $f(x) = 0$ の自明な解である。以上より $x = 0$ が最適解であり、$f(x) = 0$ が最適値であることが示される。今回の実験結果は、この理論通りの結果が出ているため、実装したアルゴリズムは意図した通りの挙動をしていると考えられる。
最急降下法では n が大きいほど収束に要する反復回数が多くなっていく。ニュートン法においてもその傾向はあるが、今回は $n = 5$ より $n = 10$ の方が反復回数が少なかった。毎回の作業で行列 A が異なることから、これは不自然な結果ではないと判断した。
また、全ての n において、最急降下法よりもニュートン法の方が少ない反復回数で収束していることがわかった。これは課題 3 の部にて述べた理論の通りの結果であると言える。ニュートン法の解の収束にかかる時間は最急降下法の 0.04,0.1173,0.001 倍で、ニュートン法の方が短い時間で収束することもわかった。反復回数と収束までにかかった時間から一反復にかかった時間を導出し両手法を比較すると、ニュートン法が最急降下法の 3.75,2.73,1.58 倍で、いずれの場合もニュートン法で長い時間を要することがわかった。これは Hesse 行列の計算や修正にかかる計算コストの増分が原因であると考えられる。
なお、ニュートン法では計算量が大体 $O(n^3)$ となることが知られている [6]。今回は最大で $n = 10$ であったが、x ベクトルの次数がより大きくなれば一回の反復にかかる計算時間が爆発的に長くなり、最急降下法のそれとの差が顕著になるのではないかと推察する。

23　コード

コード 14: (19.1) を最急降下法およびニュートン法で解き、最適解と最適値を求めるコード

```
1 import numpy as np
```

```
2  import time
3
4  def f(x, A):#関数 f(x)を定義
5      return np.dot(np.dot(A, x), x) / 2.0
6
7  def grad(x, A):#f(x) の勾配ベクトルを返す
8      return np.dot(A, x)
9
10 def hesse(x, A):#f(x) の hesse 行列を返す
11     return A
12
13 def backtrack(x, d, A):#backtrack 法
14     #パラメータ
15     ita = 10.0 ** (-4)
16     rou = 0.5
17     t_kari = 1.0
18     #反復
19     while f(x + t_kari*d, A) > f(x, A) + ita * t_kari * np.sum(d*grad(x,
           A)):
20         t_kari = rou * t_kari
21     return t_kari
22
23 def norm(x): #ベクトルのノルムを計算
24     return np.sqrt(np.sum(x ** 2))
25
26 epsilon = 0.1 ** 6 #イプシロン
27 file = open('7.txt', 'w')
28 file_grad = open('7_grad.txt','w')
29
30 for n in ([2,5,10]):
31     k_newton = 0
32     k_gd = 0
33     t_list = []
34     data_list = ["\\begin{table}[ht]␣\n␣\centering␣\n␣\caption課題、
           最急降下法の [7$n=$",\
35         str(n),"の表]{","$n=",str(n),"の時の最急降下法の結果$上から順に
               回目から回目まで.15\
36 ␣␣␣␣␣␣␣␣␣␣␣␣の計算で得られた停留点最適解 ()と最適値の値を記している下
       の二つの項目は収束までに要した.\
37 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣反復回数の平均とかかった時間の平均である
       $k$.}\n␣\\begin{tabular}[ht]{|c|c|}␣\n␣\hline␣\n"]
38     if n == 10:
39         data_grad_list = ["\\begin{table}[ht]␣\n␣\centering␣\n␣\
               caption課題、最急降下法、[7$n=10の最適値の表
               $]{","$n=10の時の最急降下法の結果$上から順に回目から回目
               まで.15\
40 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣の計算で得られた最適値の値を記している
       .}\n␣\\begin{tabular}[ht]{|c|c|}␣\n␣\hline␣\n"]
41         file_grad.writelines(data_grad_list)
42     file.writelines(data_list)
43     #5 回繰り返す
44     for t_ in range(5):
45         x = np.array([1.0 for _ in range(n)])
46         Z = np.random.rand(n, n)# 一様分布から乱数生成
47         A = np.dot(np.transpose(Z), Z)# Z^T Z
48
49         t1 = time.time()
50         #最急降下法
51         while(norm(grad(x, A)) > epsilon):
52             d = -grad(x, A)
53             t = backtrack(x, d, A) #ステップサイズ
54             x += (t * d) #更新
55             k_gd += 1
```

```
56          t2 = time.time()
57          t_list.append(t2 − t1)
58
59          kari_hairetu = []
60          for l in range(n):
61              kari_hairetu.append(x[l])
62          file.write("\multicolumn{2}{|c最急降下法|}{")
63          file.write(str(t_+1))
64          file.write("回目の計算}␣\\\␣\n␣\hline␣\n")
65          file.write("停留点␣&␣\\begin{tabular}{c}")
66          file.write("(")
67          for i in range(len(kari_hairetu)):
68              if i!=len(kari_hairetu)−1:
69                  if i%2 == 0:
70                      file.writelines([str(kari_hairetu[i]),","])
71                  else:
72                      file.writelines([str(kari_hairetu[i]),",","\\\\"])
73              else:
74                  file.write(str(kari_hairetu[i]))
75          file.write(")␣\n")
76          file.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
77
78          if n != 10:
79              file.write("最適値␣&\\begin{tabular}{c}")
80              file.write(str(f(x, A)))
81              file.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
82
83          if n == 10:
84              file_grad.write("最適値␣&\\begin{tabular}{c}")
85              file_grad.write(str(f(x,A)))
86              file_grad.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
87      data_list = ["\multicolumn{2}{|c|}{$n=",str(n),"の時の最急降下法の
            の平均$$k$}␣\\\␣\n␣\hline␣\n" ,\
88          "\multicolumn{2}{|c|}{",str(k_gd / 5.0),"}\\\␣\n␣\hline␣\n"]
89      file.writelines(data_list)
90      data_list = ["\multicolumn{2}{|c|}{$n=",str(n),"の時の最急降下法の
            平均時間$}␣\\\␣\n␣\hline␣\n",\
91          "\multicolumn{2}{|c|}{",str(np.mean(np.array(t_list))),"}\\\␣\
                n␣␣\hline␣\n␣\\end{tabular}␣␣\n␣\\end{table}␣\n"]
92      file.writelines(data_list)
93
94      if n==10:
95          data_grad_list = ["\\end{tabular}␣␣\n␣\\end{table}␣\n"]
96          file_grad.writelines(data_grad_list)
97
98      t_list = []
99      data_list = ["\\begin{table}[ht]␣\n␣\centering␣\n␣\caption課題、
            ニュートン法の [7$n=$",str(n),"の表]{",\
100         "$n=",str(n),"の時のニュートン法の結果$上から順に回目から回目ま
                での計算で得られた停留点最適解.15()と最適値\
101 ␣␣␣␣␣␣␣␣␣␣␣␣␣の値を記している下の二つの項目は収束までに要した反復回数
        .の平均とかかった時間の平均である$k$.\
102 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}\n␣\\begin{tabular}[ht]{|c|c|}␣\n␣\hline␣\n"]
103     if n == 10:
104         data_grad_list = ["\\begin{table}[ht]␣\n␣\centering␣\n␣\
                caption課題、ニュートン法、[7$n=10の最適値の表
                $]{","$n=10の時のニュートン法の結果$上から順に回目から回
                目まで.15\
105 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣の計算で得られた最適値の値を記している
        .}\n␣\\begin{tabular}[ht]{|c|c|}␣\n␣\hline␣\n"]
106         file_grad.writelines(data_grad_list)
107     file.writelines(data_list)
108     for t_ in range(5):#繰り返す
```

```
109         x = np.array([1.0 for _ in range(n)])
110         Z = np.random.rand(n, n)
111         A = np.dot(np.transpose(Z), Z)
112
113         t1 = time.time()
114         #修正ニュートン法
115         while(norm(grad(x, A)) > epsilon):
116             w,v = np.linalg.eig(hesse(x, A))
117             tau = abs(np.min(w)) + 10.0 ** (-2)
118             d = -np.dot(np.linalg.inv(hesse(x, A)+tau*np.eye(n)),grad(x
                    , A))
119             t = backtrack(x, d, A) #ステップサイズ
120             x += (t * d) #更新
121             k_newton += 1
122         t2 = time.time()
123         t_list.append(t2 - t1)
124
125         #ファイルに書き出す
126         kari_hairetu = []
127         for l in range(n):
128             kari_hairetu.append(x[l])
129         file.write("\multicolumn{2}{|c|ニュートン法|}{")
130         file.write(str(t_+1))
131         file.write("回目の計算}␣\\\␣\n␣\hline␣\n")
132         file.write("停留点␣&␣\\begin{tabular}{c}")
133         file.write("(")
134         for i in range(len(kari_hairetu)):
135             if i!=len(kari_hairetu)-1:
136                 if i%2 == 0:
137                     file.writelines([str(kari_hairetu[i]),","])
138                 else:
139                     file.writelines([str(kari_hairetu[i]),",","\\\\"])
140             else:
141                 file.write(str(kari_hairetu[i]))
142         file.write(")␣\n")
143         file.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
144
145         #n=10 の時は最適解は別で出力見づらいので ()
146         if n != 10:
147             file.write("最適値␣&\\begin{tabular}{c}")
148             file.write(str(f(x,A)))
149             file.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
150
151         #n=10 の勾配を出力
152         if n == 10:
153             file_grad.write("最適値␣&\\begin{tabular}{c}")
154             file_grad.write(str(f(x, A)))
155             file_grad.write("\end{tabular}␣\\\␣\n␣\hline␣\n")
156
157     data_list = ["\multicolumn{2}{|c|}{$n=",str(n),"の時のニュートン法
            のの平均$$k$}\
158 ␣␣␣␣␣␣␣␣\\\␣\n␣\hline␣\n" ,"\multicolumn{2}{|c|}{",str(k_newton /
        5.0),"}\\\␣\n␣\hline␣\n"]
159     file.writelines(data_list)
160     data_list = ["\multicolumn{2}{|c|}{$n=",str(n),"の時のニュートン法
            の平均時間$}␣\\\␣\n\
161 ␣␣␣␣␣␣␣␣\hline␣\n","\multicolumn{2}{|c|}{",str(np.mean(np.array(
        t_list))),"}\\\␣\n␣␣\hline\
162 ␣␣␣␣␣␣␣␣\n␣\\end{tabular}␣␣\n␣\\end{table}␣\n"]
163     file.writelines(data_list)
164
165     if n==10:
166         data_grad_list = ["\\end{tabular}␣␣\n␣\\end{table}␣\n"]
```

```
167             file_grad.writelines(data_grad_list)
168
169
170 file.close()
171 file_grad.close()
```

第 IX 部
レポートのまとめ

本レポートでは、与えられた関数の零点を求める手法としてニュートン法と二分法を学び、そのアルゴリズムを実装した (課題 2)。結果、ニュートン法は二分法と比べて収束が早いことが数値的に確認できた。さらに、関数の停留点を求める手法として最急降下法とニュートン法を学び、一次元 (課題 3)、二次元 (課題 5) の変数 x で表される関数に関する最適化問題を両手法で解いた。結果、ニュートン法の方が最急降下法と比べて少ない反復回数で収束することがわかった。また、Hesse 行列の計算に必要な手続きが多いと、ニュートン法による計算にかかる時間が長くなることがわかった。最後には総合演習として二次元 (課題 6)、多次元 (課題 7) の変数 x で表される関数に関する最適化問題を最急降下法とニュートン法で解いた。ここでは、更新の方向に修正を加えることでより広い範囲の問題への対応を可能にしたニュートン法を用いた。結果、課題 5 などと同様にニュートン法の方が圧倒的に速く収束することや、x の次数が大きくなるとニュートン法における一回の反復には時間がかかる傾向にあることが数値的に確認できた。また、特に最急降下法においては x の次数が大きいほど収束までに要する反復回数が多くなることがわかった。

参考文献

[1] 講義資料

[2] 「二分法とは?アルゴリズム、収束、例題」https://risalc.info/src/bisection-method.html

[3] 「ニュートン法の証明」http://www.swlab.cs.okayama-u.ac.jp/~gotoh/lect/p1/c_prog/c3/newton_proof.pdf

[4] 「システム制御最適化特論 (平田 健太郎)」31 ページ http://imclab.sys.okayama-u.ac.jp/~kent/DIR/opt08.pdf

[5] 「ガウス・ニュートン法とレーベンバーグ・マーカート法 (ほげ)」https://sterngerlach.github.io/doc/gauss-newton.pdf

[6] 「降下法」http://www-optima.amp.i.kyoto-u.ac.jp/~nobuo/Ryukoku/2002/course7.pdf