数理工学実験
連続最適化

滝本亘（学籍番号 1029-33-1175）

2022 年 12 月 19 日

1　課題 1

1.1　はじめに

次の関数 $f : \mathbb{R} \to \mathbb{R}$ を考える。

$$f(x) := x^3 + 2x^2 - 5x - 6 \tag{1.1}$$

Python を用いて、以下の課題に取り組め。

(a) 関数 f を $-10 \leq x \leq 10$,$-10 \leq y \leq 10$ の範囲で描写せよ。

(b) 二分法により関数 f の全ての零点を求めよ。ただし、二つの異なる初期点は、(a) で描写した関数 f の概形を参考にして適切に設定せよ。

(c) ニュートン法により関数 f の全ての零点を求めよ。ただし、(a) で描写した関数 f の概形を参考にして、零点の近くに初期点を設定せよ。

1.2　原理・方法の詳細

1.2.1　二分法

Step 0: $f(a) < 0$, $f(b) > 0$ を満たす初期点 a,b を選び、終了条件 $\varepsilon > 0$ を決める。

Step 1: a と b の中間点 $c := (a + b)/2$ を求める。もし、c が終了条件 $|f(c)| \leq \varepsilon$ を満たしていれば、c を解として終了する。

Step 2: もし、$f(c) < 0$ であれば $a \leftarrow b$ とし、$f(c) \geq 0$ であれば $b \leftarrow c$ として、Step 1 へ戻る。

二分法は、関数 f が $\mathbb{R}$ 上で連続であれば必ず収束する。実際、Step 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 の間に存在するので、a と b の間隔を繰り返し $1/2$ に縮小していけば、中間点 c は関数 f の零点へ近づいていく。

1.2.2　ニュートン法

Step 0: 初期点 x_0 を選び、終了条件 $\epsilon > 0$ を決める。$k := 0$ とする。

Step 1: ニュートン方程式

$$f'(x_k)\Delta x_k = -f(x_k) \tag{1.2}$$

を解き、Δx_k を求める。

Step 2: 点列を $x_{k+1} := x_k + \Delta x_k$ により更新する。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。

Step 3: $k \leftarrow k + 1$ として、Step 1 へ戻る。

ニュートン法は、点 x_k の周りで関数 f を一次近似し、接線の方程式 $y = f'(x_k)(x - x_k) + f(x_k)$ を考える。この方程式と x 軸の交点を求め、更新点とする（つまり、$f'(x_k)(x - x_k) + f(x_k) = 0$ を解き、$x = x_k - f(x_k)/f'(x_K)$ を更新点とする）ことで点列を生成する反復法である。一般的に収束は速いが、解の近くに初期点を選ばなければ収束しないということが知られている。

1.3 実験方法

二分法およびニュートン法の終了条件を

$$\epsilon = 10^{-6} \tag{1.3}$$

とする。

(a)

コード 1: 課題 1(a)

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(−10,10,0.1)
y = x**3+2*x**2−5*x−6
plt.plot(x, y)
plt.xlim(−10,10)
plt.ylim(−10,10)
plt.axhline(0, c='black')
plt.axvline(0, c='black')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

(b)

コード 2: 課題 1(b)

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

def nibun(i):
    e=10**−6
    a=float(input("a="))
    b=float(input("b="))

    if f(a)<0 and f(b)>=0:
        pass
    elif f(a)>=0 and f(b)<0:
        a,b=b,a
    else:
        print('初期値を再入力してください。')
        return nibun(i)

    c=(a+b)/2
    while abs(f(c))>e:
        if f(c)<0:
            a=c
        elif f(c)>=0:
            b=c
        c=(a+b)/2

    else:
        print('x={}\n'.format(c))

for i in range(3):
    nibun(i)
```

(c)

コード 3: 課題 1(c)

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

def df(x):
    return 3*x**2+4*x-5

def nibun(i):
    e=10**-6
    a=float(input("x0="))

    while abs(f(a))>e:
        a-=f(a)/df(a)

    else:
        print('x={}\n'.format(a))

for i in range(3):
    nibun(i)
```

1.4 結果

(a)

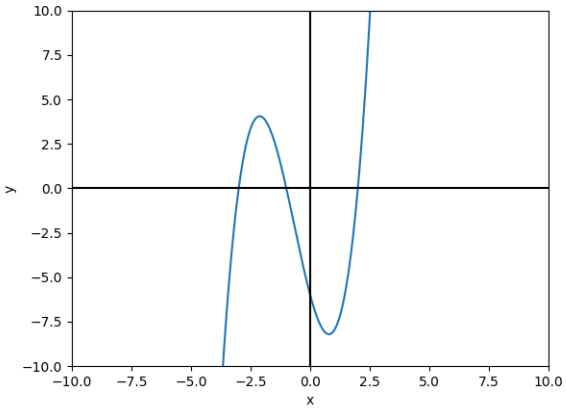

図 1: $f(x) = x^3 + 2x^2 - 5x - 6$

(b) (a) のグラフより、零点は 3 つ存在する。
初期点は、$(a_1, b_1) = (-5.0, -2.5)$, $(a_2, b_2) = (-2.5, 0.0)$, $(a_3, b_3) = (0.0, 2.5)$ とする。

コード 4: 課題 1(b) 結果

```
a=-5.0
b=-2.5
x=-2.9999999701976776

a=-2.5
b=0.0
x=-0.9999999403953552

a=0.0
b=2.5
x=2.0000000298023224
```

(c) 初期点は $x_0 = -2.5, 0.0, 2.5$ とする。

コード 5: 課題 1(c) 結果

```
1  x0=−2.5
2  x=−3.0
3
4  x0=0
5  x=−1.0
6
7  x0=2.5
8  x=2.0
```

1.5 考察

(b)(c)

$$f(x) = 0 \tag{1.4}$$

とすると、

$$x^3 + 2x^2 - 5x - 6 = 0 \tag{1.5}$$

$$(x+3)(x+1)(x-2) = 0 \tag{1.6}$$

$$x = -3, -1, 2 \tag{1.7}$$

となる。したがって、結果は正しい。

2　課題 2

2.1　はじめに

関数 $f : \mathbb{R} \to \mathbb{R}$ を以下で定義する。

$$f(x) := \frac{1}{3}x^3 - x^2 - 3x + \frac{5}{3} \tag{2.1}$$

Python を用いて

(a) 最急降下法を実装し、関数 f の停留点を求めよ。ただし、初期点は $x_0 := 1/2$ と設定し、各反復 $k \in \mathbb{N} \cup 0$ において、ステップサイズは $t_k := 1/(k+1)$ を採用せよ。

(b) ニュートン法を実装し、関数 f の停留点を求めよ。ただし、初期点は $x_0 := 5$ と設定し、各反復 $k \in \mathbb{N} \cup 0$ において、ステップサイズは $t_k := 1$ を採用せよ。

2.2　原理・方法の詳細

2.2.1　降下法

最急降下法やニュートン法は降下法と呼ばれる手法である。降下法は、関数の値を減少させる点列 $x^0, x^1, x^2, \cdots$ を生成していく反復法である。つまり、

$$f(x^0) > f(x^1) > \cdots \tag{2.2}$$

となる点列 x_k を生成する。この点列は

$$x^{k+1} := x^k + t_k d^k \tag{2.3}$$

で与えられる。ここで、d_k を探索方向、t_k をステップサイズと呼ぶ。降下法では、前提として、d^k は次の条件を満たすと仮定する。

$$\langle d^k, \nabla f(x^k) \rangle < 0 \tag{2.4}$$

ここで、$\langle y, z \rangle$ はベクトル $y, z \in \mathbb{R}^n$ の内積を表している。つまり、$\langle y, z \rangle := \sum_{i=0}^{n-1} |y|_i |z|_i$ である。このような条件を満たしているベクトル d^k のことを降下方向と呼ぶ。

降下法を以下に与える。ただし、$\|z\| := (\sum_{i=0}^{n-1} [z]_i^2)^{1/2}$ である。

Step 0: 初期点 x^0、終了条件 $\epsilon > 0$ を決める。$k := 0$ とする。

Step 1: 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。

Step 2: 次式を満たす降下方向 d^k を定める。

$$\langle d^k, \nabla f(x^k) \rangle < 0 \tag{2.5}$$

Step 3: 点 x^k を $x^{k+1} := x^k + t_k d^k$ と更新する。$k \leftarrow k + 1$ として、Step 1 へ戻る。

この降下法によって生成される点列 x^k に対して、以下で与える収束に関する定理が示されている。

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する。また、ある正の定数 γ が存在し、全ての $k \in \mathbb{N} \cup 0$ に対して、以下の不等式が成り立つと仮定する。

$$\langle d^k, \nabla f(x^k) \rangle \leq -\gamma \|\nabla f(x^k)\|^2 \tag{2.6}$$

このとき、点列 x^k の任意の集積点 x^* は、関数 f の停留点となる。つまり、$\nabla f(x^*) = 0$ が成り立つ。

この定理により、上手に降下方向を選んでやれば、問題の停留点（多くの場合、局所最小解）を求めることができる。

2.2.2 最急降下法

最急降下法は、点 x^k が与えられたときに、降下方向として以下を用いる手法である。

$$\textbf{[最急降下方向]} \qquad d^k := -\nabla f(x^k) \tag{2.7}$$

いま、$\nabla f(x^k) \neq 0$ と仮定する。（もし、$\nabla f(x^k) = 0$ であれば、点 x^k は既に f の停留点となっている。）このとき、

$$\langle d^k, \nabla f(x^k) \rangle = -\|\nabla f(x^k)\|^2 < 0 \tag{2.8}$$

が成り立つため、d^k は降下方向であり、定理 1 の仮定を満たすことがわかる。（実際、上記の等号により、定理 1 における正の定数 γ として、$\gamma = 1$ の存在が確かめられる。）

2.2.3 ニュートン法

最急降下法は、点 x^k が与えられたときに、降下方向として以下を用いる手法である。

$$\textbf{[ニュートン方向]} \qquad d^k := -\nabla^2 f(x^k)^{-1} \nabla f(x^k) \tag{2.9}$$

ただし、$\nabla^2 f(x)$ は x における f のヘッセ行列を表しており、

$$\begin{bmatrix} \frac{\partial^2 f(x)}{\partial [x]_0^2} & \frac{\partial^2 f(x)}{\partial [x]_0 [x]_1} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_0 [x]_{n-1}} \\ \frac{\partial^2 f(x)}{\partial [x]_1 [x]_0} & \frac{\partial^2 f(x)}{\partial [x]_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_1 [x]_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial [x]_{n-1} [x]_0} & \frac{\partial^2 f(x)}{\partial [x]_{n-1} [x]_1} & \cdots & \frac{\partial^2 f(x)}{\partial [x]_{n-1}^2} \end{bmatrix} \tag{2.10}$$

で与えられる。ここで、$\nabla^2 f(x^k)$ は有界かつ一様正定値であると仮定する。これは、ある $\mu > 0$ が存在して、全ての $k \in \mathbb{N} \cup 0$ と $v \in \mathbb{R}^n$ に対して、

$$\frac{1}{\mu} \leq \langle \nabla^2 f(x^k)^{-1} v, v \rangle \leq \mu \|v\|^2 \tag{2.11}$$

が成り立つことと同値である。このとき、すべての k に対して、$\nabla f(x^k) \neq 0$ であれば、

$$\begin{aligned} \langle d^k, \nabla f(x^k) \rangle &= -\langle \nabla^2 f(x^k)^{-1} \nabla f(x^k), \nabla f(x^k) \rangle \\ &\leq -\frac{1}{\mu} \|\nabla f(x^k)\|^2 \\ &< 0 \end{aligned} \tag{2.12}$$

となるため、d^k は降下方向であり、定理 1 の仮定を満たすことがわかる。

2.3 実験方法

勾配ベクトルとヘッセ行列は

$$\nabla f(x) = x^2 - 2x - 3 \tag{2.13}$$

$$\nabla^2 f(x) = 2x - 2 \tag{2.14}$$

である。

また、最急降下法およびニュートン法の終了条件は

$$\epsilon = 10^{-6} \tag{2.15}$$

とする。反復回数が

$$k = 10^7 \tag{2.16}$$

に達すると、k は十分大きいと判断し、解は求まらないとする。

(a)

コード 6: 課題 2(a)

```
def f(x):
    return 1/3*x**3-x**2-3*x+5/3

def g(x):
    return x**2-2*x-3

def d(x):
    return -g(x)

def t(k):
    return 1/(k+1)

x=1/2
e=10**-6
k=0

while abs(g(x))>e:
    x+=t(k)*d(x)
    k+=1

    if k==10**7:
        print('停留点は存在しない')
        break

    else:
        continue

else:
    print('停留点は x={}'.format(x))
```

(b)

コード 7: 課題 2(b)

```
def f(x):
    return 1/3*x**3-x**2-3*x+5/3

def g(x):
```

```
5       return x**2-2*x-3
6
7   def h(x):
8       return 2*x-2
9
10  def d(x):
11      return -h(x)**-1*g(x)
12
13  def t(k):
14      return 1/(k+1)
15
16  x=5
17  e=10**-6
18  k=0
19
20  while abs(g(x))>e:
21      x+=t(k)*d(x)
22      k+=1
23
24      if k==10**7:
25          print('停留点は存在しない')
26          break
27
28      else:
29          continue
30
31  else:
32      print('停留点は x={}'.format(x))
```

2.4 結果

(a) 停留点は x=2.999999750304564

(b) 停留点は x=3.0000002499999647

2.5 考察

(a)(b)

$$f'(x) = 0 \tag{2.17}$$

とすると、

$$x^2 - 2x - 3 = 0 \tag{2.18}$$

$$(x+1)(x-3) = 0 \tag{2.19}$$

$$x = -1, 3 \tag{2.20}$$

となる。また、下のグラフより $x = -1$ は局所最大解、$x = 3$ は局所最小解になっている。したがって、結果は正しい。

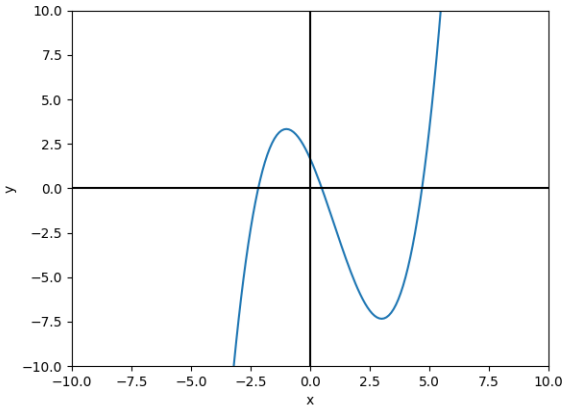

図 2: $f(x) = \frac{1}{3}x^3 - x^2 - 3x + \frac{5}{3}$

3 課題 3

3.1 はじめに

次の 2 次元の最適化問題を考える。

$$minimize \qquad f(x) := x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3 \tag{3.1}$$

$$subject \quad to \qquad x := (x_0, x_1)^T \in \mathbb{R}^2 \tag{3.2}$$

Python を用いて、

(a) 点 x が与えられたとき、目的関数の値 $f(x)$ を出力する関数を作成せよ。

(b) 点 x が与えられたとき、勾配ベクトル $\nabla f(x)$ を出力する関数を作成せよ。

(c) 点 x が与えられたとき、ヘッセ行列 $\nabla^2 f(x)$ を出力する関数を作成せよ。

3.2 実験方法

勾配ベクトルとヘッセ行列は

$$\nabla f(x) = \begin{pmatrix} 2x_0 + e^{x_0} - 2x_1 \\ 4x_1^3 + 2x_1 - 2x_0 \end{pmatrix} \tag{3.3}$$

$$\nabla^2 f(x) = 2\begin{pmatrix} 2 + e^{x_0} & -2 \\ -2 & 12x_1^2 + 2 \end{pmatrix} \tag{3.4}$$

である。

(a)

コード 8: 課題 3(a)

```
1  import math
2
3  def evalf(x):
4      f=x[0]**2+math.exp(x[0])+x[1]**4+x[1]**2-2*x[0]*x[1]+3
5      print('f={}'.format(f))
```

(b)

コード 9: 課題 3(b)

```
1  import math
2  import numpy as np
3
4  def evalg(x):
5      g=np.array([2*x[0]+math.exp(x[0])-2*x[1],4*x[1]**3+2*x[1]-2*x[0]])
6      print('f={}'.format(g))
```

(c)

コード 10: 課題 3(c)

```
1  import math
2  import numpy as np
3
4  def evalh(x):
5      H=np.array([[2+math.exp(x[0]),-2], [-2,12*x[1]**2+2]])
6      print('H=\n{}'.format(H))
```

4　課題 4

4.1　はじめに

(a) バックトラック法を用いた最急降下法を実装することで課題 3 の最適化問題を解き、最適解、最適値、あよび反復回数を与えよ。ただし、初期点は $x^0 = (1,1)^T$ と設定し、またバックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定せよ。

(b) バックトラック法を用いたニュートン法を実装することで課題 3 の最適化問題を解き、最適解、最適値、および反復回数を与えよ。ただし、初期点は $x^0 = (1,1)^T$ と設定し、またバックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定せよ。

4.2　原理・方法の詳細

4.2.1　直線探索法

直線探索法は、ステップサイズ $t_k > 0$ を決定するための手法であり、点 x_k を x_{k+1} へ更新する際に、$f(x_{k+1}) < f(x_k)$ を満たすようにするために行う。これまでにいくつかの直線探索法が提案されているが、以下に与えるバックトラック法は理論的にも実用的にも優れていることが知られている。

Step 0: $\xi \in (0,1)$, $\rho \in (0,1)$, 初期ステップサイズ $\bar{t} > 0$ を選ぶ。$t = \bar{t}$ とする。

Step 1: 次式を満たすまで $t \leftarrow \rho t$ をする。

$$f(x^k + td^k) \leq f(x^k) + \xi t \langle d^k, \nabla f(x^k) \rangle \tag{4.1}$$

Step 2: $t_k = t$ として終了。

Step 1 の不等式はアルミホ条件と呼ばれる。ステップサイズ t に対して、$\varphi := f(x^k + td^k)$ とする。原点における φ の接線は $y = f(x^k) + t\langle d^k, \nabla f(x^k) \rangle$ である。アルミホ条件を満たすステップサイズを選ぶことは、φ の接線の傾きを緩和して得られる直線 $y = f(x^k) + \xi t \langle d^k, \nabla f(x^k) \rangle$ よりも関数値が小さくなる t の区間からステップサイズを選ぶことに対応していると言える。ξ を小さく設定すれば、アルミホ条件を満たすステップサイズを速く見つけることができる。

4.3　実験方法

また、最急降下法およびニュートン法の終了条件は

$$\epsilon = 10^{-6} \tag{4.2}$$

とする。反復回数が

$$k = 10^7 \tag{4.3}$$

に達すると、k は十分大きいと判断し、解は求まらないとする。

(a)

コード 11: 課題 4(a)

```
import math
import numpy as np

def evalf(x):
    f=x[0]**2+math.exp(x[0])+x[1]**4+x[1]**2-2*x[0]*x[1]+3
    return f

def evalg(x):
    g=np.array([2*x[0]+math.exp(x[0])-2*x[1],4*x[1]**3+2*x[1]-2*x[0]])
    return g

def d(x):
    d=-evalg(x)
    return d

def t(x):
    t=1
    xi=10**-4
    rho=0.5
    while evalf(x+t*d(x))>evalf(x)+xi*t*np.inner(d(x),evalg(x)):
        t=rho*t

    else:
        return t

x=np.array([1,1])
e=10**-6
k=0

while np.linalg.norm(evalg(x),ord=2)>e:
    x=x+t(x)*d(x)
    k+=1

    if k==10**7:
        print('最適解は存在しない')
        break

    else:
        continue

else:
    print('最適解␣x={}\最適値 n␣f(x)={}\反復回数 n␣回{}'.format(x,evalf(x),k))
```

(b)

コード 12: 課題 4(b)

```
import math
import numpy as np

def evalf(x):
    f=x[0]**2+math.exp(x[0])+x[1]**4+x[1]**2-2*x[0]*x[1]+3
    return f

def evalg(x):
    g=np.array([2*x[0]+math.exp(x[0])-2*x[1],4*x[1]**3+2*x[1]-2*x[0]])
    return g

def evalh(x):
    h=np.array([[2+math.exp(x[0]),-2], [-2,12*x[1]**2+2]])
    return h

def d(x):
    d=-np.dot(np.linalg.inv(evalh(x)),evalg(x))
    return d

```

```
20  def t(x):
21      t=1
22      xi=10**-4
23      rho=0.5
24      while evalf(x+t*d(x))>evalf(x)+xi*t*np.inner(d(x),evalg(x)):
25          t=rho*t
26
27      else:
28          return t
29
30  x=np.array([1,1])
31  e=10**-6
32  k=0
33
34  while np.linalg.norm(evalg(x),ord=2)>e:
35      x=x+t(x)*d(x)
36      k+=1
37
38      if k==10**7:
39          print('最適解は存在しない')
40          break
41
42      else:
43          continue
44
45  else:
46      print('最適解␣x={}\最適値 n␣f(x)={}\反復回数 n␣回{}'.format(x,evalf(x),k))
```

4.4 結果

(a) 最適解 x=[-0.73345152 -0.49332751]
最適値 f(x)=3.597138024959682
反復回数 31 回

(b) 最適解 x=[-0.73345172 -0.4933275]
最適値 f(x)=3.597138024959629
反復回数 6 回

4.5 考察

x^* を求める問題の解とし、x_k は x^* へ収束するとする。このとき、ある $\kappa \in (0,)$ と $m_0 \in \mathbb{N}$ が存在して、

$$\|x^{k+1} - x^*\| \leq \kappa \|x^k - x^* \tag{4.4}$$

が成り立つとき、点列 x^k は解 x^* に 1 次収束するという。一方、

$$\lim_{k \in \infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|^2} < \infty \tag{4.5}$$

が成り立つとき、点列 x^k は解 x^* に 2 次収束するという。これは、ある $\eta > 0$ と $n_0 \in \mathbb{N}$ が存在して、

$$\|x^{k+1} - x^*\| \leq \eta \|x^k - x^*\|^2 \tag{4.6}$$

となることを意味しており、一旦 $\|x^k - x^*\| < 1$ となると、非常に速く収束する。

ここで、次の点列 y_k と z_k を考える。

$$y_k = 0.5^k \quad z_k = 0.5z_{k-1}^2 \tag{4.7}$$

ただし、$z_0 = 1$ とする。点列 y_k は 0 に 1 次収束することが確かめられる。一方、点列 z_k は、0 に 2 次収束することが確かめられる。これらの点列を計算してみると、表 1 となる。表 1 により、2 次収束する点列は、1 次収束する点列に比べて、極めて速い収束性を持つことがわかる。

表 1: 1 次収束と 2 次収束

k	y_k	z_k
0	1	1
1	0.5	0.5
2	0.25	0.125
3	0.125	0.0078125
4	0.0625	0.0000305
⋮	⋮	⋮

最急降下法は適切な仮定の下で 1 次収束することが示されている。しかし、最小化する関数の一階微分（つまり一次の情報）しか用いていないため、2 次収束しないことが多い。一方、ニュートン法は関数のヘッセ行列（つまり二次の情報）を用いた方法であり、適切な仮定の下で 2 次収束することが知られている。よって、最急降下法とニュートン法の解への収束性を比較すると、一般にニュートン法に軍配が上がる。

実際、最急降下法の反復回数は $k = 31$、ニュートン法の反復回数は $k = 6$ である。

5　課題 5

5.1　はじめに

Python を用いて、最急降下法およびニュートン法を実装することで次の最適化問題を解き、最適解および各手法の反復回数を与えよ。ただし、初期点は $x^0 = [2, 0]^T$ と設定し、またバックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定せよ。

$$minimize \qquad f(x) := \sum_{i=0}^{2} f_i(x)^2 \tag{5.1}$$

$$subject \quad to \qquad x \in \mathbb{R}^2 \tag{5.2}$$

ただし、$f_i(x) := y_i - [x]_0(1 - [x]_1^{i+1})(i = 0, 1, 2)$ と定義し、$y_0 = 1.5$, $y_1 = 2.25$, $y_2 = 2.625$ である。

5.2　実験方法

$f_i(x)$ の勾配ベクトルとヘッセ行列は

$$\nabla f_i(x) = \begin{pmatrix} -(1 - x_1^{i+1}) \\ (i+1)x_0 x_1^i \end{pmatrix} \tag{5.3}$$

$$\nabla^2 f_i(x) = \begin{pmatrix} 0 & (i+1)x_1^i \\ (i+1)x_1^i & i(i+1)x_0 x_1^{i-1} \end{pmatrix} \tag{5.4}$$

となる。また、$f(x)$ の勾配ベクトルとヘッセ行列は

$$\nabla f(x) = 2\sum_{i=0}^{2} f_i(x)\nabla f_i(x) \tag{5.5}$$

$$\nabla^2 f(x) = 2\sum_{i=0}^{2} (f_i(x)\nabla^2 f_i(x) + \nabla f_i(x)\nabla f_i(x)^T) \tag{5.6}$$

となる。

最急降下法

コード 13: 課題 5 最急降下法

```
1   import numpy as np
2
3   def evalfi(x,i):
4       fi=y[i]−x[0]∗(1−x[1]∗∗(i+1))
5       return fi
6
7   def evalf(x):
8       f=0
9       for i in range(3):
10          f+=evalfi(x,i)∗∗2
11      return f
12
13  def evalgi(x,i):
14      gi=np.array([−(1−x[1]∗∗(i+1)),(i+1)∗x[0]∗x[1]∗∗i])
15      return gi
```

```
16
17  def evalg(x):
18      g=0
19      for i in range(3):
20          g+=2*evalfi(x,i)*evalgi(x,i)
21      return g
22
23  def d(x):
24      d=-evalg(x)
25      return d
26
27  def t(x):
28      t=1
29      xi=10**-4
30      rho=0.5
31      while evalf(x+t*d(x))>evalf(x)+xi*t*np.inner(d(x),evalg(x)):
32          t*=rho
33
34      else:
35          return t
36
37  x=np.array([2.0,0.0])
38  e=10**-6
39  k=0
40  y=[1.5,2.25,2.625]
41
42  while np.linalg.norm(evalg(x),ord=2)>e:
43      x+=t(x)*d(x)
44      k+=1
45
46      if k==10**7:
47          print('最適解は存在しない')
48          break
49
50      else:
51          continue
52
53  else:
54      print('最適解␣x={}\最適値␣f(x)={}\反復回数␣回{}'.format(x,evalf(x),k))
```

ニュートン法

コード 14: 課題 5 ニュートン法

```
1   import numpy as np
2
3   def evalfi(x,i):
4       fi=y[i]-x[0]*(1-x[1]**(i+1))
5       return fi
6
7   def evalf(x):
8       f=0
9       for i in range(3):
10          f+=evalfi(x,i)**2
11      return f
12
13  def evalgi(x,i):
14      gi=np.array([-(1-x[1]**(i+1)),(i+1)*x[0]*x[1]**i])
15      return gi
16
17  def evalg(x):
18      g=0
19      for i in range(3):
20          g+=2*evalfi(x,i)*evalgi(x,i)
21      return g
22
23
```

```
24  def evalhi(x,i):
25      if i==0:
26          hi=np.array([[0,1],[1,0]])
27      else:
28          hi=np.array([[0,(i+1)*x[1]**i],[(i+1)*x[1]**i,(i+1)*i*x[0]*x[1]**(i-1)]])
29      return hi
30
31  def evalh(x):
32      h=0
33      for i in range(3):
34          h+=2*(evalfi(x,i)*evalhi(x,i)+np.dot(np.reshape(evalgi(x,i),(2,1)),np.reshape(evalgi(x,
                i),(1,2))))
35      return h
36
37  def d(x):
38      d=-np.dot(np.linalg.inv(evalh(x)),evalg(x))
39      return d
40
41  def t(x):
42      t=1
43      xi=10**-4
44      rho=0.5
45      while evalf(x+t*d(x))>evalf(x)+xi*t*np.inner(d(x),evalg(x)):
46          t=rho*t
47
48      else:
49          return t
50
51  x=np.array([2.0,0.0])
52  e=10**-6
53  k=0
54  y=[1.5,2.25,2.625]
55
56  while np.linalg.norm(evalg(x),ord=2)>e:
57      x+=t(x)*d(x)
58      k+=1
59
60      if k==10**7:
61          print('最適解は存在しない')
62          break
63
64      else:
65          continue
66
67  else:
68      print('最適解␣x={}\最適値␣f(x)={}\反復回数␣回{}'.format(x,evalf(x),k))
```

5.3 結果

最急降下法

表 2: 最急降下法

ϵ	最適解	最適値	反復回数
10^{-1}	[2.80535943 0.44598633]	0.00763758023454063	16
10^{-2}	[2.97627274 0.49387788]	9.32552667632546e-05	124
10^{-3}	[2.99755823 0.49937815]	9.626243421991083e-07	284
10^{-4}	[2.99976898 0.4999412]	8.596965630726712e-09	454
10^{-5}	[2.99997612 0.49999391]	9.190947400042634e-11	618
10^{-6}	[2.99999781 0.49999946]	7.676324891043213e-13	791
10^{-7}	[2.99999977 0.49999994]	8.690545545880527e-15	953
10^{-8}	[2.99999998 0.49999999]	9.838868012705024e-17	1115
10^{-9}	[3. 0.5]	8.218677915652985e-19	1288
10^{-10}	[3. 0.5]	9.304686870440796e-21	1450

ニュートン法

表 3: ニュートン法

ϵ	最適解	最適値	反復回数
10^{-1}	[2.99973996 0.49991123]	2.442141693672833e-08	4
10^{-2}	[2.99973996 0.49991123]	2.442141693672833e-08	4
10^{-3}	[3.00000013 0.50000005]	9.470356069887723e-15	5
10^{-4}	[3.00000013 0.50000005]	9.470356069887723e-15	5
10^{-5}	[3.00000013 0.50000005]	9.470356069887723e-15	5
10^{-6}	[3.00000013 0.50000005]	9.470356069887723e-15	5
10^{-7}	[3. 0.5]	1.3921915862953569e-27	6
10^{-8}	[3. 0.5]	1.3921915862953569e-27	6
10^{-9}	[3. 0.5]	1.3921915862953569e-27	6
10^{-10}	[3. 0.5]	1.3921915862953569e-27	6

5.4 考察

最急降下法およびニュートン法どちらにおいても、ϵ の値を小さくすればするほど、最適解は厳密解に近づいており、反復回数は多くなっている。

また、最急降下法に比べてニュートン法は、反復回数が極端に少ない。