連続最適化

目次

1 はじめに 2

2 課題 1 2
2.1 二分法 2
2.2 ニュートン法 2
2.3 1-a 3
2.4 1-b 3
2.5 1-c 3
2.6 結果の考察 4

3 課題 2 4
3.1 降下法 4
3.2 2-a 5
3.3 2-b 5
3.4 結果の考察 5

4 課題 3,4 6
4.1 バックトラック法 6
4.2 3-a,b,c 6
4.3 4-a 7
4.4 4-b 7
4.5 結果の考察 7

5 課題 5 8
5.1 準備 8
5.2 計算結果 8
5.3 結果の考察 9

6 付録 9

7 参考文献 17

1　はじめに

本レポートでは, 与えられた関数 $f : \mathbb{R}^n \to \mathbb{R}$ の零点や最大値, 最小値を求める最適化手法について述べる. なお, 実験で用いたプログラムは付録に記す.

2　課題 1

ここでは, 関数の零点を求める手法である二分法とニュートン法について述べる.
また, 本問では関数 $f : \mathbb{R} \to \mathbb{R}$ を以下のように与える.

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

2.1　二分法

以下に二分法のアルゴリズムを示す.

二分法のアルゴリズム

Step 0: $f(a) < 0, f(b) \geq 0$ を満たす初期点 a, b を選び, 終了条件 $\epsilon > 0$ 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 $|f(c)| \leq \epsilon$ 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 $a \leftarrow c$ とし, $f(c) \geq 0$ であれば $b \leftarrow c$ として, Step1 へ戻る.

二分法は, 二つの異なる初期点 a, b の間に零点が存在すれば, 必ず収束する. しかし, 一般的には収束が遅いことで知られる.

2.2　ニュートン法

以下にニュートン法のアルゴリズムを示す.

ニュートン法のアルゴリズム

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

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

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, Step1 へ戻る.

ニュートン法は, 点 x_k の周りで関数 f を一時近似し, 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, 更新点とすることで点列を生成する反復法である. 一般的に, 二分法よりも収束が速いが, 解の近くに収束点を選ばなければ収束しないことが知られている.

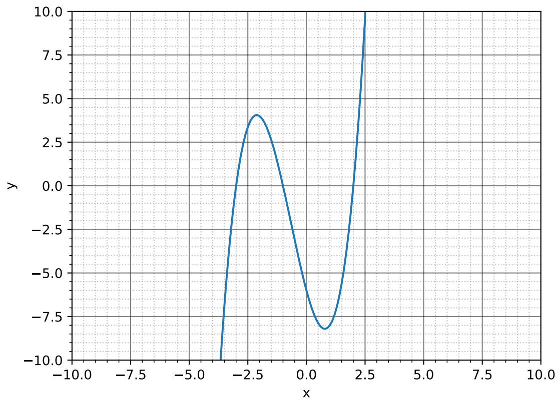

図 1: 関数 f のグラフ.

表 1: 二分法の結果

a	b	零点
-3.5	-2.5	-3.0
0	-1.5	-1.0001220703125
1	2.5	1.99993896484375

表 2: ニュートン法の結果

初期点	零点
-2.5	-3.0000104996724555
-1.5	-0.9999995152497579
2.5	2.0000104905873

2.3 1-a

関数 f を $-10 \leq x \leq 10, -10 \leq y \leq 10$ の範囲で描写する．グラフは図 1 に示す．

2.4 1-b

二分法により関数 f のすべての零点を求める．計算結果を表 1 に示す．ただし，初期点は (a) で描写した関数 f の概形を参考に，条件を満たすように選んだ．終了条件は $|f(x_{k+1})| \leq \epsilon = 10^{-6}$ とした．

2.5 1-c

ニュートン法により関数 f のすべての零点を求める．計算結果を表 2 に示す．ただし，初期点は (a) で描写した関数 f の概形を参考に，零点の近くで選んだ．終了条件は二分法と同じく $\epsilon = 10^{-6}$ とした．

2.6 結果の考察

式 (1) から求められる零点は $x = -3, -1, 2$ であり, 二分法, ニュートン法ともに零点に収束していることがわかる. 二分法で $a = -3.5,\ b = -2.5$ としたとき, 零点はちょうど -3 に等しくなっているが, これは初めに c を求める際に $c = (-3.5 - 2.5)/2 = -3$ となり, 零点と等しくなったためである.

3 課題 2

関数 $f : \mathbb{R} \to \mathbb{R}$ を

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

で定める. この関数 f の停留点を (a) 最急降下法および (b) ニュートン法で求める.

3.1 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. 降下法では,

$$f(x^0) > f(x^1) > \cdots$$

となる点列 x^k を生成する. この点列は

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

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.

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

ただし, $\langle y, z\rangle$ はベクトル y, z の内積を表す. このような条件を満たしているベクトル d^k を降下方向と呼ぶ. 最急降下法とニュートン法では, この降下方向の定め方が異なる. さらに, 点列が条件を満たすためには t_k を適切な値にする必要がある. この t_k を定める方法は直線探索法と呼ばれる. 降下法のアルゴリズムを以下にまとめる.

降下法のアルゴリズム

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 を求める.

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

Step 3: ステップサイズ t_k を求める.
Step 4: 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, Step1 へ戻る.

なお, 最急降下法では $d^k := -\nabla f(x^k)$, ニュートン法では $d^k := -\nabla^2 f(x^k)^{-1}\nabla f(x^k)$ と定める. ここで $\nabla^2 f(x)$ は点 x における f のヘッセ行列を表している. また, 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.

表 3: 最急降下法の結果

反復回数	停留点
62	2.999999750304564

表 4: ニュートン法の結果

反復回数	停留点
4	3.0000000929222947

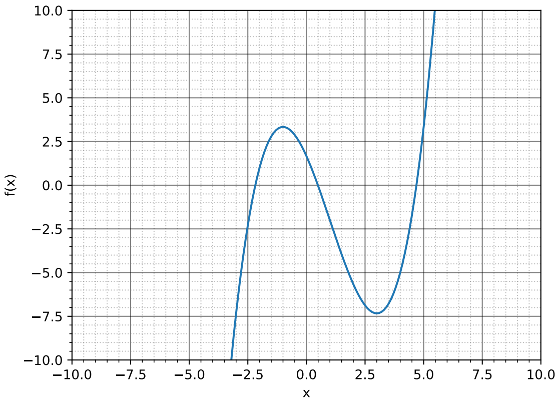

図 2: 関数 $f(x) = \frac{1}{3}x^3 - x^2 - 3x + \frac{5}{3}$ のグラフ.

3.2 2-a

最急降下法で (2) の停留点を求める．ただし, 初期点 $x_0 := 1/2$, ステップサイズ $t_k := 1/(k+1)$, 終了条件 $\epsilon = 10^{-6}$ とする．いま, $\nabla f(x) = f'(x) = x^2 - 2x3$ である．結果を表 3 に示す.

3.3 2-b

ニュートン法で (2) の停留点を求める．ただし, 初期点 $x_0 := 5$, ステップサイズ $t_k := 1$, 終了条件 $\epsilon = 10^{-6}$ とする．いま, $\nabla f(x) = f'(x) = x^2 - 2x - 3$, $\nabla^2 f(x) = f''(x) = 2x - 2$ である．結果を表 4 に示す.

3.4 結果の考察

両手法とも, $x = 3$ に収束している．一方で, 関数 (2) のグラフは図 2 のようになる．$f'(x) = x^2 - 2x - 3 = (x-1)(x+3)$ より, 確かに $x = 3$ が停留点であることがわかる.

4 課題 3,4

次の 2 次元の最適化問題を考える.

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

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で最適化問題を解く. ただし, 初期点は $x^0 = (1,1)^T$, バックトラック法における ξ, ρ, $\bar{t}$ は, それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ とする. 終了条件は $\epsilon = 2 * 10^{-6}$ とする.

4.1 バックトラック法

ここで, バックトラック法について述べる. バックトラック法は直線探索法の一種である. アルゴリズムを以下に示す.

バックトラック法のアルゴリズム

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 $t \leftarrow \rho t$ とする.

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

Step 2: $t_k = t$ として終了.

4.2 3-a,b,c

Python を用いて, 点 $\boldsymbol{x} = (x_0, x_1)^T$ が与えられたとき, $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$ を出力する関数を作成する. ここで,

$$f(x) = x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3$$

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

となる.

コードは付録のコード 5 に示す. なお, $\boldsymbol{x}$ や $\nabla f(x)$ を横ベクトルのように書いているが, これは numpy の仕様上都合がよいためである. 1 次元配列と行列の積を計算する際には自動で縦/横ベクトルのうち都合が良い方とみなされること, アクセスする際に表記が短くなることが理由として挙げられる. 仮に $\boldsymbol{x}$ を縦ベクトルとして, x=np.array([[x_0],[x_1]]) と書いてしまうと, アクセスの際には x[0][0] や x[1][0] とする必要があり, やや面倒である. なお、例えば $x = (0.3, 5)$ とすれば出力は以下のようになる。

```
␣␣␣␣f=␣651.439858807576
g=␣[␣-8.05014119␣509.4␣␣␣␣␣␣␣]
H=␣[[␣␣3.34985881␣␣-2.␣␣␣␣␣␣␣␣]
␣[␣-2.␣␣␣␣␣␣␣␣␣302.␣␣␣␣␣␣␣␣]]
```

4.3 4-a

バックトラック法を用いた最急降下法で最適化問題 (3) を解く．結果を表 5 に示す．

表 5: 最急降下法の結果

最適解	最適値	反復回数
$(-0.7334516, -0.4933272)^T$	3.5971380249598006	30

4.4 4-b

バックトラック法を用いたニュートン法で最適化問題 (3) を解く．結果を表 6 に示す．

表 6: ニュートン法の結果

最適解	最適値	反復回数
$(-0.73345172, -0.4933275)^T$	3.597138024959629	6

4.5 結果の考察

各手法で終了条件を変更した場合の反復回数を表 7 に示す．

表 7: 終了条件と反復回数

(a) 最急降下法のデータ

終了条件	反復回数
$2 * 10^{-3}$	16
$2 * 10^{-4}$	19
$2 * 10^{-5}$	24
$2 * 10^{-6}$	30
$2 * 10^{-7}$	33
$2 * 10^{-8}$	41
$2 * 10^{-9}$	111
$2 * 10^{-10}$	10000 以上

(b) ニュートン法のデータ

終了条件	反復回数
$2 * 10^{-3}$	5
$2 * 10^{-4}$	5
$2 * 10^{-5}$	5
$2 * 10^{-6}$	6
$2 * 10^{-7}$	6
$2 * 10^{-8}$	6
$2 * 10^{-9}$	7
$2 * 10^{-10}$	8
$2 * 10^{-11}$	8
$\vdots$	$\vdots$

最急降下法では $\epsilon = 2 * 10^{-10}$ から反復回数が 10000 以上と大幅に増加している．一方，ニュートン法では $\epsilon = 2 * 10^{-11}$ でも反復回数が 8 回と増加は非常に緩やかである．ニュートン法はヘッセ行列と勾配の計

算が必要であるため, 最急降下法より反復 1 回あたりの計算負荷は大きいが, 低次元においては高精度の解が高速に得られることがわかる. なお, 実行環境では, 反復回数が 9 回以上になると $|\nabla f(x^k)| = 0$ となり, $x = (-0.73345172 - 0.4933275)$ から更新されなかった.

5 課題 5

最急降下法およびニュートン法により, 以下の最適化問題を解く. ただし, 初期点は $x^0 = [2, 0]^T$ とし, バックトラック法における $\xi, \rho, \bar{t}$ は, それぞれ $\xi = 10^{-4}, \rho = 0.5, \bar{t} = 1$ とする.

$$\begin{aligned} &\text{minimize} \quad f(x) := \sum_{i=0}^{2} f_i(x)^2 \\ &\text{subject to} \quad x := (x_0, x_1)^T \in \mathbb{R}^2 \end{aligned} \tag{4}$$

ただし, $f_i(x) := y_i - [x]_0(1 - [x]_1^{i+1})(i = 0, 1, 2)$ と定義し, $y_0 = 1.5, y_1 = 2.25, y_2 = 2.625$ とする. 終了条件は $\epsilon = 2 * 10^{-6}$ とした.

5.1 準備

(4) での f の勾配とヘッセ行列は, それぞれ次のようになる.

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

ここで,

$$\nabla f_i(x) = (-1 + [x]_1^{i+1}, (i+1)[x]_0[x]_1^i)^T$$

$$\nabla^2 f_i(x) = \begin{cases} \begin{pmatrix} 0 & (i+1)[x]_i^i \\ (i+1)[x]_1^i & i(i+1)[x]_0[x]_1^{i-1} \end{pmatrix} & (i \geq 1) \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & (i = 0) \end{cases}$$

である.

5.2 計算結果

計算結果を表 8 に示す.

表 8: 終了条件と反復回数

(a) 最急降下法の結果

最適解	最適値	反復回数
$(2.9999952, 0.49999878)^T$	3.713324085577011e-12	734

(b) ニュートン法の結果

最適解	最適値	反復回数
$(3.00000013, 0.50000005)^T$	9.470356069887723e-15	5

5.3 結果の考察

課題 4 に続いて, この最適化問題でもニュートン法が有効であることがわかった.

6 付録

実験で用いたプログラムを記す. ただし, グラフの出力に用いたプログラムは省略している.

コード 1: 課題 1(b)

```
import os

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

def binary(a,b,e):
    ans = []
    mid = (a + b)*0.5
    ans.append(mid)
    while (abs(f(mid)) > e):
        if (f(mid) < 0):
            a = mid
            mid = (a + b)*0.5
            ans.append(mid)
        else:
            b = mid
            mid = (a + b)*0.5
            ans.append(mid)
    return ans

e = 10**-6
ans = []
ans.append(binary(-3.5,-2.5,e))
ans.append(binary(0,-1.5,e))
ans.append(binary(1,2.5,e))
print(len(binary(1,2.5,e)))

path = os.path.dirname(__file__) + "/res1.txt"
```

```
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(str(ans[i]))
        f.write("\n")
```

コード 2: 課題 1(c)

```
import os

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

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

def newton(start,e):
    ans = []
    x = start
    ans.append(x)
    while (abs(f(x)) > e):
        delta_x = -f(x)/df(x)
        x += delta_x
        ans.append(x)
    return ans


ans = []
e = 10**-6
ans.append(newton(-2.5,e))
ans.append(newton(-1.5,e))
ans.append(newton(2.5,e))
path = os.path.dirname(__file__) + "/res2.txt"
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(str(ans[i]))
        f.write("\n")
```

コード 3: 課題 2(a)

```
import os
def f(x):
    return x*x*x/3 - x*x - 3*x + 0.6

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

def d2f(x):
    return 2*x - 2


def saikyu(start,e):
    k = 0
    t = 1
    x = start
    ans = [[x,k]]
    while (abs(df(x)) > e):
        d = -df(x)
```

```
        t = 1/(k+1)
        x += t*d
        k += 1
        ans.append([x,k])
    return ans

ans = saikyu(0.5,10**-6)
path = os.path.dirname(__file__) + "/res2a.txt"
with open(path,mode="w") as f:
        f.write(str(ans))
```

コード 4: 課題 2(b)

```
import os
def f(x):
    return x*x*x/3 - x*x - 3*x + 0.6

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

def d2f(x):
    return 2*x - 2


def newton(start,e):
    k = 0
    t = 1
    x = start
    ans = [[x,k]]
    while (abs(df(x)) > e):
        d = -df(x)/d2f(x)
        x += t*d
        k += 1
        ans.append([x,k])
    return ans

ans = newton(5,10**-6)
path = os.path.dirname(__file__) + "/res2b.txt"
with open(path,mode="w") as f:
        f.write(str(ans))
```

コード 5: 課題 3

```
import os
import math
import numpy as np
def evalf(x): #関数値を求める関数
    return x[0]**2 + math.e**(x[0]) + x[1]**4 + x[1]**2 - 2*x[0]*x[1] + 3

def evalg(x): #勾配を求める関数
    g = np.array([2*x[0] + math.e**(x[0]) -2*x[1],
                4*x[1]**3 + 2*x[1] - 2*x[0]])

    return g

def evalh(x): #ヘッセ行列を求める関数
    H = np.array([[2+math.e**(x[0]), -2],
```

```
                  [-2, 12*x[1]**2 + 2]])

    return H

def main():
    x = np.array([0.3, 5])
    f = evalf(x)
    g = evalg(x)
    H = evalh(x)
    path = os.path.dirname(__file__) + "/res3.txt"
    with open(path,mode="w") as file:
        print("f=", f, "\ng=", g, "\nH=",H, file=file)

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

コード 6: 課題 4(a)

```
import os
import math
import numpy as np

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


def evalg(x):
    g = np.array([2*x[0] + math.e**(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.e**(x[0]), -2],
                  [-2, 12*x[1]**2 + 2]])

    return H


def saikyu(start,e):
    k = 0
    xi = 10**-4
    rho = 0.5
    x = start
    ans = [str(x) + ",␣" + str(evalf(x)) + ",␣" + str(k) + "\n"]
    while (np.linalg.norm(evalg(x)) > e):
        t = 1
        d = -evalg(x)
        while (evalf(x + t*d) > evalf(x) + xi*t*np.dot(d,evalg(x))):
            t *= rho
        x += t*d
        k += 1
        ans.append(str(x) + ",␣" + str(evalf(x)) + ",␣" + str(k) + "\n")
        if k > 5000:
            ans.append("calc␣aborted")
            break
    return ans
```

```

x = [1,1]
ans = saikyu(x,2*10**-6)
path = os.path.dirname(__file__) + "/res4a.txt"
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(str(ans[i]))
```

コード 7: 課題 4(b)

```
import os
import math
import numpy as np

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


def evalg(x):
    g = np.array([2*x[0] + math.e**(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.e**(x[0]), -2],
                  [-2, 12*x[1]**2 + 2]])

    return H


def newton(start,e):
    k = 0
    xi = 10**-4
    rho = 0.5
    x = start
    ans = [str(x) + ",␣" + str(evalf(x)) + ",␣" + str(k) + "\n"]
    while (np.linalg.norm(evalg(x)) > e):
        t = 1
        d = -np.dot(np.linalg.inv(evalh(x)), evalg(x))
        while (evalf(x + t*d) > evalf(x) + xi*t*np.dot(d,evalg(x))):
            t *= rho
        x += t*d
        k += 1
        ans.append(str(x) + ",␣" + str(evalf(x)) + ",␣" + str(k) + "\n")
        if k > 5000:
            ans.append("calc␣aborted")
            break
    return ans

x = [1,1]
ans = newton(x,2*10**-6)
path = os.path.dirname(__file__) + "/res4b.txt"
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(ans[i])
```

コード 8: 課題 5 の最急降下法

```
import os
import numpy as np

def f_i(x,i):
    y = [1.5, 2.25, 2.625]
    return y[i] - x[0]*(1 - x[1]**(i+1))

def gf_i(x,i):
    g = np.array([-1 + x[1]**(i+1), (i+1)*x[0]*x[1]**i])
    return g

def hf_i(x,i):
    if (i==0):
        h = np.array([[0,1],[1,0]])
    else:
        h = np.array([[0, (i+1)*x[1]**i],
                       [(i+1)*x[1]**i, i*(i+1)*x[0]*x[1]]])
    return h


def evalf(x):
    return f_i(x,0)**2 + f_i(x,1)**2 + f_i(x,2)**2


def evalg(x):
    g = np.array([0,0])
    for i in range(3):
        g = g + 2*f_i(x,i)*gf_i(x,i)
    return g

def evalh(x):
    H = np.zeros((2,2))
    for i in range(3):
        H = H + 2*(f_i(x,i)*hf_i(x,i) + np.dot(gf_i(x,i).reshape(2,1), gf_i(x,i).reshape(1,2)))
    return H


def saikyu(start,e):
    k = 0
    xi = 10**-4
    rho = 0.5
    x = start
    ans = []
    while (np.linalg.norm(evalg(x)) > e):
        t = 1
        d = -evalg(x)
        while (evalf(x + t*d) > evalf(x) + xi*t*np.dot(d,evalg(x))):
            t *= rho
        x = x + t*d
        k += 1
        ans.append(str(x)+",␣"+str(evalf(x))+",␣"+str(k)+"\n")
        if k > 5000:
            ans.append("calc␣aborted")
            break
    return ans
```

```

x = np.array([2,0])
ans = saikyu(x,2*10**-6)
path = os.path.dirname(__file__) + "/res5a.txt"
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(ans[i])
```

コード 9: 課題 5 のニュートン法

```
import os
import numpy as np

def f_i(x,i):
    y = [1.5, 2.25, 2.625]
    return y[i] - x[0]*(1 - x[1]**(i+1))

def gf_i(x,i):
    g = np.array([-1 + x[1]**(i+1), (i+1)*x[0]*x[1]**i])
    return g

def hf_i(x,i):
    if (i==0):
        h = np.array([[0,1],[1,0]])
    else:
        h = np.array([[0, (i+1)*x[1]**i],
                      [(i+1)*x[1]**i, i*(i+1)*x[0]*x[1]**(i-1)]])
    return h


def evalf(x):
    return f_i(x,0)**2 + f_i(x,1)**2 + f_i(x,2)**2


def evalg(x):
    g = np.array([0,0])
    for i in range(3):
        g = g + 2*f_i(x,i)*gf_i(x,i)
    return g

def evalh(x):
    H = np.zeros((2,2))
    for i in range(3):
        H = H + 2*(f_i(x,i)*hf_i(x,i) + np.dot(gf_i(x,i).reshape(2,1), gf_i(x,i).reshape
            (1,2)))
    return H


def newton(start,e):
    k = 0
    xi = 10**-4
    rho = 0.5
    x = start
    ans = []
    while (np.linalg.norm(evalg(x)) > e):
        t = 1
        d = -np.dot(np.linalg.inv(evalh(x)), evalg(x))
```

```
        d = np.ravel(d)
        while (evalf(x + t*d) > evalf(x) + xi*t*np.dot(d,evalg(x))):
            t *= rho
        x = x + t*d
        k += 1
        ans.append(str(x)+", "+str(evalf(x))+", "+str(k)+"\n")
        if k > 5000:
            ans.append("calc aborted")
            break
    return ans

x = np.array([2,0])
ans = newton(x,2*10**-6)
path = os.path.dirname(__file__) + "/res5b.txt"
with open(path,mode="w") as f:
    for i in range(len(ans)):
        f.write(ans[i])
```

7　参考文献

参考文献

[1] 実験演習ワーキンググループ (2022),「数理工学実験テキスト_2022 年版」