数理工学実験
テーマ５　連続最適化

2021年12月13日 提出

工学部情報学科数理工学コース２年
1029-32-7314　岡本淳志

この実験では、与えられた関数の零点や最適解（停留点）を、Python を用いて数値計算によって求める手法を学ぶ。具体的には、関数の零点を求める手法として二分法とニュートン法について、最適化手法として最急降下法とニュートン法について学ぶ。

1 課題 1

- `https://sukkiri.jp/technologies/ides/anaconda-win_install.html`

を参考にして、Anaconda3 を用いて Python を PC にインストールした。

2 課題 2

次の関数

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

を、Python を用いて、

- (a) 関数 f を $-10 \leq x \leq 10$, $-10 \leq y \leq 10$ の範囲で描写
- (b) 関数 f の零点を二分法を用いて求める
- (c) 関数 f の零点をニュートン法を用いて求める

ことを考える。

ここで、二分法とニュートン法の導入を行う。

2.1 実験準備

関数の零点探索のための二分法とニュートン法を導入する。

2.1.1 二分法

二分法のアルゴリズムを以下に示す。

- 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 $\varepsilon(> 0)$ を決める。
- Step 1: a と b の中間点 $c = \frac{a+b}{2}$ 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: $f(c) < 0$ であれば $a \leftarrow c$ とし、$f(c) \geq 0$ であれば $b \leftarrow c$ として、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。しかし、一般的には収束が遅い。より収束が早い手法として、ニュートン法が知られている。

2.1.2 ニュートン法

ニュートン法のアルゴリズムを以下に示す。

- Step 0: 初期点 x_0 を選び、終了条件 $\varepsilon(>0)$ を決める。
- Step 1: ニュートン方程式

$$f'(x_k)\Delta x_k = -f(x_k) \tag{2}$$

  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、$f(x_(k+1))| \leq \varepsilon$ 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。

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。

2.2 実験方法

課題 2 を行うためのコードを付録のソースコード 1 に示す。
ニュートン法で用いる $f'(x)$ は、

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

である。
また、(b),(c) は、(a) で描写した関数 f の概形を参考にして初期点を適切に設定する。

2.3 実験結果

得られた図（(a) の結果）を図 1 に示す。

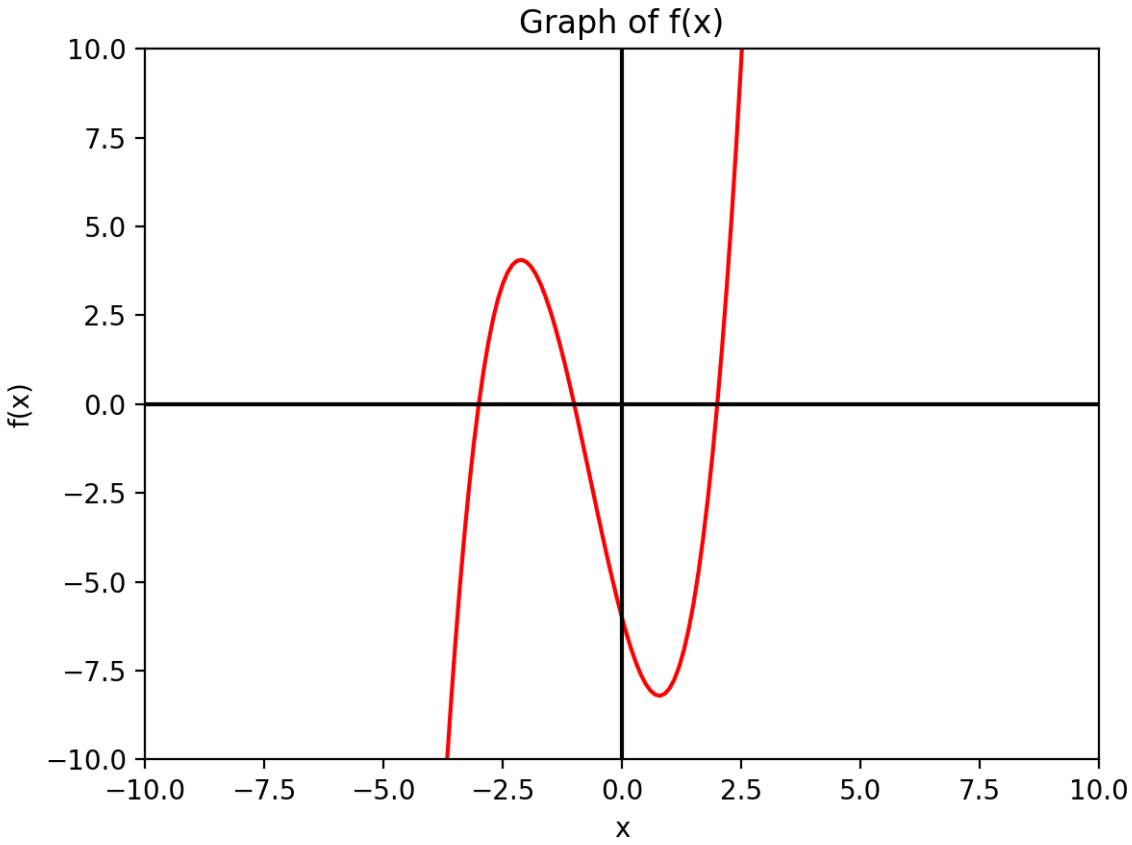

図 1: 関数 f を $-10 \leq x \leq 10$, $-10 \leq y \leq 10$ の範囲で描写したもの

また、求めた零点の値（(b),(c) の結果）を表 3 に示す。

終了条件の ε はいずれも、$\varepsilon = 10^{-6}$ とした。

(b) の初期点は (a) の結果を参考にし、$f(a) < 0$, $f(b) \geq 0$ を満たす異なる 3 つの初期点の組み a, b を以下のように選んだ。

表 1: 二分法の（異なる 3 つの）初期点

i	a_i	b_i
1	-5.0	-2.5
2	0	-2.5
3	0	2.5

(c) の初期点は (a) の結果を参考にし、零点付近にある初期点として、異なる 3 つの初期点 x_0 を以下のように選んだ。

表 2: ニュートン法の（異なる 3 つの）初期点

i	x_{0_i}
1	-3.75
2	-1.25
3	2.5

表 3: 関数 f の零点を二分法とニュートン法を用いて求めた結果（「零点」の後の数字はそれぞれ同じ添字を持つ初期点から得られた値であることを意味している）

	零点 1	零点 2	零点 3
二分法	-2.9999999701976776	-0.9999999403953552	2.0000000298023224
ニュートン法	-3.0000000000000164	-0.9999999995869381	2.0000000000586944

2.4 考察

関数 f のグラフを（増減表などを用いて）解析的に描写すると、概形が図 1 と同じグラフが得られた。また、関数 f の零点を解析的に求めると、$x = -3, -1, 2$ であるので、二分法, ニュートン法ともに高い精度で数値解が得られたと言える。この精度は終了条件 $\varepsilon(> 0)$ に依存するものであるので、ε をより小さくとれば、より精度を高くすることができるはずである。

3 課題 3

次の関数

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

を、Python を用いて、

- (a) 最急降下法により関数 f の停留点を求める（ただし、初期点 $x_0 = \frac{1}{2}$, ステップサイズ $t_k = \frac{1}{k+1}$ とする）
- (b) ニュートン法により関数 f の停留点を求める（ただし、初期点 $x_0 = 5$, ステップサイズ $t_k = 1$ とする）

ことを考える。

ここで、最急降下法とニュートン法の導入を行う。

3.1 実験準備

最適化手法として代表的な降下法の概要を説明し、具体例として最急降下法とニュートン法を導入する。

3.1.1 降下法

今回は最適化問題の一つとして、局所的最小解（停留点）を求めることを考える。ある関数に対して、ある点が局所的最小解や大域的最小解であれば、その点は停留点である（逆は言えない）。すなわち、$\nabla f(x) = 0$ を満たす。

今回扱う降下法（最急降下法とニュートン法）は、この停留点を求める手法である。先述のように、最適化問題では局所的最小解を求めることに帰着する場合が多いが、大抵の場合この停留点を求めることから局所的最小解を求めることができる。

降下法とは、$f(x^0) > f(x^1) > ...$ となる点列 $\{x^k\}$ を生成し、局所的最小解（停留点）を求める手法である。そのアルゴリズムを以下に示す。

- Step 0: 初期点 x^0, 終了条件 $\varepsilon(> 0)$ を決める。$k = 0$ とする。
- Step 1: もし、x^k が終了条件 $||\nabla f(x^k)|| \leq \varepsilon$ 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法 d^k を定める。

$$\langle d^k, \nabla f(x^k) \rangle < 0 \tag{5}$$

- Step 3: 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。
- 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、Step 1 へ戻る。

Step 2 の、式 5 を満たしているベクトル d^k のことを降下方向と呼ぶ。降下方向へ適切に進めば、関数値は減少するが、進みすぎると増加する場合がある。そのため、$f(x^k) > f(x^{k+1})$ 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 を決める方法は直線探索法と呼ばれ、今回（課題 5 以降で）はバックトラック法を用いる。

Step 1 における終了条件 ε は、$\varepsilon = n * 10^{-6}$ のように設定されることが多い。これは、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ため、その誤差を許容するということである。

降下法の具体例として、最急降下法とニュートン法を次に示す。（これらの降下方向は、式 5 を満たす。）

3.1.2 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

$$d^k = -\nabla f(x^k) \tag{6}$$

を用いる降下法である。

3.1.3 ニュートン法

ニュートン法は、点 x^k が与えられた時に、降下方向として

$$d^k = -\nabla^2 f(x^k)^{-1}\ \nabla f(x^k) \tag{7}$$

を用いる降下法である。ただし、$\nabla^2 f(x)$ は x における f のヘッセ行列を表している。

3.2 実験方法

課題 3 を行うためのコードを付録のソースコード 2 に示す。
また、

$$\nabla f(x) = f'(x) = x^2 - 2x - 3 \tag{8}$$

$$\nabla^2 f(x) = f''(x) = 2x - 2 \tag{9}$$

である。

初期点とステップサイズは先述のようにした。今回は 1 次元の問題であるので、終了条件を、$\varepsilon = 10^{-6}$ と設定した。最大反復回数は、$k < 100$ と設定した。

3.3 実験結果

求めた停留点の値（と反復回数 k）を表 4 に示す。

表 4: 関数 f の停留点を最急降下法とニュートン法を用いて求めた結果とその反復回数

手法	停留点	反復回数 k
最急降下法	3.0	1
ニュートン法	3.0000000929222947	4

3.4 考察

関数 f の停留点（この場合局所的最小解）を解析的に求めると、$x = 3$ であるので、最急降下法, ニュートン法ともに高い精度で数値解が得られたと言える。特に、最急降下法では一回の試行で解析値と同等の数値解を得られた。前問と同様、この精度は終了条件 $\varepsilon(> 0)$ に依存するものであるので、ε をより小さくとれば、より精度を高くすることができるはずである。

またこの実験より、一般にはニュートン法の方が最急降下法に比べて速く収束するとされているが、本題のように最適化問題で扱われる目的関数が単純で、反復回数が比較的数なくて済む場合、ニュートン法の方が収束が遅くなることが分かった。これは、ニュートン法では降下方向 d^k を定める際に、ヘッセ行列を用い、式が最急降下法のそれよりも少し複雑になることなどが原因ではないかと考えた。

4 課題 4 & 5

次の 2 次元の最適化問題を考える。

$$\begin{aligned} &\text{minimize} \quad f(x) = x_0^2 + e^{x_0} + x_1^4 + x_1^2 - 2x_0x_1 + 3 \\ &\text{subject to} \quad x = (x_0, x_1)^\top \in \mathbb{R}^2 \end{aligned} \tag{10}$$

Python を用いて、

- (4.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する。
- (4.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する。
- (4.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する。
- (5.a) バックトラック法を用いた最急降下法を実装し、この最適化問題を解く。
- (5.b) バックトラック法を用いたニュートン法を実装し、この最適化問題を解く。

ただし、(5.a),(5.b) の初期点は $x^0 = (1, 1)^\top$ を用い、バックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定する。

ここで、バックトラック法の導入を行う。

4.1 実験準備

前セクションで触れた、直線探索法の一種であるバックトラック法を導入する。

4.1.1 バックトラック法

バックトラック法は、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ステップサイズ $t_k(> 0)$ を決定する直線探索法の中でも理論的にも実用的にも優れていると知られている。バックトラック法のアルゴリズムを以下に示す。

- 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。

$$f(x^k + td^k) \leq f(x^k) + \xi t \langle d^k, \nabla f(x^k) \rangle \tag{11}$$

- 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。横軸をステップサイズ t, 縦軸を $\varphi(t) = f(x^k + td^k)$（Step 1 の不等式の左辺）としたグラフで考えると、原点における φ の接線は、$y = f(x^k) + t\langle d^k, \nabla f(x^k) \rangle$ とである。アルミホ条件を満たすステップサイズを選ぶことは、φ の接戦の傾きを緩和して得られる直線 $y = f(x^k) + \xi t \langle d^k, \nabla f(x^k) \rangle$（Step 1 の不等式の右辺）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。

4.2 実験方法

課題 4,5 を行うためのコードを付録のソースコード 3 に示す。
また、

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

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

である。

(4,a),(4,b),(4,c) に対応する関数はそれぞれ、ソースコード 3 中の「evalf」,「evalg」,「evalh」である。

今回は 2 次元の問題であるので、終了条件を、$\varepsilon = 2 * 10^(-6)$ と設定した。最大反復回数は、$k < 100$ と設定した。

4.3 実験結果

求めた停留点の値（と反復回数 k）を表 5 に示す。

表 5: 関数 f の停留点を最急降下法とニュートン法を用いて求めた結果とその反復回数

手法	停留点	反復回数 k
最急降下法	$(-0.7334516,\ -0.4933272)^\top$	30
ニュートン法	$(-0.73345172,\ -0.4933275)^\top$	6

4.4 考察

表 5 より、最急降下法, ニュートン法で得た数値解の差（の絶対値）が、10^{-6} 以下になっているので、微小の誤差はあるが、この点が関数 f の停留点であると言える。前問と同様、この精度は終了条件 $\varepsilon(>0)$ に依存するものであるので、ε をより小さくとれば、より精度を高くすることができるはずである。

またこの実験より、「一般に、ニュートン法の方が最急降下法に比べて速く収束する」という性質を確認することができた。課題 3 との違いは、問題が 2 次元になったことや、関数 f が e^{x_0} という項を持つことなどだと考えられる。

5 課題 6

次の最適化問題を考える。

$$\begin{aligned} &\text{minimize} \quad f(x) = \sum_{i=0}^{2} f_i(x)^2 \\ &\text{subject to} \quad x \in \mathbb{R}^2 \end{aligned} \tag{14}$$

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。

この問題を Python を用いて、最急降下法およびニュートン法を実装し解く。初期点は $x^0 = (2, 0)^\top$ を用い、バックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定する。

5.1 実験準備

この問題を解く前に、ニュートン法で用いる $\{\nabla^2 f(x^k)\}$ が一様正定値であるという仮定が成り立たない場合について考える。そのことを考えるにあたって有効な定理を導入する。

- 定理：対称行列が正定値であることは、その固有値が全て正であることと同値である。

ここまでニュートン法で用いる $\{\nabla^2 f(x^k)\}$ が「有界かつ一様正定値である」と仮定したが、一様正定値性については成り立たないことが多々ある。このような場合に対応するために、次に述べるような修正を行う。

各反復 k に対して、ヘッセ行列 $\nabla^2 f(x^k)$ に単位行列 I の τ 倍を加えた次の方程式を考える。（式 (7) 参考）

$$(\nabla^2 f(x^k) + \tau I)d^k = -\nabla f(x^k) \tag{15}$$

もし、$\nabla^2 f(x^k)$ が一様正定値であるならば、$\tau = 0$ とすれば良い。$\nabla^2 f(x^k)$ が一様正定値でない場合、$\nabla^2 f(x^k) + \tau I$ の固有値は $\lambda_j(nabla^2 f(x^k)) + \tau$ $(j = 1, ..., n)$ となることを考えると、$\nabla^2 f(x^k)$ の最小固有値の絶対値にある定数を加えた値を τ とすれば、$\lambda_j(nabla^2 f(x^k)) + \tau > 0$ $(j = 1, ..., n)$ となり、上記の定理より、$\nabla^2 f(x^k) + \tau I$ が一様正定値となる。よってこの τ を用いた式（15）から d^k を生成し、ニュートン方向の修正を行う。

5.2 実験方法

課題 6 を行うためのコードを付録のソースコード 4 に示す。

また、

$$\nabla f(x) = 2\sum_{i=0}^{2} f_i(x)\nabla f_i(x) \tag{16}$$

$$\nabla^2 f(x) = 2\sum_{i=0}^{2}\Big(f_i(x)\nabla^2 f_i(x) + \nabla f_i(x)\nabla f_i(x)^\top\Big) \tag{17}$$

であり、

$$\nabla f_i(x) = \begin{pmatrix} -(1 - x_1^{i+1}) \\ (i+1)x_0 x_1^i \end{pmatrix} \tag{18}$$

$$\nabla^2 f_i(x) = \begin{pmatrix} 0 & (i+1)x_1^i \\ (i+1)x_1^i & i(i+1)x_0 x_1^{i-1} \end{pmatrix} \tag{19}$$

である。これらの式から、$\nabla^2 f(x)$ は対称行列（これに単位行列のスカラ倍を加えた行列も対称行列）であるので、先述の正定値に関する定理を用いることができる。

今回は 2 次元の問題であるので、終了条件を、$\varepsilon = 2 * 10^(-6)$ と設定した。最大反復回数は、$k < 1000$ と設定した。

また、プログラムの実行中に、配列の要素が 0^n で値が”nan”とされる部分があったので、ここで必要としている値 0 を代入するようにプログラムを変更した。（np.nan_to_num() の部分）

5.3 実験結果

求めた最適解と最適値（と反復回数 k）を表 6 に示す。

表 6: 関数 f の最適解と最適値を、最急降下法とニュートン法を用いて求めた結果とその反復回数

手法	最適解	最適値	反復回数 k
最急降下法	$(2.9999952,\ 0.49999878)^\top$	3.713324085577011e-12	734
ニュートン法	$(3.00000013,\ 0.50000005)^\top$	9.47035602522931e-15	6

5.4 考察

与えられた最適解と最適値はそれぞれ、$(3, 0.5)^\top$, 0 である。表 6 より、最急降下法, ニュートン法ともに、高い精度で数値解が得られたといえる。前問と同様、この精度は終了条件 $\varepsilon(>0)$ に依存するものであるので、ε をより小さくとれば、より精度を高くすることができるはずである。

またこの実験より、最急降下法では反復回数が $k = 734$ となっているのに対し、ニュートン法では $k = 6$ となっていることから、「一般に、ニュートン法の方が最急降下法に比べて速く収束する」という性質を顕著に確認することができた。課題 5 よりもこの差が大きく現れたのは、課題 6 の関数 f がより複雑な形をしていていることなどが原因だと考えられる。

6 課題 7

次の最適化問題を考える。

$$\begin{aligned} &\text{minimize} \quad f(x) = \frac{1}{2}\langle Ax, x\rangle \\ &\text{subject to} \quad x \in \mathbb{R}^n \end{aligned} \tag{20}$$

ただし、$A \in \mathbb{R}^{n\times n}$ 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、$A = Z^\top Z$ により定義される半正定値対称行列である。

この問題を Python を用いて、最急降下法およびニュートン法を実装し解く。初期点は要素が全て 1 のベクトル用い、バックトラック法における ξ, ρ, $\bar{t}$ は、それぞれ $\xi = 10^{-4}$, $\rho = 0.5$, $\bar{t} = 1$ と設定する。

このとき、$n = 2, 5, 10$ のそれぞれに対して 5 回計算を実行する。また、各手法と各 n にたいして、反復回数 (k) の平均 $\bar{k}_n$ を計算する。

6.1 実験方法

課題 7 を行うためのコードを付録のソースコード 5 に示す。
また、

$$\nabla f(x) = Ax \tag{21}$$

$$\nabla^2 f(x) = A \tag{22}$$

である。

今回は n 次元の問題であるので、終了条件を、$\varepsilon = n * 10^(-6)$ と設定した。最大反復回数は、$k < 10^5$ と設定した。(初めは、$k < 10^3$ と設定していたが、収束しない場合があったので最大反復回数を大きくしていった。)

乱数の生成は、一様分布の乱数生成 (np.random.rand()) を用いて行なった。

また、各 n に対して 5 回 A を定義し、それぞれにおいて最急降下法とニュートン法を用いる、という方法をとった。

6.2 実験結果

出力の全体 (生成した行列 $(A : A_n)$, それぞれの手法による最適解と最適値と反復回数 (k), それぞれの手法の反復回数 (k) の平均) を付録のソースコード 5 の次に示す。

最急降下法, ニュートン法ともに、最適解は、全ての要素が 0 に近い値を、最適値は 0 に近い値をとった。

ニュートン法の反復回数は、いずれの n に対しても全て $k = 1$ であった (つまり、$\bar{k}_n = 1$)。最急降下法の反復回数を表 7 に示す。

表 7: $n = 2, 5, 10$ に対して、関数 f の最適解を最急降下法を用いて 5 回ずつ計算したときの反復回数とその平均

	$n = 2$	$n = 5$	$n = 10$
1 回目	42	19164	3573
2 回目	788	1547	2608
3 回目	66	1413	85077
4 回目	289	2725	1585
5 回目	100	1493	2384
平均 $\bar{k}_n$	257.0	5268.4	19045.4

プログラムが実行を完了するのには 10 秒程度かかった。

6.3　考察

「最急降下法, ニュートン法ともに、最適解は、全ての要素が 0 に近い値を、最適値は 0 に近い値をとった。」とした。ただ、最急降下法ではいずれの n に対しても、最適解の要素が 10^{-2}〜10^{-6} 程度、最適値が 10^{-7}〜10^{-12} という出力結果であったが、ニュートン法では、いずれの n に対しても、最適解の要素が 10^{-9}〜10^{-16} 程度、最適値が 10^{-18}〜10^{-32} という出力結果で、手法の間に大きく差が開いた結果となった。

行列 $A \in \mathbb{R}^{n\times n}$ の次元が変わると最急降下法が要する反復回数が大きく異なることが分かったが、同じ次元でも生成される行列 A によって反復回数が大きく異なることも分かった。

この問題に関しては最急降下法よりニュートン法が適した手法であるというのは実験結果より明らかである。ニュートン法において、反復回数が $k = 1$ で収束したのは、

$$\begin{aligned} d^0 &= -\nabla^2 f(x^0)^{-1} \nabla f(x^0) \\ &= -A^{-1} A x^0 \\ &= -x^0 \end{aligned} \tag{23}$$

であることと初期点 $x^0 = [1, ..., 1]$ と設定していたこと、最適解が $[0, ..., 0]$ と推測できることなどが理由であると考えた。

7　総括

零点探索において、零点付近に初期点を設定できるなら、二分法よりもニュートン法の方が収束が速いという点で優れた手法だと言える。

しかし、最適化問題を考える際、問題によって適した手法が異なることが分かった。一般に最急降下法よりもニュートン法の方が収束が速いとされていて、今回の実験でもニュートン法の方が収束が速く、精度も高い場合があったが、逆に最急降下法のほうが収束が速く、精度も高い場合もあった。また、今回の実験では現れなかったが、問題の規模が大きくなれば、最急降下法では計算が可能であってもニュートン法では計算が困難となる場合も有り得る。最適化問題を解く際にはニュートン法の方が優れた手法であるとは一概には言えないことが分かった。

ある問題を解く際、その問題が一見ではどの手法が適しているのかわからない場合があるので、負担が大きくならない場合は複数の手法で確かめるのが賢明であると思った。

8 付録

ソースコード 1: Python を用いて、関数の描写と、零点を求めるプログラム

```
1  # 数理工学実験 #テーマ 5 #課題 2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  def func(x): # f(x)の関数定義
7      return x**3+2*x**2-5*x-6
8
9
10 def dfunc(x): # f(x)の導関数f'(x)の関数定義
11     return 3*x**2+4*x-5
12
13
14 x = np.arange(-10, 10, 0.1)
15 y = func(x)
16 plt.plot(x, y, "red")
17 plt.xlim(-10, 10)
18 plt.ylim(-10, 10)
19 plt.axhline(0, c="black")
20 plt.axvline(0, c="black")
21 plt.xlabel("x")
22 plt.ylabel("f(x)")
23 plt.title("Graph of f(x)")
24 plt.show()
25
26 epsilon = 10**(-6) # 終了条件 ε の定義
27
28 # 二分法ここから
29 a = [-5.0, 0.0, 0.0] # 異なる 3 つの零点に対するa の初期値
30 b = [-2.5, -2.5, 2.5] # 異なる 3 つの零点に対するb の初期値
31 c = [(a[0]+b[0])/2, (a[1]+b[1])/2, (a[2]+b[2])/2]
32
33 zero1 = [None]*3 # 二分法により求めた零点を格納する配列
34
35 while zero1[0] is None or zero1[1] is None or zero1[2] is
       None:
36     for i in range(0, 3, 1):
37         if np.abs(func(c[i])) > epsilon:
38             c[i] = (a[i]+b[i])/2
39             if func(c[i]) < 0:
40                 a[i] = c[i]
41             else:
```

```
42                 b[i] = c[i]
43         else:
44             zero1[i] = c[i]
45
46 print("二分法により求めた零点")
47 for i in range(0, 3, 1):
48     print(zero1[i])
49 # 二分法ここまで
50
51 print("\n")
52
53 # ニュートン法ここから
54 xk = [-3.75, -1.25, 2.5] # 零点付近にある初期点x_0
55
56 zero2 = [None]*3 # ニュートン法により求めた零点を格納する配列
57
58 while zero2[0] is None or zero2[1] is None or zero2[2] is
       None:
59     for i in range(0, 3, 1):
60         if np.abs(func(xk[i])) > epsilon:
61             deltaX = -func(xk[i])/dfunc(xk[i])
62             xk[i] = xk[i]+deltaX
63         else:
64             zero2[i] = xk[i]
65
66 print("ニュートン法により求めた零点")
67 for i in range(0, 3, 1):
68     print(zero2[i])
69 # ニュートン法ここまで
```

ソースコード 2: Python を用いて、関数の停留点を求めるプログラム

```
1  # 数理工学実験 #テーマ 5 #課題 3
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  def func(x): # f(x)の関数定義
7      return 1/3.0*x**3-x**2-3*x+5/3.0
8
9
10 def dfunc(x): # f(x)の導関数f'(x)の関数定義
11     return x**2-2*x-3
12
13
14 def d2func(x): # f''(x)の関数定義
```

```
15      return 2*x-2
16
17
18  x = np.arange(-10, 10, 0.1)
19  y = func(x)
20  plt.plot(x, y, "red")
21  plt.xlim(-10, 10)
22  plt.ylim(-10, 10)
23  plt.axhline(0, c="black")
24  plt.axvline(0, c="black")
25  plt.xlabel("x")
26  plt.ylabel("f(x)")
27  plt.title("Graph of f(x)")
28  plt.show()
29
30  epsilon = 10**(-6) # 終了条件εの定義
31
32  # 最急降下法ここから
33  k = 0
34  xk = 0 # 初期点
35  while k < 100:
36      if np.abs(dfunc(xk)) > epsilon:
37          dk = -dfunc(xk) # 降下方向
38          tk = 1/(k+1) # ステップサイズ
39          xk = xk+tk*dk # 点xkの更新
40          k = k+1
41      else:
42          sp1 = xk # 停留点
43          break
44  print("最急降下法により求めた停留点")
45  print(sp1)
46  print("k=", k)
47  # 最急降下法ここまで
48
49  print("\n")
50
51  # ニュートン法ここから
52  k = 0
53  xk = 5 # 初期点
54  tk = 1 # ステップサイズ
55  while k < 100: # 最大反復回数
56      if np.abs(dfunc(xk)) > epsilon:
57          dk = -dfunc(xk)/d2func(xk) # 降下方向
58          xk = xk+tk*dk # 点xkの更新
59          k = k+1
60      else:
```

```
61          sp2 = xk # 停留点
62          break
63  print("ニュートン法により求めた停留点")
64  print(sp2)
65  print("k=", k)
66  # ニュートン法ここまで
```

ソースコード 3: Python を用いて、関数の停留点を求めるプログラム

```
1   # 数理工学実験 #テーマ 5 #課題 4,5
2   import numpy as np
3   import matplotlib.pyplot as plt
4
5
6   def evalf(x):
7       # 目的関数の値f(x)を計算する
8       f = x[0]**2 + np.exp(x[0]) + x[1]**4 + x[1]**2 - 2*x
            [0]*x[1] + 3
9       return f
10
11
12  def evalg(x):
13      # f の勾配を計算する
14      g = np.array([2*x[0] + np.exp(x[0]) - 2*x[1], 4*x
            [1]**3 + 2*x[1] - 2*x[0]])
15      return g
16
17
18  def evalh(x):
19      # f のヘッセ行列を計算する
20      H = np.array([[2 + np.exp(x[0]), -2], [-2, 12*x[1]**2
            + 2]])
21      return H
22
23
24  epsilon = 2*10**(-6) # 終了条件 ε の定義
25
26  # 最急降下法ここから
27  k = 0
28  xk = np.array([1.0, 1.0]) # 初期点
29  rho = 0.5 # バックトラック法の ρ
30  kusai = 10**(-4) # バックトラック法の ξ
31  while k < 100:
32      if np.linalg.norm(evalg(xk)) > epsilon:
33          dk = -evalg(xk) # 降下方向
34          tk = 1.0 # t の初期値
```

```
35          while evalf(xk+tk*dk) > evalf(xk)+kusai*tk*np.dot(dk,
                evalg(xk)):
36              tk = rho*tk # ステップサイズ
37          xk = xk+tk*dk # 点xk の更新
38          k = k+1
39      else:
40          sp1 = xk # 停留点
41          break
42  print("最急降下法により求めた停留点")
43  print(sp1)
44  print("k=", k)
45  # 最急降下法ここまで
46
47  print("\n")
48
49  # ニュートン法ここから
50  k = 0
51  xk = np.array([1.0, 1.0]) # 初期点
52  rho = 0.5 # バックトラック法の ρ
53  kusai = 10**(-4) # バックトラック法の ξ
54  while k < 100: # 最大反復回数
55      if np.linalg.norm(evalg(xk)) > epsilon:
56          dk = -np.dot(np.linalg.inv(evalh(xk)), evalg(xk)) #
                降下方向
57          tk = 1.0 # t の初期値
58          while evalf(xk+tk*dk) > evalf(xk)+kusai*tk*np.dot(dk,
                evalg(xk)):
59              tk = rho*tk # ステップサイズ
60          xk = xk+tk*dk # 点xk の更新
61          k = k+1
62      else:
63          sp2 = xk # 停留点
64          break
65  print("ニュートン法により求めた停留点")
66  print(sp2)
67  print("k=", k)
68  # ニュートン法ここまで
```

ソースコード 4: Python を用いて、関数の最適解を求めるプログラム

```
1  # 数理工学実験 #テーマ 5 #課題 6
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  y = [1.5, 2.25, 2.625]
6
```

```
7
8  def evalfi(x, i): # f_i(x)
9      fi = y[i]-x[0]*(1-x[1]**(i+1))
10     return fi
11
12
13 def evalgi(x, i): # ∇(f_i(x))
14     gi = np.array([-(1-x[1]**(i+1)), (i+1)*x[0]*x[1]**i])
15     return gi
16
17
18 def evalhi(x, i): # ∇^2(f_i(x))
19     hi = np.array(
20         [[0, (i+1)*x[1]**i], [(i+1)*x[1]**i, i*(i+1)*x[0]*
               x[1]**(i-1)]])
21     return hi
22
23
24 def evalf(x): # 目的関数の値f(x)を計算する
25     f = 0
26     for i in range(3):
27         f = f+evalfi(x, i)**2
28     return f
29
30
31 def evalg(x): # f の勾配を計算する
32     g = np.array([0, 0])
33     for i in range(3):
34         g = g+2*(evalfi(x, i)*evalgi(x, i))
35     return g
36
37
38 def evalh(x): # f のヘッセ行列を計算する
39     H = np.array([[0, 0], [0, 0]])
40     for i in range(3):
41         m = evalgi(x, i)
42         M = np.array([[m[0]**2, m[0]*m[1]], [m[1]*m[0], m
               [1]**2]])
43         H = H+2*(evalfi(x, i)*evalhi(x, i))+2*M
44     return H
45
46
47 epsilon = 2*10**(-6) # 終了条件εの定義
48
49 # 最急降下法ここから
50 k = 0
```

```
xk = np.array([2.0, 0]) # 初期点
rho = 0.5 # バックトラック法のρ
kusai = 10**(-4) # バックトラック法のξ
while k < 1000:
    if np.linalg.norm(evalg(xk)) > epsilon:
        dk = -evalg(xk) # 降下方向
        tk = 1.0 # t の初期値
        while evalf(xk+tk*dk) > evalf(xk)+kusai*tk*np.dot(dk,
             evalg(xk)):
            tk = rho*tk # ステップサイズ
        xk = xk+tk*dk # 点xk の更新
        k = k+1
    else:
        sp1 = xk # 停留点
        break
print("最急降下法")
print("最適解は、", sp1)
print("最適値は、", evalf(sp1))
print("k=", k)
# 最急降下法ここまで

print("\n")

# ニュートン法ここから
k = 0
xk = np.array([2.0, 0]) # 初期点
rho = 0.5 # バックトラック法のρ
kusai = 10**(-4) # バックトラック法のξ
I = np.array([[1, 0], [0, 1]]) # 単位行列
while k < 1000: # 最大反復回数
    if np.linalg.norm(evalg(xk)) > epsilon:
        # f のヘッセ行列の固有値（と固有ベクトル）を取得
        w, v = np.linalg.eig(np.nan_to_num(evalh(xk)))
        if w[0] > 0 and w[1] > 0: # 正定値の場合
            tau = 0
        else:
            tau = np.abs(min(w))+10**(-2) # ニュートン法の修
                正で用いるτ
        dk = -np.dot(np.linalg.inv(np.nan_to_num(evalh(xk)+
            tau*I)),
                     evalg(xk)) # 降下方向
        tk = 1.0 # t の初期値
        while evalf(xk+tk*dk) > evalf(xk)+kusai*tk*np.dot(dk,
             evalg(xk)):
            tk = rho*tk # ステップサイズ
        xk = xk+tk*dk # 点xk の更新
```

```
93          k = k+1
94      else:
95          sp2 = xk # 停留点
96          break
97  print("ニュートン法")
98  print("最適解は、", sp2)
99  print("最適値は、", evalf(sp2))
100 print("k=", k)
101 # ニュートン法ここまで
```

ソースコード 5: Python を用いて、関数の最適解を求めるプログラム

```
1   # 数理工学実験 #テーマ 5 #課題 7
2   import numpy as np
3   import matplotlib.pyplot as plt
4
5   n = [2, 5, 10]
6
7
8   def evalA(i): # 行列A を生成する
9       Z = np.random.rand(n[i], n[i])
10      A = np.dot(Z.T, Z)
11      return A
12
13
14  def evalf(A, x): # 目的関数の値f(x)を計算する
15      f = 0.5*np.dot(np.dot(A, x), x)
16      return f
17
18
19  def evalg(A, x): # f の勾配を計算する
20      g = np.dot(A, x)
21      return g
22
23
24  def evalh(A): # f のヘッセ行列を計算する
25      H = A
26      return H
27
28
29  rho = 0.5 # バックトラック法の ρ
30  kusai = 10**(-4) # バックトラック法の ξ
31
32  for i in range(3):
33
34      epsilon = n[i]*10**(-6) # 終了条件 ε の定義
```

```
35
36      k1 = [0]*5 # 最急降下法の反復回数
37      k2 = [0]*5 # ニュートン法の反復回数
38
39      I = np.eye(n[i]) # 単位行列
40
41      for j in range(5):
42          An = evalA(i)
43          print("An=", An)
44
45          # 最急降下法ここから
46          xk1 = np.array([1]*n[i]) # 初期点
47          while k1[j] < 100000:
48              if np.linalg.norm(evalg(An, xk1)) > epsilon:
49                  dk = -evalg(An, xk1) # 降下方向
50                  tk = 1.0 # t の初期値
51                  while evalf(An, xk1+tk*dk) > evalf(An, xk1)+
                        kusai*tk*np.dot(dk, evalg(An, xk1)):
52                      tk = rho*tk # ステップサイズ
53                  xk1 = xk1+tk*dk # 点xk の更新
54                  k1[j] = k1[j]+1
55              else:
56                  sp1 = xk1 # 停留点
57                  break
58          print("最急降下法")
59          print("n=", n[i], ",", j+1, "回目")
60          print("最適解は、", sp1)
61          print("最適値は、", evalf(An, sp1))
62          print("k=", k1[j])
63          # 最急降下法ここまで
64
65          # ニュートン法ここから
66          xk2 = np.array([1]*n[i]) # 初期点
67          while k2[j] < 100000: # 最大反復回数
68              if np.linalg.norm(evalg(An, xk2)) > epsilon:
69                  w, v = np.linalg.eig(evalh(An))
70                  if np.all(w > 0): # 正定値の場合
71                      tau = 0
72                  else:
73                      tau = np.abs(min(w))+10**(-2) # ニュート
                            ン法の修正で用いる τ
74                  dk = -np.dot(np.linalg.inv(evalh(An)+tau*I),
75                               evalg(An, xk2)) # 降下方向
76                  tk = 1.0 # t の初期値
77                  while evalf(An, xk2+tk*dk) > evalf(An, xk2)+
                        kusai*tk*np.dot(dk, evalg(An, xk2)):
```

```
78                      tk = rho*tk # ステップサイズ
79                  xk2 = xk2+tk*dk # 点xk の更新
80                  k2[j] = k2[j]+1
81              else:
82                  sp2 = xk2 # 停留点
83                  break
84          print("ニュートン法")
85          print("n=", n[i], ",", j+1, "回目")
86          print("最適解は、", sp2)
87          print("最適値は、", evalf(An, sp2))
88          print("k=", k2[j])
89          # ニュートン法ここまで
90          print("\n")
91
92      print("最急降下法のk の平均=", (k1[0]+k1[1]+k1[2]+k1[3]+
            k1[4])/5.0)
93      print("ニュートン法のk の平均=", (k2[0]+k2[1]+k2[2]+k2
            [3]+k2[4])/5.0)
94      print("\n")
```

以下に、ソースコード 5 の出力結果を示す。

```
An= [[1.00176288 0.97149755]
 [0.97149755 1.51083111]]
最急降下法
n= 2 , 1 回目
最適解は、 [ 4.21402106e-06 -2.82533761e-06]
最適値は、 3.358087482359911e-12
k= 42
ニュートン法
n= 2 , 1 回目
最適解は、 [ 8.88178420e-16 -1.33226763e-15]
最適値は、 5.863761728764686e-31
k= 1


An= [[0.24974871 0.43982338]
 [0.43982338 0.81488507]]
最急降下法
n= 2 , 2 回目
最適解は、 [ 1.82386442e-04 -9.96075154e-05]
最適値は、 2.0612956247230683e-10
k= 788
ニュートン法
n= 2 , 2 回目
最適解は、 [ 1.42108547e-14 -3.55271368e-15]
最適値は、 8.155417383093652e-30
k= 1


An= [[1.11476884 0.79135162]
 [0.79135162 0.7691912 ]]
最急降下法
n= 2 , 3 回目
最適解は、 [-8.33406281e-06  1.03652651e-05]
最適値は、 1.1673758799982392e-11
k= 66
ニュートン法
n= 2 , 3 回目
最適解は、 [-8.8817842e-16  8.8817842e-16]
```

```
最適値は、 1.1882484920915162e-31
k= 1


An= [[0.10631884 0.26564457]
 [0.26564457 0.97806447]]
最急降下法
n= 2 , 4 回目
最適解は、 [ 5.95024443e-05 -1.67031973e-05]
最適値は、 6.063242721998152e-11
k= 289
ニュートン法
n= 2 , 4 回目
最適解は、 [0.0000000e+00 4.4408921e-16]
最適値は、 9.644460329222392e-32
k= 1


An= [[0.49505243 0.71471099]
 [0.71471099 1.36475728]]
最急降下法
n= 2 , 5 回目
最適解は、 [ 1.85562356e-05 -1.04308146e-05]
最適値は、 2.1138683552122644e-11
k= 100
ニュートン法
n= 2 , 5 回目
最適解は、 [-1.77635684e-15  8.88178420e-16]
最適値は、 1.917417390528357e-31
k= 1


最急降下法の k の平均= 257.0
ニュートン法の k の平均= 1.0


An= [[1.8382513  1.9297273  1.22385197 1.27872924 1.46560208]
 [1.9297273  2.22387437 1.37685357 1.51248732 1.62541566]
 [1.22385197 1.37685357 1.29713904 0.95772674 1.22634857]
```

```
 [1.27872924 1.51248732 0.95772674 1.10009478 1.20012751]
 [1.46560208 1.62541566 1.22634857 1.20012751 1.44769992]]
最急降下法
n= 5 , 1 回目
最適解は、 [ 0.00156929 -0.00279063  0.00103751  0.00404534 -0.00268806]
最適値は、 1.087530312281795e-08
k= 19164
ニュートン法
n= 5 , 1 回目
最適解は、 [ 0.00000000e+00  0.00000000e+00  2.27373675e-13 -9.09494702e-13
  0.00000000e+00]
最適値は、 2.9046554704896126e-25
k= 1


An= [[1.85806179 2.16072401 0.9339079  1.51145354 1.15063418]
 [2.16072401 3.36037293 1.80651765 2.41627764 1.74345851]
 [0.9339079  1.80651765 1.40233668 1.27167297 0.95358935]
 [1.51145354 2.41627764 1.27167297 1.80893756 1.30868307]
 [1.15063418 1.74345851 0.95358935 1.30868307 1.01211368]]
最急降下法
n= 5 , 2 回目
最適解は、 [ 7.58657319e-05 -1.94033817e-04  6.96134011e-05  2.82302645e-04
 -1.85132649e-04]
最適値は、 9.416259428802308e-10
k= 1547
ニュートン法
n= 5 , 2 回目
最適解は、 [ 7.10542736e-14  0.00000000e+00  1.42108547e-14  5.68434189e-14
 -5.68434189e-14]
最適値は、 7.818367915836031e-27
k= 1


An= [[1.65630005 1.23813241 1.59456741 2.1451727  0.59396582]
 [1.23813241 1.14871233 1.25582226 1.49814851 0.25834481]
 [1.59456741 1.25582226 1.74992261 2.20861559 0.47947426]
 [2.1451727  1.49814851 2.20861559 3.15866947 0.81330651]
 [0.59396582 0.25834481 0.47947426 0.81330651 0.45110303]]
```

```
最急降下法
n= 5 , 3 回目
最適解は、 [-1.40687842e-04  9.92313692e-05 -3.18833791e-06  2.93410181e-05
  8.10689277e-05]
最適値は、 3.4564052753101734e-10
k= 1413
ニュートン法
n= 5 , 3 回目
最適解は、 [-2.84217094e-14  3.55271368e-15  1.06581410e-14  0.00000000e+00
 -3.55271368e-15]
最適値は、 2.565238051972829e-28
k= 1


An= [[2.0059897  1.07680471 1.29965628 1.57936051 1.16006661]
 [1.07680471 0.92722422 0.77673743 0.8351493  0.62580667]
 [1.29965628 0.77673743 1.75507584 0.96641344 0.94048418]
 [1.57936051 0.8351493  0.96641344 1.79857114 1.01347412]
 [1.16006661 0.62580667 0.94048418 1.01347412 0.73892027]]
最急降下法
n= 5 , 4 回目
最適解は、 [-2.22819173e-04  2.77801769e-05 -1.68057227e-04 -1.52435511e-04
  7.52959632e-04]
最適値は、 1.668816516654587e-09
k= 2725
ニュートン法
n= 5 , 4 回目
最適解は、 [-1.42108547e-14  1.77635684e-15 -2.84217094e-14 -4.26325641e-14
  5.68434189e-14]
最適値は、 1.4149121373860123e-27
k= 1


An= [[1.59535581 1.25717835 1.76924919 1.12319172 0.85919331]
 [1.25717835 1.84306561 2.02665465 0.6904924  1.17189631]
 [1.76924919 2.02665465 2.76255431 1.16483202 1.7450155 ]
 [1.12319172 0.6904924  1.16483202 0.94903019 0.53170056]
 [0.85919331 1.17189631 1.7450155  0.53170056 1.37500396]]
最急降下法
```

```
n= 5 , 5 回目
最適解は、 [ 2.04495511e-05  7.12693931e-05 -1.59560596e-04  6.05051324e-05
  1.05971661e-04]
最適値は、 3.8875170927579975e-10
k= 1493
ニュートン法
n= 5 , 5 回目
最適解は、 [-4.26325641e-14 -1.42108547e-14  0.00000000e+00 -7.10542736e-15
  0.00000000e+00]
最適値は、 2.831484377335632e-27
k= 1


最急降下法の k の平均= 5268.4
ニュートン法の k の平均= 1.0


An= [[3.81800629 2.82093079 2.35799468 3.19544732 3.03595101 3.78316792
  2.10888167 2.23045935 2.84092023 4.14042838]
 [2.82093079 2.88296654 2.61537621 2.59383671 2.64480045 3.10924293
  1.66391835 1.66556052 2.11646605 3.7311502 ]
 [2.35799468 2.61537621 3.09857322 2.46618528 2.78232989 3.17639016
  1.80797642 1.62199145 2.17969268 3.59682983]
 [3.19544732 2.59383671 2.46618528 3.46564394 2.97362081 3.32467159
  1.86230511 2.31183103 2.9465389  3.68368099]
 [3.03595101 2.64480045 2.78232989 2.97362081 4.08797057 3.43427845
  2.29784499 2.07836234 2.84616483 3.47846568]
 [3.78316792 3.10924293 3.17639016 3.32467159 3.43427845 4.83620349
  2.18911652 2.38898474 2.71391433 4.43679938]
 [2.10888167 1.66391835 1.80797642 1.86230511 2.29784499 2.18911652
  1.81125119 1.19431252 1.90178283 2.4835558 ]
 [2.23045935 1.66556052 1.62199145 2.31183103 2.07836234 2.38898474
  1.19431252 1.72097899 2.07672965 2.48279888]
 [2.84092023 2.11646605 2.17969268 2.9465389  2.84616483 2.71391433
  1.90178283 2.07672965 3.11066603 3.10512454]
 [4.14042838 3.7311502  3.59682983 3.68368099 3.47846568 4.43679938
  2.4835558  2.48279888 3.10512454 5.71774259]]
最急降下法
n= 10 , 1 回目
```

```
最適解は、 [ 1.35213494e-04 -1.90415163e-05  1.29637201e-04 -4.20214049e-05
 -3.10824202e-05 -9.82539712e-05  4.30797355e-05  1.67243392e-04
 -1.09470491e-04 -7.74924914e-05]
最適値は、 1.0693476832223853e-09
k= 3573
ニュートン法
n= 10 , 1 回目
最適解は、 [-6.25277607e-13  5.82645043e-13 -5.68434189e-13 -7.10542736e-14
 -1.42108547e-14  3.41060513e-13  1.42108547e-13  1.13686838e-13
  1.13686838e-13  1.13686838e-13]
最適値は、 4.5211644885114845e-26
k= 1


An= [[3.61006805 2.97319346 2.55951186 1.98539718 3.44478007 2.85444311
  2.97349581 2.41764729 3.48971486 2.3147884 ]
 [2.97319346 2.70155904 2.18320146 1.84081372 2.78033672 2.29371715
  2.57789564 1.98333077 2.82657925 1.70422506]
 [2.55951186 2.18320146 2.18142589 1.51695778 2.57798749 1.70771832
  2.30651924 1.55932879 2.72755164 1.66206029]
 [1.98539718 1.84081372 1.51695778 2.02592525 2.18330497 2.11150036
  2.39311495 2.00063583 2.47934741 1.77075286]
 [3.44478007 2.78033672 2.57798749 2.18330497 4.39155634 3.23353296
  3.28355584 2.57816725 4.19456245 3.02921007]
 [2.85444311 2.29371715 1.70771832 2.11150036 3.23353296 3.43681962
  2.83266499 2.61571913 3.23603818 2.45888123]
 [2.97349581 2.57789564 2.30651924 2.39311495 3.28355584 2.83266499
  3.64416163 2.51421963 3.73390786 2.59043527]
 [2.41764729 1.98333077 1.55932879 2.00063583 2.57816725 2.61571913
  2.51421963 2.77970058 2.68215015 2.39152052]
 [3.48971486 2.82657925 2.72755164 2.47934741 4.19456245 3.23603818
  3.73390786 2.68215015 4.52996891 3.16941543]
 [2.3147884  1.70422506 1.66206029 1.77075286 3.02921007 2.45888123
  2.59043527 2.39152052 3.16941543 3.24358759]]
最急降下法
n= 10 , 2 回目
最適解は、 [-1.70966394e-04  1.17188844e-04  9.58681701e-05 -9.86243738e-05
 -5.42936225e-05  6.94454756e-05 -3.05648819e-05  3.63659298e-05
  5.38875765e-05  8.79210659e-06]
```

```
最適値は、 1.0307657020984702e-09
k= 2608
ニュートン法
n= 10 , 2 回目
最適解は、 [-9.94759830e-14 -2.84217094e-14  5.68434189e-14 -9.94759830e-14
  4.97379915e-14  3.55271368e-14  0.00000000e+00 -7.10542736e-15
  0.00000000e+00  1.06581410e-14]
最適値は、 1.1680865463538095e-26
k= 1


An= [[3.57489974 2.40976335 1.92796498 3.04911337 3.59131693 2.4075717
  3.29069835 2.81678747 2.20647417 2.78989116]
 [2.40976335 2.85933885 1.37072026 1.95198508 3.11996368 2.35544034
  1.95200549 1.90595234 2.37809853 2.43211587]
 [1.92796498 1.37072026 1.69813612 2.1853596  2.32398036 2.15872771
  2.17533453 1.75355333 1.47006286 2.06283999]
 [3.04911337 1.95198508 2.1853596  3.97011701 3.08951718 3.11455811
  3.25263995 3.29277215 2.04878533 2.47169046]
 [3.59131693 3.11996368 2.32398036 3.08951718 5.20412699 3.24820842
  4.18888433 3.27030392 2.75051174 3.70275854]
 [2.4075717  2.35544034 2.15872771 3.11455811 3.24820842 3.34901184
  2.69360464 2.56540306 2.23620169 2.8856916 ]
 [3.29069835 1.95200549 2.17533453 3.25263995 4.18888433 2.69360464
  4.51529895 3.35501599 2.18233603 2.99170553]
 [2.81678747 1.90595234 1.75355333 3.29277215 3.27030392 2.56540306
  3.35501599 3.75341706 1.58378697 2.21227152]
 [2.20647417 2.37809853 1.47006286 2.04878533 2.75051174 2.23620169
  2.18233603 1.58378697 2.62585403 2.0746262 ]
 [2.78989116 2.43211587 2.06283999 2.47169046 3.70275854 2.8856916
  2.99170553 2.21227152 2.0746262  3.46023701]]
最急降下法
n= 10 , 3 回目
最適解は、 [-0.02711453  0.02607624  0.02258709  0.02330028 -0.00235783 -0.03863167
  0.00467064 -0.00260153 -0.00603347  0.00940671]
最適値は、 2.455039787394417e-07
k= 85077
ニュートン法
n= 10 , 3 回目
```

```
最適解は、 [ 1.45519152e-11 -2.18278728e-11 -1.45519152e-11 -1.45519152e-11
  9.09494702e-13  2.91038305e-11 -3.63797881e-12  2.72848411e-12
  5.45696821e-12 -7.27595761e-12]
最適値は、 3.907983608748827e-23
k= 1


An= [[2.50768394 1.99365011 1.83192989 1.58191809 2.57135898 2.30594452
  1.53307649 1.13613128 2.2034903  2.24352749]
 [1.99365011 2.13096623 1.49045151 1.69832639 2.00694647 1.95398178
  1.389077   1.20892107 2.23407699 2.29284662]
 [1.83192989 1.49045151 2.46354093 1.53660763 1.76692185 2.31590944
  1.3245296  1.23065006 1.66887121 1.41192873]
 [1.58191809 1.69832639 1.53660763 2.38395925 1.65515031 2.24519304
  1.57904501 1.57896275 2.33584347 1.98469923]
 [2.57135898 2.00694647 1.76692185 1.65515031 4.15230628 2.60154839
  2.31918804 1.0905688  3.19326128 2.4300079 ]
 [2.30594452 1.95398178 2.31590944 2.24519304 2.60154839 3.42966511
  1.49297996 1.78854884 2.58900107 1.94517046]
 [1.53307649 1.389077   1.3245296  1.57904501 2.31918804 1.49297996
  2.93312182 1.06275234 2.11804233 1.75880979]
 [1.13613128 1.20892107 1.23065006 1.57896275 1.0905688  1.78854884
  1.06275234 1.21899948 1.60737    1.29156426]
 [2.2034903  2.23407699 1.66887121 2.33584347 3.19326128 2.58900107
  2.11804233 1.60737    3.64665796 2.91116647]
 [2.24352749 2.29284662 1.41192873 1.98469923 2.4300079  1.94517046
  1.75880979 1.29156426 2.91116647 2.93063599]]
最急降下法
n= 10 , 4 回目
最適解は、 [-0.0543657  -0.0368175   0.01006552 -0.01886172  0.06302661 -0.01734899
 -0.0185595   0.1086839  -0.08594407  0.0862151 ]
最適値は、 2.8083818801718938e-08
k= 1585
ニュートン法
n= 10 , 4 回目
最適解は、 [ 2.32830644e-10 -1.86264515e-09  2.91038305e-10 -3.49245965e-10
  4.65661287e-10 -2.32830644e-10 -1.16415322e-10  1.16415322e-09
 -2.32830644e-10  2.32830644e-10]
最適値は、 1.1514080377078483e-18
```

```
k= 1


An= [[2.08537147 2.30426358 1.45039701 1.18612989 2.64023489 1.24869001
  2.24995765 2.23395089 2.13831282 2.18539168]
 [2.30426358 3.81596781 1.79239002 2.40275958 3.56349738 2.08708698
  2.22820916 3.48272636 3.18009626 3.23043674]
 [1.45039701 1.79239002 2.44232927 1.65384542 2.35184286 1.43332566
  1.70296579 1.74318269 1.6133816  2.28014413]
 [1.18612989 2.40275958 1.65384542 2.33148098 2.21537991 1.52566516
  1.21402713 2.44258827 1.84741269 2.55885655]
 [2.64023489 3.56349738 2.35184286 2.21537991 4.26037592 2.12834842
  3.01831603 3.38044995 3.10535285 3.18072938]
 [1.24869001 2.08708698 1.43332566 1.52566516 2.12834842 2.47235796
  1.23064975 2.33463954 1.61199763 2.40026432]
 [2.24995765 2.22820916 1.70296579 1.21402713 3.01831603 1.23064975
  2.98789045 2.36185187 2.11053484 2.38391066]
 [2.23395089 3.48272636 1.74318269 2.44258827 3.38044995 2.33463954
  2.36185187 3.83635397 2.62180775 3.60730581]
 [2.13831282 3.18009626 1.6133816  1.84741269 3.10535285 1.61199763
  2.11053484 2.62180775 3.12825978 2.62273915]
 [2.18539168 3.23043674 2.28014413 2.55885655 3.18072938 2.40026432
  2.38391066 3.60730581 2.62273915 3.99038292]]
最急降下法
n= 10 , 5 回目
最適解は、 [-9.49547104e-05 -1.34480762e-04  1.16852784e-04 -6.49770554e-05
 -6.82513542e-05 -2.70412136e-05  2.99503195e-06  2.67423108e-04
  1.51471361e-04 -1.36764523e-04]
最適値は、 1.5915862192812102e-09
k= 2384
ニュートン法
n= 10 , 5 回目
最適解は、 [-1.13686838e-13 -5.68434189e-14  1.13686838e-13  0.00000000e+00
  5.68434189e-14  3.55271368e-14  1.42108547e-14  1.13686838e-13
  0.00000000e+00 -5.68434189e-14]
最適値は、 2.3502123319835773e-26
k= 1
```

最急降下法の k の平均= 19045.4
ニュートン法の k の平均= 1.0

9 参考文献

- 数理工学実験テキスト
- Cloud LaTeX
  https://cloudlatex.io
- LaTeX コマンド一覧（リスト）
  https://medemanabu.net/latex/latex-commands-list/
- LaTeX にソースコードを【美しく】貼る方法
  https://ta-b0.hateblo.jp/entry/2020/08/13/001223
- コメント（注釈） - 複数行のコメントアウト
  https://medemanabu.net/latex/comment/
- 【LaTeX】箇条書きの方法ついて徹底解説
  https://mathlandscape.com/latex-enum/
- TeX で複素数 C、実数 R、有理数 Q、整数 Z、自然数 N などの白抜き文字を使う方法
  http://did2.blog64.fc2.com/blog-entry-204.html
- Anaconda（Python3）インストール手順＜ macOS 用＞
  https://sukkiri.jp/technologies/ides/anaconda-mac_install.html
- Python のコードの書き方の基本
  https://kitsuneblg.com/2018/11/03/python のコードの書き方その 1 /
- あなたのデータサイエンス力を飛躍的に向上させる NumPy 徹底入門
  https://deepage.net/features/numpy/
- LaTeX で URL を入力する方法
  https://qiita.com/hiroooooooaki/items/84d45c83c6bcb1d52b79
- 行列と行列式 - matri, pmatrix, bmatrix
  https://medemanabu.net/latex/matrix-pmatrix-bmatrix/