レポート提出票

科目名:	情報工学実験2
実験テーマ:	実験テーマ1 数理計画法
実施日:	2020年 10月 26日
学籍番号:	4619055
氏名:	辰川力駆
共同実験者:	

1 実験の要旨

実験を通して、最急降下法とニュートン法についての理解を深めるとともに、二次元平面に プロットするなどをして議論をする。

2 実験の目的

非線形最適化問題に対し、最急降下法とニュートン法を実装・適用を通し、アルゴリズムの 特性を理解する。

3 実験の原理(理論)

3.1 無制約最小化問題に対する基礎理論

無制約最小化問題とは、n変数関数 $f: \mathbb{R}^n \to \mathbb{R}$ に対して定義される以下の問題である。

Minimize
$$f(\boldsymbol{x})$$
 subject to $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$

3.1.1 諸定義

n変数関数 $f: \mathbb{R}^n \to \mathbb{R}$ に対し、勾配ベクトル $\nabla f(x)$ とヘッセ行列 $\nabla^2 f(x)$ はそれぞれ以下のように定義されるベクトルと行列である。

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \ \nabla^2 f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix}$$

勾配ベクトルとヘッセ行列はそれぞれ 1 変数関数 f(x) の微分係数 f'(x) と 2 階微分係数 f''(x) を n 変数関数に拡張したものである。ヘッセ行列は対称行列になることに注意する。勾配ベクトルとヘッセ行列は f のテイラー展開に現れる。具体的には、x=a の周りで 2 次の項まで求めると、以下のようになる。

$$f(\boldsymbol{a} + \boldsymbol{d}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \nabla^2 f(\boldsymbol{a}) \boldsymbol{d} + (\mathbf{E} \boldsymbol{E})$$

また、n 次実正方行列 X に対し、

- X が半正定値とは、 $X^T = X$ と、 $\forall d \in \mathbb{R}^n$, $d^T X d \geq 0$ が成り立つことを言い、
- X が正定値とは、 $X^T = X$ と、 $\forall d \in \mathbb{R}^n \setminus \{0\}, d^T X d > 0$ が成り立つことを言う。

さらに、n 次実対称行列 X に対し、「X が半正定値 (正定値) $\Leftrightarrow X$ の固有値がすべて正」が成り立つ。

3.1.2 最適性条件

一般に非線形最適化問題において大域的最適解を求めることは極めて難しく、多くの場合は局所的最適解を求めることを目指す。局所最適解については以下の最適性条件が成り立つ。

定理 (最適性の必要条件) $x^* \in \mathbb{R}^n$ を無制約最小化問題の局所的最適解とする。このとき、つぎが成り立つ。

- f が連続的微分可能 $\Rightarrow \nabla f(x^*) = \mathbf{0}$
- f が 2 回連続的微分可能 $\Rightarrow \nabla^2 f(x^*)$ は半正定値行列

1次の必要条件 $\nabla f(x^*) = \mathbf{0}$ を満たす x^* を停留点という。

定理 (2 次の十分条件) 関数 f が 2 回連続的微分可能とする。 $x^* \in \mathbb{R}^n$ が以下の条件を満たすならば、 $x^* \in \mathbb{R}^n$ は無制約最小化問題の局所的最適解である。

- $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$
- $\nabla^2 f(\boldsymbol{x}^*)$ が正定値行列

関数が凸性と呼ばれる「良い」性質を持つ場合には、実は、大域的最適解を求めることが 比較的容易になる。関数 $f: \mathbb{R}^n \to \mathbb{R}$ が以下の条件を満たすとき、f は凸関数という。

$$f(\lambda \boldsymbol{x} + (1 - \lambda)\boldsymbol{y}) \le \lambda f(\boldsymbol{x}) + (1 - \lambda)f(\boldsymbol{y}), \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \ \forall \lambda \in (0, 1)$$

fが2回連続的微分可能であるとき、以下の事実が成り立つ。

$$f$$
 が凸関数 $\Leftrightarrow \nabla^2 f(x)$ が半正定値行列 $\forall x \in \mathbb{R}^n$

さらに、凸関数は次の「局所最適性=大域的最適性」を満たす。

f が凸関数のとき、 x^* が f の局所的最適解 $\Rightarrow x^*$ は f の大域的最適解

これにより、微分可能な凸関数に対しては、 $\nabla f(x^*) = \mathbf{0}$ を満たす x^* を求めれば、その x^* は最適解であることが保証される。このように、大域的最適解が必ず求まる関数のクラスとして、凸関数は重要なクラスである。

3.2 反復法

適当な初期点 $x_0 \in \mathbb{R}^n$ からスタートし、以下の更新式で次々と点 x_1, x_2, \cdots ,を生成するアルゴリズムを反復法という。

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

この式における d_k を探索方向、 α_k をステップ幅という。

探索方向としては降下方向になっているものを用いるのが一般的である。具体的には、 $x \in \mathbb{R}^n$ における降下方向とは以下の条件を満たす $d \in \mathbb{R}^n$ であり、目的関数値が減少する方向である。

$$\nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0$$

 $\nabla f(x) \neq \mathbf{0}$ であれば、 $-\nabla f(x)$ は自明な降下方向である。これを最急降下方向という。

また、ステップ幅は次小節で説明する直線探索を利用して求めることが多い。直線探索を用いた反復法を形式的に書くと以下のようになる。

直線探索を用いた反復法

ステップ 0: 初期点 x_0 を選び、k := 0 とする

ステップ1: 停止基準が満たされていれば終了とする

ステップ 2: 探査方向 d_k を定める

ステップ 3: 直線探索を用いてステップ幅 α_k を定める

ステップ 4: $x_{k+1} := x_k + \alpha_k d_k$, k := k+1 としてステップ幅 1 に戻る

ステップ1の停止基準としては、勾配ベクトルの大きさ $||\nabla f(x_k)||$ が十分小さくなったことや、解の更新幅 $||x_{k+1}-x_k||$ が十分小さくなったことなどに設定する。

3.3 直線探索

点 $oldsymbol{x}_k \in \mathbb{R}^n$ と $oldsymbol{x}_k$ における降下方向 $oldsymbol{d}_k$ が与えられたときに可能ならばステップ幅 $lpha_k$ を

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) = \min\{f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) | \alpha > 0\}$$

となるように選びたい。関数 f が凸 2 次関数の場合は正確に α_k が求まるが、そうでなければこの問題は非常に難しい (それ自体が 1 変数の最小化問題)。多くの場合は次のアルミホ基準を満たすように α_k を求める。

定義 (アルミホ基準) $0 < \xi < 1$ を満たす定数 ξ に対し、

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \le f(\boldsymbol{x}_k) + \xi \alpha \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k$$

アルミホ条件に対する直線探索 $(x_k, d_k$ は所与とする)

ステップ 0: パラメータ $0 < \xi < 1, 0 < \tau < 1$ を選び、 $\alpha := 1$ とする

ステップ 1: $f(\mathbf{x}_k + \alpha \mathbf{d}_k) \leq f(\mathbf{x}_k) + \xi \alpha \nabla f(\mathbf{x}_k)^T \mathbf{d}_k$ (アルミホ条件) が成り立てば終了する

ステップ 2: $\alpha := \tau \alpha$ としてステップ 1 に戻る

なお、アルミホ条件に $\nabla f(\mathbf{x}_k + \alpha \mathbf{d}_k)$ に関する条件を付加したウルフ条件も実用的に多く用いられている。

3.4 実験で扱うアルゴリズム

3.4.1 最急降下法

最急降下法とは、反復法におけるステップ 2 の探索方向として、 x_k における最急降下方向 $-\nabla f(x_k)$ を常に用いる方法である。本実験で用いる最急降下法は以下の通りである。

最急降下法

ステップ 0: 初期点 x_k を選び、k := 0 とする。また、 $\epsilon := 10^{-8}$ とする

ステップ 1: $||\nabla f(x_k)|| < \epsilon$ が満たされていれば終了する

ステップ 2: 探索方向を $d_k = -\nabla f(x_k)$ とする

ステップ 3: アルミホ条件による直線探索を用いてステップ幅 α_k を定める

ステップ 4: $x_{k+1} = x_k + \alpha_k d_k$, k := k+1 としてステップ 2 に戻る

最急降下法の大域的収束性 関数 f(x) に関するいくつかの仮定の下ではウルフ条件を用いた最 急降下法は任意の初期点に対して以下の式が成り立つ。

$$\lim_{k \to \infty} ||\nabla f(\boldsymbol{x}_k)|| = 0$$

このように、初期点に依存せずに停留点に収束するのは最急降下法の強みであるが、その 一方で収束スピードが非常に遅い欠点を持つ。

最急降下法の1次収束性 最急降下法で生成される点列 $\{x_k\}$ の収束先を x^* とすると以下の式が成り立つ定数 0 < c < 1 が存在する。

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*|| \le c||\boldsymbol{x}_k - \boldsymbol{x}^*||$$

この式では一見、収束が遅いようには見えないが、 x^* 付近では $||x_k - x^*||$ が非常に微小のため、最急降下法が収束するまでに要する反復数は非常に多い。

3.4.2 ニュートン法

ニュートン法は f(x)2 次の項までのテイラー展開を最小化することを繰り返す方法である。 すなわち、

$$q(\boldsymbol{d}) = f(\boldsymbol{x}_k + \boldsymbol{d}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{d}$$

として、 $\nabla q(\mathbf{d}) = \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k) \mathbf{d} = \mathbf{0}$ を満たす \mathbf{d} を探索方向とする。方向 \mathbf{d} はニュートン方向と呼ばれ、以下のニュートン方程式を解くことで得られる。

$$abla^2 f(\boldsymbol{x}_k) \boldsymbol{d} = -\nabla f(\boldsymbol{x}_k)$$

本実験で用いるニュートン法は以下の通りである。

ニュートン法

ステップ 0: 初期点 x_0 を選び、k=0 とする。 $\epsilon=10^{-8}$ とする

ステップ 1: $||\nabla f(x_k)|| < \epsilon$ が満たされていれば終了する

ステップ 2: 方程式 $\nabla^2 f(\boldsymbol{x}_k) \boldsymbol{d} = -\nabla f(\boldsymbol{x}_k)$ を解いて探索方向 \boldsymbol{d}_k を求める

ステップ 3: $x_{k+1} := x_k + d_k$, k := k+1 としてステップ 1 に戻る

ニュートン法の局所的 2 次収束性 初期点 x_0 を x^* の十分近くに点を取ると、ニュートン法で 生成される点列 $\{x_k\}$ は収束し、その収束先を x^* とすると以下を満たす定数 $c \geq 0$ が存在 する。

$$||x_{k+1} - x^*|| \le c||x_k - x^*||^2$$

1次収束と異なり、上の2次収束は非常に速い。一方で、初期点の取り方が悪ければニュートン法は収束しなかったり、局所最適解でない点に収束することもしばしばある。

4 実験課題

次の各項目に取り組む。

- (1) 実験目的とアルゴリズムに関する数学的理論について概要を述べる。
- (2) 以下の2種類の2変数関数に対して最急降下法を実行するプログラムを完成させる。

$$f_{1}(x_{1}, x_{2}) = \frac{1}{2} \begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix}^{T} \begin{pmatrix} p & q \\ q & r \end{pmatrix} \begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} + \begin{pmatrix} c_{1} \\ c_{2} \end{pmatrix}^{T} \begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} + e^{(x_{1} - x_{2})^{2}}$$

$$f_{2}(x_{1}, x_{2}) = \sum_{i=0}^{4} \sum_{j=0}^{2} a_{ij} x_{1}^{i} x_{2}^{j}$$

- (3) (2) で作成したプログラムを動かしてふるまいを観察し、性能を考察する。
- (4) 関数 f_1 と関数 f_2 に対してニュートン法を実行するプログラムを完成させる。
- (5) (4) で作成したプログラムを動かしてふるまいを観察し、性能を考察する。

5 実験結果・検討・考察

(1)

上記(目的、原理)で述べた。

(2)

それぞれの関数に対して作成したソースコード 2 つを付録に載せた。 f_1 のソースコードも f_2 のソースコードも仕組みは同じなのでソースコード $1(4619055_辰川力駆_SD_1.c)$ に対しての説明をする。

まず、27 行目から 48 行目でファイルを読み取り、最急降下法を使う行列を受け取っている。 そして、確認のために 50 行目から 71 行目で出力している。次に、メインの最急降下法 (77 行目から 114 行目) について説明する。79 行目から 83 行目までは $\nabla f(x_k)$ と d_k を求めている。その後、ノルム、関数値を求めた後に、アルミホ条件による直線探索を用いてステップ幅を定めている。アルミホ条件によるステップ幅の決定は 116 行目から 142 行目の関数で求めている。

最後に x_k を更新したあと、101 行目から 108 行目で終了条件が満たされていれば終了する。満たされていなければさらに実行というようにしている。 $||\nabla f(x_k)|| < \epsilon$ という条件だけでは、最急降下法はたくさんの回数を要するため時間がかかってしまう可能性があるので反復回数が1000 回を越えたら停止するように工夫している。

出力部分に関しては、反復回数、現在位置、目的関数、勾配ベクトル、ノルムを表示するようにした。表示する頻度は多すぎるといけないので、0.1,2,3,4,5,10,20,30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000とするように工夫をした。

(3)

学籍番号は 4619055 なので seed 値を 55 としてデータを発生させた。実験に用いるデータは下記である。なお、図 2 の一番右のデータは data55.txt のデータである。

```
data set 1
matrix Q
    6.0
           -5.0
   -5.0
           13.0
vector c
   -5.0
            1.0
three initial solutions
            0.0
    9.0
  -13.0
            1.0
    8.0
           -1.0
```

```
data set 2
matrix Q
   10.0
           6.0
    6.0
           7.0
vector c
   -7.0
          -8.0
three initial solutions
          -1.0
   10.0
  -11.0
          -3.0
    1.0
          -6.0
```

```
data set 3
matrix Q
    2.0
           1.0
    1.0
           9.0
vector c
   -7.0
          -6.0
three initial solutions
   -8.0
          -3.0
   -5.0
          -4.0
   1.0
          -7.0
```

図 1: f₁で用いる実験データ

```
data set 1
coefficient matrix A
    0.0 -27.0
                  1.0
   32.0 -11.0
                  0.0
           0.0
    4.0
                  0.0
   -3.0
           0.0
                  0.0
   1.0
           0.0
                  0.0
three initial solutions
  -17.0 -36.0
  -33.0 -33.0
  27.0
          11.0
```

```
data set 2
coefficient matrix A
    0.0 -33.0
                  1.0
 -15.0 -31.0
                  0.0
   -8.0
           0.0
                  0.0
   9.0
           0.0
                  0.0
    1.0
           0.0
                  0.0
three initial solutions
 -24.0 -33.0
   36.0 -15.0
  25.0 -13.0
```

```
coefficient matrix A
    0.0
           2.0
                   1.0
   -4.0
          14.0
                   0.0
    4.0
           0.0
                   0.0
    7.0
           0.0
                   0.0
           0.0
    1.0
                   0.0
three initial solutions A
  -41.0
          32.0
   -3.0 -50.0
   31.0 -21.0
```

図 2: f₂で用いる実験データ

実行結果は以下のようになった。

```
data set 1
coefficient matrix A
    0.0 -27.0
                  1.0
        -11.0
   32.0
                  0.0
           0.0
   4.0
                  0.0
   -3.0
           0.0
                  0.0
   1.0
           0.0
                  0.0
three initial solutions
  -17.0 -36.0
  -33.0 -33.0
   27.0 11.0
```

```
data set 2
coefficient matrix A
    0.0 -33.0
                  1.0
 -15.0 -31.0
                  0.0
           0.0
   -8.0
                  0.0
    9.0
           0.0
                  0.0
   1.0
           0.0
                  0.0
three initial solutions
 -24.0 -33.0
   36.0 -15.0
  25.0 -13.0
```

```
coefficient matrix A
    0.0
           2.0
                   1.0
   -4.0
          14.0
                  0.0
    4.0
           0.0
                  0.0
    7.0
           0.0
                   0.0
    1.0
           0.0
                   0.0
three initial solutions A
  -41.0
          32.0
   -3.0 -50.0
   31.0 -21.0
```

図 3: f_1 , data set 1, \boldsymbol{x}_0 の値 1 行目の結果

```
data set 1
coefficient matrix A
    0.0 -27.0
                   1.0
   32.0
        -11.0
                   0.0
    4.0
           0.0
                   0.0
   -3.0
           0.0
                   0.0
    1.0
           0.0
                   0.0
three initial solutions
  -17.0 -36.0
  -33.0 -33.0
   27.0
          11.0
```

```
data set 2
coefficient matrix A
   0.0 -33.0
         -31.0
  -15.0
                  0.0
   -8.0
           0.0
                  0.0
   9.0
           0.0
                  0.0
    1.0
           0.0
                  0.0
three initial solutions
  -24.0 -33.0
   36.0 -15.0
   25.0 -13.0
```

```
coefficient matrix A
    0.0
           2.0
                   1.0
                   0.0
   -4.0
          14.0
    4.0
           0.0
                  0.0
    7.0
           0.0
                   0.0
    1.0
           0.0
                   0.0
three initial solutions A
          32.0
  -41.0
   -3.0 -50.0
   31.0 -21.0
```

図 4: f_1 , data set 1, \boldsymbol{x}_0 の値 2 行目の結果

```
data set 1
coefficient matrix A
    0.0
         -27.0
   32.0
         -11.0
                   0.0
    4.0
           0.0
                   0.0
   -3.0
           0.0
                   0.0
    1.0
           0.0
                   0.0
three initial solutions
  -17.0 -36.0
  -33.0 -33.0
   27.0
          11.0
```

```
data set 2
coefficient matrix A
   0.0 -33.0
                  1.0
 -15.0 -31.0
                  0.0
   -8.0
           0.0
                  0.0
   9.0
           0.0
                  0.0
   1.0
           0.0
                  0.0
three initial solutions
  -24.0 -33.0
   36.0 -15.0
  25.0 -13.0
```

```
coefficient matrix A
    0.0
           2.0
                   1.0
   -4.0
          14.0
                   0.0
   4.0
           0.0
                  0.0
    7.0
           0.0
                   0.0
    1.0
           0.0
                   0.0
three initial solutions A
  -41.0
          32.0
   -3.0 -50.0
   31.0 -21.0
```

図 5: f_1 , data set 1, \boldsymbol{x}_0 の値 3 行目の結果

(4)

それぞれの関数に対して作成したソースコード 2 つを付録に載せた。 f_1 のソースコードも f_2 のソースコードも仕組みは同じなのでソースコード $3(4619055_辰川力駆_newton_1.c)$ に対しての説明をする。

まず、29 行目から 50 行目でファイルを読み取り、ニュートン法を使う行列を受け取っている。 そして、確認のために 52 行目から 73 行目で出力している。次に、メインの最急降下法 (79 行目から 111 行目) について説明する。81,82 行目では $\nabla f(x_k)$ を求めている。その後、ノルム、探索方向 d_k 、関数値を求めている。探索方向はニュートン法を使っている。ニュートン法は関数化している。ニュートン法では、まずヘッセ行列 $\nabla^2 f(x_k)$ を求めて、正則でなければ、 $d=\inf$ としている。正則であればヘッセ行列の逆行列を求めて、dを計算している。

最後に x_k を更新したあと、98 行目から 110 行目で終了条件が満たされていれば終了する。満たされていなければさらに実行というようにしている。 $||\nabla f(x_k)|| < \epsilon$ という停止条件と反復回数が 500 回を超えるという停止条件以外に $\nabla^2 f(x_k)$ が非正則なら停止するようにしている。

出力部分に関しては最急降下法と同様に、反復回数、現在位置、目的関数、勾配ベクトル、ノルムを表示するようにした。表示する頻度は多すぎるといけないので、0,1,2,3,4,5,10,20,30,40,50,60,70, 80,90,100,200,300,400,500,600,700,800,900,1000 とするように工夫をした。

(5)

ニュートン法は非正則になったら終了しないといけないのでつらい

6 まとめ

最急降下法とニュートン法をC言語で実装した。また、それらの適用を通し、アルゴリズムの特性を理解することができた。

参考文献

- [1] 東京理科大学工学部情報工学科 情報工学実験 2 2020 年度東京理科大学工学部情報工学科 出版
- [2] 関数グラフ GeoGebra

https://www.geogebra.org/graphing?lang=ja

最終閲覧日 2020/10/29

A 付録

ソースコード 1: 4619055_辰川力駆_SD_1.c

```
//4619055 辰川力駆
1
       #include <stdio.h>
2
       #include <stdlib.h>
3
       #include <string.h>
4
       #include <math.h>
5
       #define N 1000
6
       #define res 0.00000001 ///停止条件
7
8
       double func(double *x); ///f1 の式を用いている
9
       double Armijo(double *x, double *d, double *nabla); //Armijo
10
11
       double Q[2][2]; //Q のデータを保存
12
       double c[2]; //c のデータを保存
13
14
       int main()
15
16
           char fname[128];
17
           double x[2];
18
           double p, q, r;
19
           double d[2], norm;
20
           double nabla[2];
21
           double f;
22
           FILE *fp;
23
           int i, j, k;
24
           double alpha;
25
26
           printf("input_filename:");
27
           fgets(fname, sizeof(fname), stdin);
28
           fname[strlen(fname) -1] = '\0';
29
           fflush(stdin);
30
           fp = fopen(fname, "r");
31
32
           for (i = 0; i < 2; i++)
33
34
               for (j = 0; j < 2; j++)
35
36
                   fscanf(fp, "%lf", &Q[i][j]);
37
38
39
           for (i = 0; i < 2; i++)
40
41
               fscanf(fp, "%lf", &c[i]);
42
43
           for (i = 0; i < 2; i++)
44
45
               fscanf(fp, "%lf", &x[i]);
46
47
           fclose(fp);
48
```

```
49
                                                      printf("Q=\n");
50
                                                      for (i = 0; i < 2; i++)
51
52
                                                                         for (j = 0; j < 2; j++)
53
                                                                         {
54
                                                                                           printf("%4.lf", Q[i][j]);
55
56
                                                                         printf("\n");
57
58
                                                      printf("C=\n");
59
                                                      for (i = 0; i < 2; i++)
60
61
                                                                         printf("%4.lf", c[i]);
62
63
                                                      printf("\n");
64
                                                      printf("x=\n");
65
                                                      for (i = 0; i < 2; i++)
66
67
                                                                         printf("%4.lf", x[i]);
68
69
                                                      printf("\n");
70
                                                      printf("----\n");
71
72
                                                      p = Q[0][0];
73
                                                      q = Q[0][1];
74
                                                      r = Q[1][1];
75
                                                      k = 0;
76
                                                      while (1)
77
78
                                                      {
                                                                         nabla[0] = p * x[0] + q * x[1] + c[0] + 2 * (x[0] - x[1]) * exp((x[0] - x[1]) * (x[0] - x[1]
79
                                                                                             [0] - x[1]);
                                                                         nabla[1] = q * x[0] + r * x[1] + c[1] - 2 * (x[0] - x[1]) * exp((x[0] - x[1]) * (x[0] - x[1]
80
                                                                                                -x[1]);
81
                                                                        d[0] = -nabla[0];
82
                                                                        d[1] = -nabla[1];
83
84
                                                                        norm = sqrt(nabla[0] * nabla[0] + nabla[1] * nabla[1]); ///ノルムを計算
85
86
                                                                         f = func(x);
87
                                                                         alpha = Armijo(x, d, nabla);
88
89
                                                                        if (k <= 5 || (k <= 100 && k % 10 == 0) || k % 100 == 0) //5回目までと 10
90
                                                                                           n回目,100n回目(n=1,2,3,...10)を表示
                                                                         {
91
                                                                                           printf("%d \Box \exists \n", k);
92
                                                                                            printf("現在位置=(%.81f,%.81f)\n 目的関数=%.81f\n 勾配ベクトル=(%.81f
93
                                                                                                                ,\%.81f)\n / \nu = \%.81f\n'', x[0], x[1], f, nabla[0], nabla[1], norm);
                                                                                           printf("----\n");
94
                                                                         }
95
96
```

```
x[0] = x[0] + alpha * d[0];
97
               x[1] = x[1] + alpha * d[1];
98
               k++;
99
100
               if (norm < res) ///nabla のノルムが 10^{-8}を越えていたら停止
101
               {
102
103
                   break;
104
               if (k > N) ///反復回数が 1000回越えたら停止
105
               {
106
                   break;
107
108
               }
           }
109
110
           printf("--結果--\n");
111
           printf("%d回\n現在位置=(%.81f,%.81f)\n目的関数=%.81f\n勾配ベクトル=(%.81f
112
                ,%.81f)\n ノルム=%.81f\n", k - 1, x[0], x[1], f, nabla[0], nabla[1], norm);
           return 0;
113
       }
114
115
       double Armijo(double *x, double *d, double *nabla)
116
       {
117
           double alpha = 1.0;
118
           double xi = 0.1; ///今回の課題条件
119
           double tau = 0.5; ///今回の課題条件
120
           double fx, fy;
121
           double y[2];
122
123
           ///f(x_k)を求める
124
           fx = func(x);
125
126
           while (1)
127
128
               ///新しいalpha に対しての fy を求める
129
               y[0] = x[0] + alpha * d[0];
130
               y[1] = x[1] + alpha * d[1];
131
               fy = func(y);
132
133
               if (fy \le fx + xi * (nabla[0] * d[0] + nabla[1] * d[1]) * alpha) //Armijo 条件
134
               {
135
                   break;
136
137
138
               alpha = tau * alpha; //alpha の更新
139
140
           return alpha;
141
       }
142
143
       // f1の式を用いている
144
       double func(double *x)
145
146
           double p, q, r;
147
```

```
148 p = Q[0][0];

149 q = Q[0][1];

150 r = Q[1][1];

151 \mathbf{return} \ 0.5 * (p * x[0] * x[0] + 2 * q * x[0] * x[1] + r * x[1] * x[1]) + c[0] * x[0] + c

[1] * x[1] + exp((x[0] - x[1]) * (x[0] - x[1]));
```

ソースコード 2: 4619055_辰川力駆_SD_2.c

```
//4619055 辰川力駆
1
2
       #include <stdio.h>
       #include <stdlib.h>
3
       #include <string.h>
4
       #include <math.h>
5
       #define N 1000
6
       #define res 0.00000001 ///停止条件
7
8
       double func(double *x); ///f2 の式を用いている
9
       double Armijo(double *x, double *d, double *nabla); //Armijo
10
11
       double A[5][3]; //A のデータを保存
12
13
       int main()
14
       {
15
           char fname[128];
16
           double x[2];
17
           double d[2], norm;
18
           double nabla[2];
19
           double f;
20
21
           FILE *fp;
           int i, j, k;
22
           double alpha;
23
24
           printf("input_filename:");
25
           fgets(fname, sizeof(fname), stdin);
26
           fname[strlen(fname) -1] = '\0';
27
28
           fflush(stdin);
           fp = fopen(fname, "r");
29
30
           for (i = 0; i < 5; i++)
31
32
               for (j = 0; j < 3; j++)
33
34
                   fscanf(fp, "%lf", &A[i][j]);
35
36
37
           for (i = 0; i < 2; i++)
38
39
               fscanf(fp, "%lf", &x[i]);
40
41
           fclose(fp);
42
43
44
           printf("A=\n");
```

```
for (i = 0; i < 5; i++)
45
                               {
46
                                          for (j = 0; j < 3; j++)
47
48
                                                     printf("%4.lf", A[i][j]);
49
50
                                          \operatorname{printf}("\n");
51
                               }
52
                               printf("x=\n");
53
                               for (i = 0; i < 2; i++)
54
                               {
55
                                          printf("%4.lf", x[i]);
56
57
                               printf("\n");
58
                               printf("----\n");
59
60
                               k = 0;
61
                               while (1)
62
                               {
63
                                          nabla[0] = A[1][0] + A[1][1] * x[1] + 2 * A[2][0] * x[0] + 3 * A[3][0] * pow(x[0], x[0]) + 3
64
                                                     (2) + 4 * A[4][0] * pow(x[0], 3);
                                          nabla[1] = A[0][1] + 2 * A[0][2] * x[1] + A[1][1] * x[0];
65
66
                                          d[0] = -nabla[0];
67
                                          d[1] = -nabla[1];
68
69
                                          norm = sqrt(nabla[0] * nabla[0] + nabla[1] * nabla[1]); /// ルムを計算
70
71
                                          f = func(x);
72
                                          alpha = Armijo(x, d, nabla);
73
74
                                          if (k <= 5 || (k <= 100 && k % 10 == 0) || k % 100 == 0) //5回目までと 10
75
                                                     n回目,100n回目(n=1,2,3,...10)を表示
                                          {
76
                                                     printf("%d <math>\square \exists \n", k);
77
                                                     printf("現在位置=(%.81f,%.81f)\n 目的関数=%.81f\n 勾配ベクトル=(%.81f
78
                                                                 printf("----\n");
79
                                          }
80
81
                                         x[0] = x[0] + alpha * d[0];
82
                                          x[1] = x[1] + alpha * d[1];
83
                                          k++;
84
85
                                         if (norm < res) ///nabla のノルムが 10^{-8}を越えていたら停止
86
                                          {
87
                                                     break;
88
89
                                          if (k > N) ///反復回数が 1000回越えたら停止
90
91
92
                                                     break;
93
```

```
}
94
95
            printf("--結果--\n");
96
            printf("%d回\n 現在位置=(%.81f,%.81f)\n 目的関数=%.81f\n 勾配ベクトル=(%.81f
97
                ,%.8lf)\n ノルム=%.8lf\n", k - 1, x[0], x[1], f, nabla[0], nabla[1], norm);
            return 0;
98
        }
99
100
        double Armijo(double *x, double *d, double *nabla)
101
        {
102
            double alpha = 1.0;
103
            double xi = 0.1; ///今回の課題条件
104
            double tau = 0.5; ///今回の課題条件
105
            double fx, fy;
106
            double y[2];
107
108
            ///f(x_k)を求める
109
            fx = func(x);
110
111
            while (1)
112
113
                ///新しいalpha に対しての fy を求める
114
               y[0] = x[0] + alpha * d[0];
115
                y[1] = x[1] + alpha * d[1];
116
                fy = func(y);
117
118
               if (fy \le fx + xi * (nabla[0] * d[0] + nabla[1] * d[1]) * alpha) //Armijo 条件
119
120
                   break;
121
122
123
               alpha = tau * alpha; //alpha の更新
124
125
            return alpha;
126
        }
127
128
        //f2 の式を用いている
129
        double func(double *x)
130
        {
131
            return A[0][1] * x[1] + A[0][2] * pow(x[1], 2) + A[1][0] * x[0] + A[1][1] * x[0] * x[1]
132
                + A[2][0] * pow(x[0], 2) + A[3][0] * pow(x[0], 3) + A[4][0] * pow(x[0], 4);
133
        }
```

ソースコード 3: 4619055_辰川力駆_newton_1.c

```
//4619055 辰川力駆
1
2
      #include <stdio.h>
      #include <stdlib.h>
3
      #include <string.h>
4
      #include <math.h>
5
      #define inf 10000000000000
6
7
      #define N 500
      #define res 0.00000001 ///停止条件
8
```

```
9
       double func(double *x); ///f1 の式を用いている
10
       void Hf(double *x, double *d); ///ヘッセ行列を求める
11
       void Newton(double *x, double *d, double *nabla); //d を計算する
12
13
       double Q[2][2]; //Q のデータを保存
14
       double c[2]; //c のデータを保存
15
       double Hessian[2][2]; ///ヘッセ行列を保存
16
17
       int main()
18
19
           char fname[128];
20
           double x[2];
21
           double p, q, r;
22
           double d[2], norm;
23
           double nabla[2];
24
           double f;
25
           FILE *fp;
26
           int i, j, k;
27
28
           printf("input_filename:");
29
           fgets(fname, sizeof(fname), stdin);
30
           fname[strlen(fname) - 1] = '\0';
31
           fflush(stdin);
32
           fp = fopen(fname, "r");
33
34
           for (i = 0; i < 2; i++)
35
36
               for (j = 0; j < 2; j++)
37
38
                   fscanf(fp, "%lf", &Q[i][j]);
39
40
41
           for (i = 0; i < 2; i++)
42
43
               fscanf(fp, "%lf", &c[i]);
44
45
           for (i = 0; i < 2; i++)
46
47
               fscanf(fp, "%lf", &x[i]);
48
49
           fclose(fp);
50
51
           printf("Q=\n");
52
           for (i = 0; i < 2; i++)
53
54
               for (j = 0; j < 2; j++)
55
56
                   printf("%4.lf", Q[i][j]);
57
58
               printf("\n");
59
60
```

```
printf("C=\n");
   61
                                                     for (i = 0; i < 2; i++)
   62
   63
                                                                      printf("%4.lf", c[i]);
   64
   65
                                                     printf("\n");
   66
   67
                                                     printf("x=\n");
                                                     for (i = 0; i < 2; i++)
   68
   69
                                                                      printf("%4.lf", x[i]);
   70
                                                     }
   71
   72
                                                     printf("\n");
                                                     printf("----\n");
   73
   74
                                                     p = Q[0][0];
   75
                                                     q = Q[0][1];
   76
                                                     r = Q[1][1];
   77
                                                     k = 0;
   78
                                                     while (1)
   79
   80
                                                     {
                                                                      nabla[0] = p * x[0] + q * x[1] + c[0] + 2 * (x[0] - x[1]) * exp((x[0] - x[1]) * (x[0] - x[1]
   81
                                                                                        [0] - x[1]);
                                                                      nabla[1] = q * x[0] + r * x[1] + c[1] - 2 * (x[0] - x[1]) * exp((x[0] - x[1]) * (x[0] - x[1]
   82
                                                                                           -x[1]);
   83
                                                                      norm = sqrt(nabla[0] * nabla[0] + nabla[1] * nabla[1]); /// ルムを計算
   85
                                                                      Newton(x, d, nabla); //d を計算
   86
                                                                      f = func(x);
   87
   88
                                                                     if (k <= 5 || (k <= 100 && k % 10 == 0) || k % 100 == 0) //5回目までと 10
   89
                                                                                      n回目,100n回目(n=1,2,3,...10)を表示
                                                                      {
   90
                                                                                      printf("%d <math>\square \exists \n", k);
   91
                                                                                      printf("現在位置=(%.81f,%.81f)\n 目的関数=%.81f\n 勾配ベクトル=(%.81f
   92
                                                                                                          ,\%.81f)\n / N = \%.81f\n'', x[0], x[1], f, nabla[0], nabla[1], norm);
                                                                                      printf("----\n");
   93
   94
                                                                     x[0] = x[0] + d[0];
   95
                                                                     x[1] = x[1] + d[1];
   96
                                                                      k++;
   97
                                                                      if (norm < res) ///nabla のノルムが 10^{-8}を越えていたら停止
   98
   99
                                                                      {
100
                                                                                       break;
101
                                                                     if (k > N) ///反復回数が 500回越えたら停止
102
                                                                       {
103
                                                                                       break;
104
105
                                                                     if (d[0] == inf) ///nabla^2が非正則なら停止
106
                                                                      {
107
                                                                                       printf("非正則なので終了します\n");
108
```

```
break:
109
                   }
110
              }
111
              printf("--結果--\n");
112
              printf("%d回\n現在位置=(%.81f,%.81f)\n目的関数=%.81f\n 勾配ベクトル=(%.81f
113
                    ,%.8lf)\n ノルム=%.8lf\n", k - 1, x[0], x[1], f, nabla[0], nabla[1], norm);
114
              return 0;
          }
115
116
          void Newton(double *x, double *d, double *nabla)
117
118
              double Hessian_inverse[2][2];
119
              Hf(x, d);
120
              if (Hessian[0][0] * Hessian[1][1] — Hessian[0][1] * Hessian[1][0] == 0) //正則かどうか
121
122
              {
                   d[0] = \inf;
123
                   d[1] = \inf;
124
              }
125
              else
126
              {
127
                   \operatorname{Hessian_inverse}[0][0] = 1 / (\operatorname{Hessian}[0][0] * \operatorname{Hessian}[1][1] - \operatorname{Hessian}[0][1] * \operatorname{Hessian}[0][1]
128
                        [1][0] * Hessian[1][1];
                   Hessian\_inverse[0][1] = 1 / (Hessian[0][0] * Hessian[1][1] - Hessian[0][1] * Hessian[0][1]
129
                        [1][0]) * -Hessian[0][1];
                   Hessian\_inverse[1][0] = 1 / (Hessian[0][0] * Hessian[1][1] - Hessian[0][1] * Hessian[0][1]
130
                        [1][0]) * -Hessian[1][0];
                   \operatorname{Hessian\_inverse}[1][1] = 1 / (\operatorname{Hessian}[0][0] * \operatorname{Hessian}[1][1] - \operatorname{Hessian}[0][1] * \operatorname{Hessian}[0][1]
131
                        [1][0] * Hessian[0][0];
                   d[0] = -(Hessian\_inverse[0][0] * nabla[0] + Hessian\_inverse[1][0] * nabla[1]);
132
                   d[1] = -(Hessian\_inverse[0][1] * nabla[0] + Hessian\_inverse[1][1] * nabla[1]);
133
              }
134
         }
135
136
          void Hf(double *x, double *d)
137
138
              double p, q, r;
139
              p = Q[0][0];
140
              q = Q[0][1];
141
              r = Q[1][1];
142
              \operatorname{Hessian}[0][0] = p + 2 * (1 + 2 * \operatorname{pow}(x[0] - x[1], 2)) * \exp(\operatorname{pow}(x[0] - x[1], 2));
143
              Hessian[0][1] = q - 2 * (1 + 2 * pow(x[0] - x[1], 2)) * exp(pow(x[0] - x[1], 2));
144
              \operatorname{Hessian}[1][0] = \operatorname{Hessian}[0][1];
145
              Hessian[1][1] = r + 2 * (1 + 2 * pow(x[0] - x[1], 2)) * exp(pow(x[0] - x[1], 2));
146
         }
147
148
          // f1の式を用いている
149
          double func(double *x)
150
151
152
              double p, q, r;
              p = Q[0][0];
153
              q = Q[0][1];
154
              r = Q[1][1];
155
```

```
return 0.5 * (p * x[0] * x[0] + 2 * q * x[0] * x[1] + r * x[1] * x[1]) + c[0] * x[0] + c
[1] * x[1] + exp((x[0] - x[1]) * (x[0] - x[1]));
157
```

ソースコード 4: 4619055_辰川力駆_newton_2.c

```
//4619055 辰川力駆
1
       #include <stdio.h>
2
       #include <stdlib.h>
3
       #include <string.h>
4
       #include <math.h>
5
       #define inf 10000000000000
6
       #define N 500
7
       #define res 0.00000001 ///停止条件
8
9
       double func(double *x); ///f2 の式を用いている
10
       void Hf(double *x, double *d); ///ヘッセ行列を求める
11
       void Newton(double *x, double *d, double *nabla); //d を計算する
12
13
       double A[5][3]; //A のデータを保存
14
       double Hessian[2][2]; ///ヘッセ行列を保存
15
16
       int main()
17
       {
18
           char fname[128];
19
           double x[2];
20
           double d[2], norm;
21
           double nabla[2];
22
           double f;
23
24
           FILE *fp;
           int i, j, k;
25
26
           printf("input_filename:");
27
           fgets(fname, sizeof(fname), stdin);
28
           fname[strlen(fname) -1] = '\0';
29
           fflush(stdin);
30
           fp = fopen(fname, "r");
31
32
           for (i = 0; i < 5; i++)
33
34
               for (j = 0; j < 3; j++)
35
36
                   fscanf(fp, "%lf", &A[i][j]);
37
38
39
           for (i = 0; i < 2; i++)
40
           {
41
               fscanf(fp, "%lf", &x[i]);
42
43
           fclose(fp);
44
45
           printf("A=\n");
46
           for (i = 0; i < 5; i++)
47
```

```
{
48
              for (j = 0; j < 3; j++)
49
50
                  printf("%4.lf", A[i][j]);
51
52
              printf("\n");
53
          }
54
          printf("x=\n");
55
          for (i = 0; i < 2; i++)
56
57
              printf("%4.lf", x[i]);
58
59
          printf("\n");
60
          printf("----\n");
61
62
          k = 0;
63
          while (1)
64
65
              nabla[0] = A[1][0] + A[1][1] * x[1] + 2 * A[2][0] * x[0] + 3 * A[3][0] * pow(x[0],
66
                  (2) + 4 * A[4][0] * pow(x[0], 3);
              nabla[1] = A[0][1] + 2 * A[0][2] * x[1] + A[1][1] * x[0];
67
68
              norm = sqrt(nabla[0] * nabla[0] + nabla[1] * nabla[1]); ///ノルムを計算
69
70
              Newton(x, d, nabla); //d を計算
71
              f = func(x);
72
              73
                  n回目,100n回目(n=1,2,3,...10)を表示
              {
74
                 printf("%d <math>\square \exists \n", k);
75
                 printf("現在位置=(%.81f,%.81f)\n 目的関数=%.81f\n 勾配ベクトル=(%.81f
76
                      ,\%.81f)\n / N = \%.81f\n'', x[0], x[1], f, nabla[0], nabla[1], norm);
                 printf("----\n");
77
              }
78
              x[0] = x[0] + d[0];
79
80
              x[1] = x[1] + d[1];
              k++;
81
              if (norm < res) ///nabla のノルムが 10^{-8}を越えていたら停止
82
              {
83
                 break;
84
85
              if (k > N) ///反復回数が 500回越えたら停止
86
              {
87
88
                  break;
89
              if (d[0] == inf) ///nabla^2が非正則なら停止
90
91
                  printf("非正則なので終了します\n");
92
                  break;
93
              }
94
          }
95
96
```

```
printf("--結果--\n");
 97
              printf("%d 回\n 現在位置=(%.8lf,%.8lf)\n 目的関数=%.8lf\n 勾配ベクトル=(%.8lf
 98
                   ,%.8lf)\n ノルム=%.8lf\n", k-1, x[0], x[1], f, nabla[0], nabla[1], norm);
              return 0;
 99
         }
100
101
102
         void Newton(double *x, double *d, double *nabla)
103
              double Hessian_inverse[2][2];
104
              Hf(x, d);
105
              \mathbf{if} (Hessian[0][0] * Hessian[1][1] - Hessian[0][1] * Hessian[1][0] == 0) ///正則かどうか
106
107
                  d[0] = \inf;
108
                  d[1] = \inf;
109
              }
110
              else
111
              {
112
                  \operatorname{Hessian\_inverse}[0][0] = 1 / (\operatorname{Hessian}[0][0] * \operatorname{Hessian}[1][1] - \operatorname{Hessian}[0][1] * \operatorname{Hessian}[0][1]
113
                       [1][0] * Hessian[1][1];
                  \operatorname{Hessian\_inverse}[0][1] = 1 / (\operatorname{Hessian}[0][0] * \operatorname{Hessian}[1][1] - \operatorname{Hessian}[0][1] * \operatorname{Hessian}[0][1]
114
                       [1][0]) * -Hessian[0][1];
                  Hessian\_inverse[1][0] = 1 / (Hessian[0][0] * Hessian[1][1] - Hessian[0][1] * Hessian[0][1]
115
                       [1][0] * -\text{Hessian}[1][0];
                  Hessian\_inverse[1][1] = 1 / (Hessian[0][0] * Hessian[1][1] - Hessian[0][1] * Hessian[0][1] 
116
                       [1][0] * Hessian[0][0];
                  d[0] = -(Hessian\_inverse[0][0] * nabla[0] + Hessian\_inverse[1][0] * nabla[1]);
117
                  d[1] = -(Hessian\_inverse[0][1] * nabla[0] + Hessian\_inverse[1][1] * nabla[1]);
118
              }
119
         }
120
121
         void Hf(double *x, double *d)
122
123
              Hessian[0][0] = 2 * A[2][0] + 6 * A[3][0] * x[0] + 12 * A[4][0] * pow(x[0], 2);
124
              Hessian[0][1] = A[1][1];
125
              \operatorname{Hessian}[1][0] = \operatorname{Hessian}[0][1];
126
127
              Hessian[1][1] = 2 * A[0][2];
         }
128
129
         //f2の式を用いている
130
         double func(double *x)
131
132
              133
                  + A[2][0] * pow(x[0], 2) + A[3][0] * pow(x[0], 3) + A[4][0] * pow(x[0], 4);
         }
134
```