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 9月 28日

学籍番号: 4619060

氏名: 照永詩恩

共同実験者:

1　実験の要旨

2　実験の目的

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 $f : \boldsymbol{R}^n \to \boldsymbol{R}$ に対して定義される以下の問題である.

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

3.1.1　諸定義

n 変数関数 $f : \boldsymbol{R}^n \to \boldsymbol{R}$ に対し勾配ベクトル $\nabla f(\boldsymbol{x}$ とヘッセ行列 $\nabla^2 f(\boldsymbol{x})$ 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.

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

勾配ベクトルとヘッセ行列はそれぞれ 1 変数関数 $f(\boldsymbol{x})$ の微分係数 $f'(\boldsymbol{x})$ と 2 階微分係数 $f''(\boldsymbol{x})$ 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を注意する. 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.

$$f(\boldsymbol{a} + \boldsymbol{b}) = f\boldsymbol{a} + \nabla f(\boldsymbol{a})^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \nabla^2 f(\boldsymbol{a}) \boldsymbol{d} + (\text{残差})$$

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

さらに,n 次実行列 X に対し,「X が半正定値 $\Leftrightarrow$ X の固有値がすべて正」が成り立つ.

3.1.2　最適性条件

一般に非線形最適問題において大域的最適解を求めることは難しく, 多くの場合は同署的最適解を求めることを目指す. 局所最適解にお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) $\boldsymbol{x}^* \in \boldsymbol{R}^n$ を無制約最小問題の極小的最適解として次が成り立つ.

- f が連続的微分可能 $\Rightarrow \nabla f(\boldsymbol{x})^* = 0$.

- f が 2 回連続微分可能 $\Rightarrow \nabla f(\boldsymbol{x})^*$ は半正定値行列.

1 次の必要条件を満たす $\boldsymbol{x}^*$ 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.$\boldsymbol{x}^* \in \boldsymbol{R}^n$ が以下の条件を満たすならば,$\boldsymbol{x}^* \in \boldsymbol{R}^n$ は無制約最小化問題の極小的最適解である.

- $\nabla f(\boldsymbol{x}^*) = 0$
- $\nabla^2 f(\boldsymbol{x}^*)$ 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に容易になる. 関数 $f : \boldsymbol{R}^n \to \boldsymbol{R}$ が異常の条件を満たすときは f が凸関数である.

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

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.

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

さらに, 凸関数は次の「局所最適性=大域的最適性」を満たす.

$$f \text{が凸関数の時,} \boldsymbol{x} \text{が} f \text{の局所的最適解} \to \boldsymbol{x}^* \text{は} f \text{の大域的最適解}$$

これにより, 微分可能な凸関数に対しては,$\nabla f(\boldsymbol{x}) = 0$ を満たす $\boldsymbol{x}$ を求めれば, その $\boldsymbol{x}^*$ は最適解であることが保証される. 凸関数はこのように重要なクラスである.

3.1.3 反復法

適当な初期点 $x_0 \in \boldsymbol{R}^n$ 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.

$$x_{k+1} = x_k + \alpha_k d_k$$

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

3.1.4　最急降下法

最急降下法とは,x_k における最急降下方向を常に用いる方法である. 手順は以下の通り.

1. 初期点を選び,$k=0$ とする.$\epsilon = 10^{-8}$ とする.
2. $||\nabla f(\boldsymbol{x_k})|| < \epsilon$ が満たされていれば終了する.
3. 探索方向を $d_k = -\nabla f(\boldsymbol{x_k})$ とする.
4. アルミホ条件による直線探索を用いてステップ幅 α を定める.
5. $x_{k+1} = x_k + \alpha_k d_k$,$k = k+1$ として (2) に戻る.

最急降下法の大域的収束性 関数 $f(\boldsymbol{x})$ に関する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店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, 初期店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 $\{\boldsymbol{x}_k\}$ の収束先を $\boldsymbol{x}^*$ とすると以下の式が成り立つ $0 < c < 1$ が存在する.

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

この式では一見, 収束が遅いように見えないが $||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*$ が非常に微小なため急降下法が収束するまでに要する反復数は非常に多い.

3.1.5　ニュートン法

ニュートン法は $f(\boldsymbol{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. 以下のニュートン方程式を解くことで得られる.

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

手順は以下の通り.

1. 初期点を選び,$k=0$ とする.$\epsilon = 10^{-8}$ とする.
2. $||\nabla f(\boldsymbol{x_k})|| < \epsilon$ が満たされていれば終了する.
3. ニュートン方程式を解いて d_k を求める.

4. $x_{k+1} = x_k + \alpha_k d_k$,$k = k + 1$ として (2) に戻る.

ニュートン法の局所的 2 次収束性 初期点を $\boldsymbol{x}^*$ 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 $\boldsymbol{x}^*$ とすると以下を満たす定数 $c \geq 0$ が存在する.

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*|| \leq c||\boldsymbol{x}_k - \boldsymbol{x}^*||^2$$

1 次収束と異なり, 上の 2 次収束は非常に速い. 一方で, 初期店のと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

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よ.
2. 以下の 2 種類の 2 変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よ.

$$\begin{aligned} f_1(x_1, x_2) &= \frac{1}{2}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} p & q \\ q & r \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + e^{(x_1 - x_2)^2} \\ f_2(x_1, x_2) &= \sum_{i=0}^{4}\sum_{j=0}^{2} a_{ij} x_1^i x_2^j \end{aligned}$$

3. プログラムを動かしてふるまいを観察して考察せよ.
4. 上の関数に対してニュートン法のプログラムを完成せよ
5. プログラムを動かしてふるまいを考察せよ.

5 結果

1. 上記より目的と理論は述べた.
2. 付録にてソースコードを載せる.
3. 各コード 3 つあるが 1 つ取り上げて表にする.
4. ソースコードは付録にのせる
5. それぞれのソースコードの結果を表にする. データが 3 つあるが一つを例に記載する.

表 1: kadai1.c

回数	23
位置	(0.827955,0.232528)
目的関数	-2.559041
勾配ベクトル	(0.000000,0.000000)
ノルム	0.000000

表 2: kadai1_2.c の結果 (data60.txt)

回数	1000
位置	(-15.351735,-287.442815)
目的関数	-51647.764894
勾配ベクトル	(27.717200,10.962134)
ノルム	29.806234

表 3: kadai2.c の結果 (rand_exp1.txt)

回数	16
位置	(0.827955,0.232528)
目的関数	-2.559041
勾配ベクトル	(0.000000,-0.000000)
ノルム	0.000000

表 4: kadai2_2.c の結果 (rand_quart2.txt)

回数	7
位置	(-14.729993,-248.869871)
目的関数	-31254.330351
勾配ベクトル	(0.000000,0.000000)
ノルム	0.000000

6 検討・考察

6.0.1 最急降下法のプログラムの実行結果について

それぞれ最急降下法のソースコードの目的関数の動きをグラフにした.

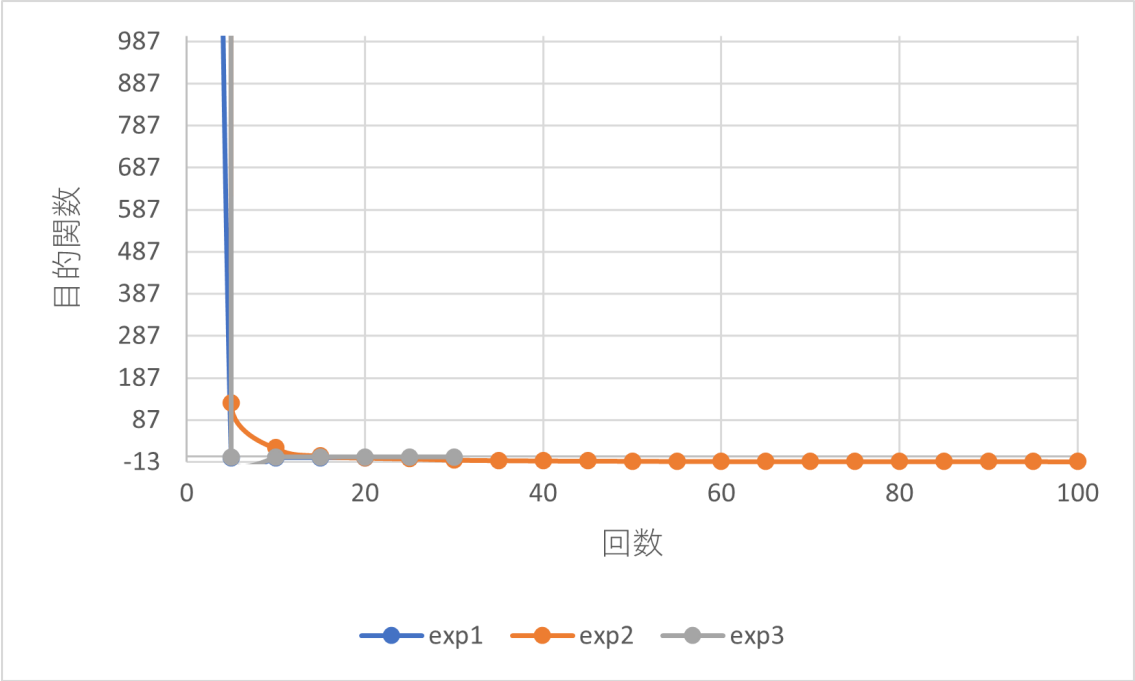

図 1.kadai1.c においてそれぞれのデータの目的関数の動き

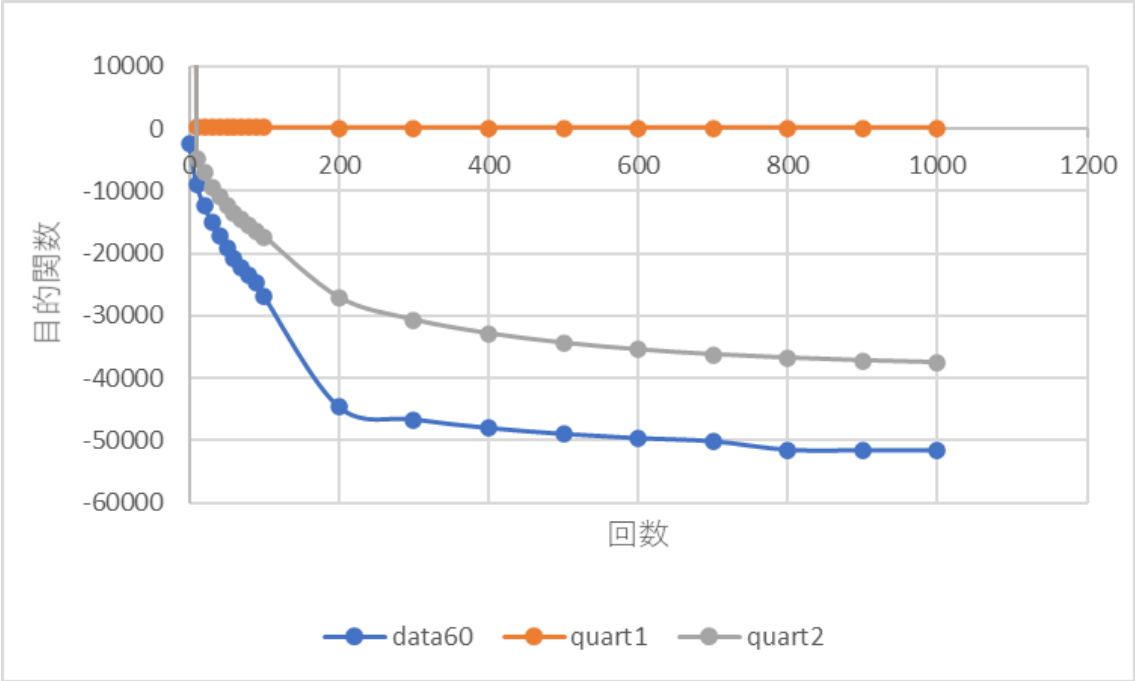

図 2.kadai1_1.c においてそれぞれのデータの目的関数の動き

各図よりそれぞれとある値に収束すると考察した.

6.0.2　ニュートン法のプログラムの実行結果について

それぞれのニュートン法のソースコードの目的関数の動きをグラフにした.

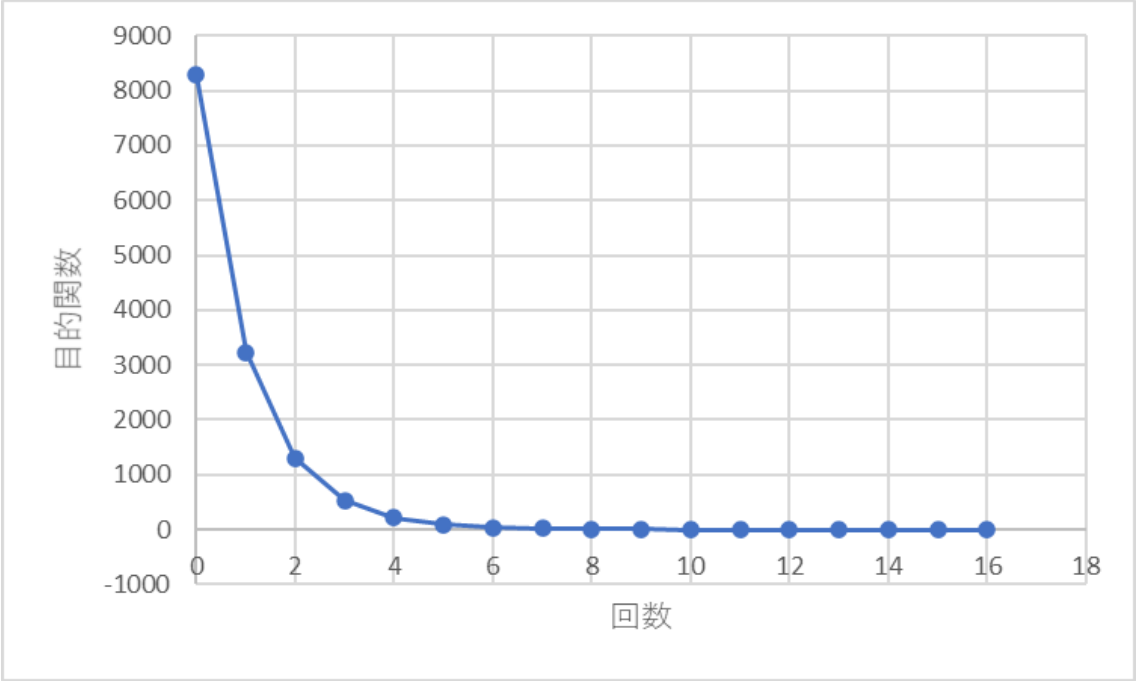

図 3.kadai2.c においてそれぞれのデータの目的関数の動き

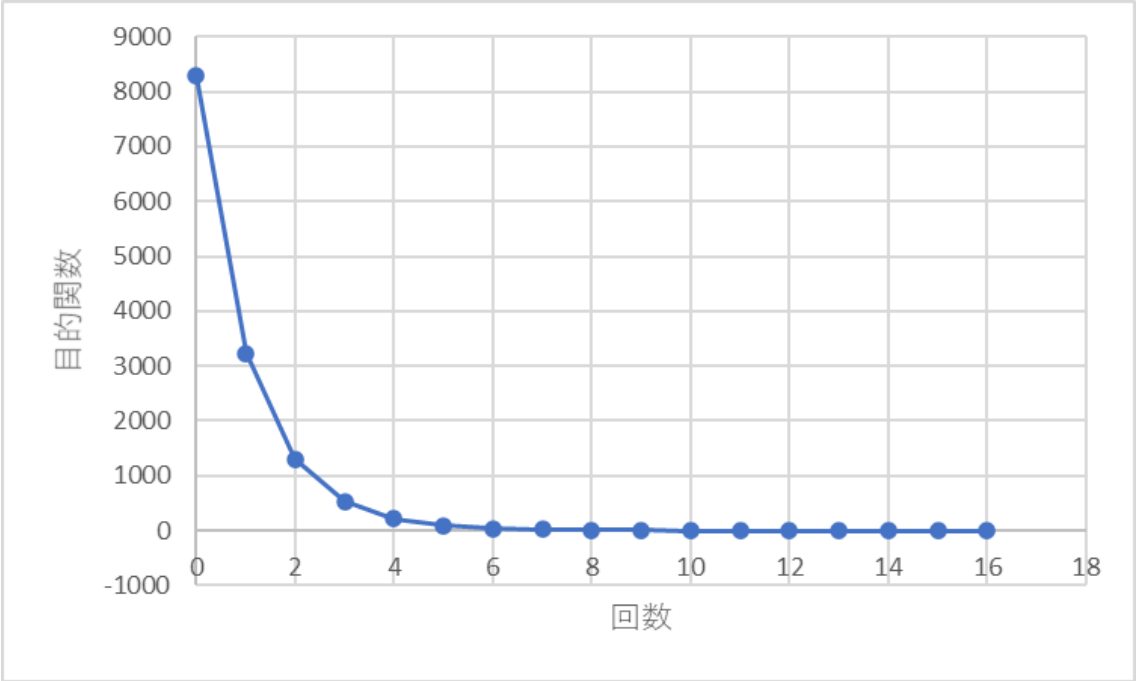

図 4.kadai2_2.c においてそれぞれのデータの目的関数の動き

各グラフより目的関数はとある値に収束している. また, データの値によってヘッセ行列が正則でない場合が出てきてしまうこともあることもあると考える. また, 最急降下法と比較してみるとニュートン法の方が収束するときの回数が早いが, ヘッセ行列が正則でないと正しい値が出ないためその点に関しては最急降下法に劣っていると考えた.

6.0.3　凸かどうかについて

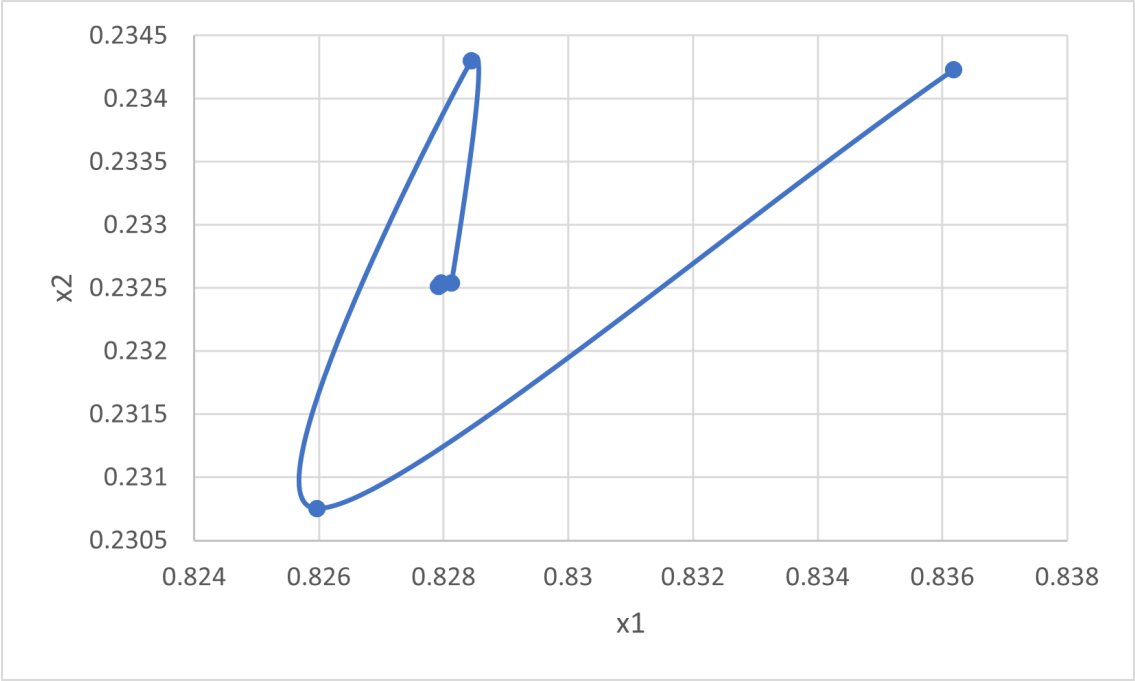

図 5. 最急降下法の $(x_1 - x_2)$ 平面,rand_exp1.txt より

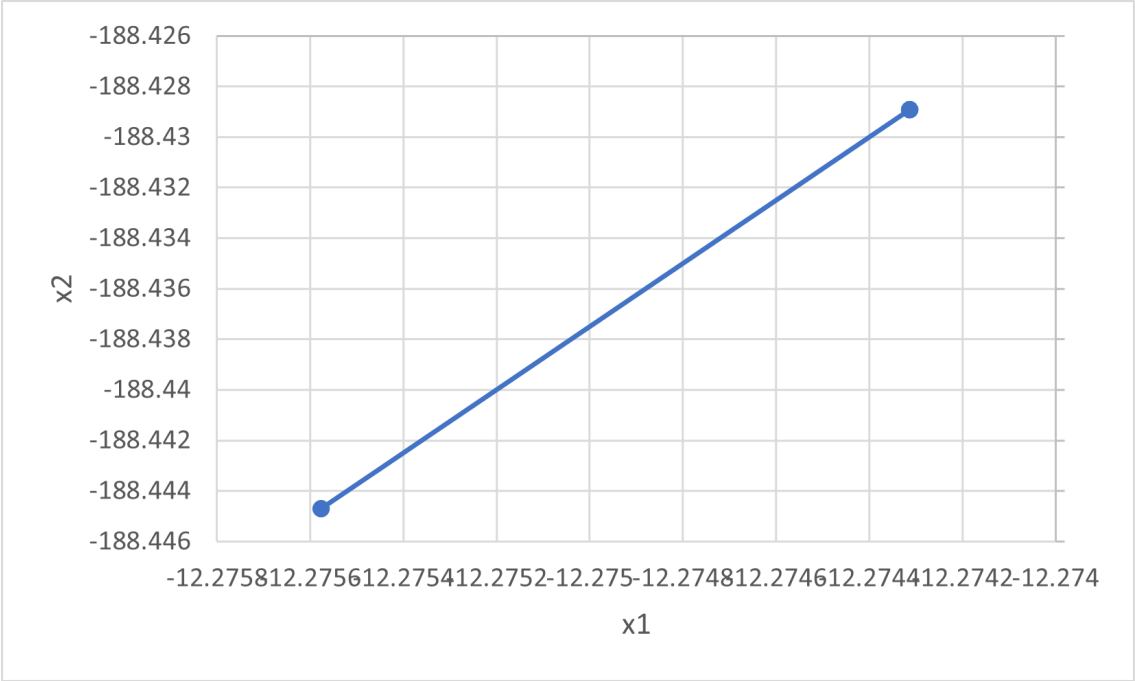

図 6. ニュートン法の $(x_1 - x_2)$ 平面,rand_quart1.txt より

それぞれ,f_1,f_2 関数のデータを 1 つずつ実行してノルムが大幅に小さくなった時に現在の位置をプロットした. 図 5 より最急降下法の法で使ったデータにおいては放物線のような形となったため凸性ではないかと考察した. 逆にニュートン法に用いたデータは直線のようになったためこれは凸性ではないと考えた.

7　結論

最急降下法とニュートン法のプログラムを実際に書いて適応することができさらにその仕組みを知ることができた. またこれらのことを通してアルゴリズムの特性を学ぶことができた.

参考文献

[1] 情報工学実験 2, 東京理科大学 工学部 情報工学科, 2020.

A 付録

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>//exp と pow を使うため
#define N 1000//反復回数の最大
#define eps 0.00000001
double M[2][2];
double C[2];
double function(double *x);//目的関数
double Armijo(double *x,double *d,double *g);//アルミホ条件
int main()
{
    int i,j,k;
    double alpha;
    char fname[128];
    double x[2];
    double p,q,r;
    double g[3],d[2],norm;
    double f;
    FILE *fp;
    //データのファイル操作
    printf("input filename:");
    fgets(fname,sizeof(fname),stdin);
    fname[strlen(fname)-1] = '\0';
    fflush(stdin);
    fp = fopen(fname, "r");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            fscanf(fp, "%lf", &M[i][j]);
        }
    }
    for(i=0;i<2;i++)
    {
        fscanf(fp, "%lf", &C[i]);
    }
    for(i=0;i<2;i++)
    {
```

```
        fscanf(fp, "%lf", &x[i]);
    }
    fclose(fp);
    //初期化表示
    printf("Q=\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            printf("%5.lf",M[i][j]);
        }
        printf("\n");
    }
    printf("C=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",C[i]);
    }
    printf("\n");
    printf("x=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",x[i]);
    }
    printf("\n");
    printf("-------------------------------------\n");
    //ここから開始
    k=0;
    p=M[0][0];
    q=M[0][1];
    r=M[1][1];
    while(1)
    {
        //勾配ベクトル
        g[0]=p*x[0]+q*x[1]+C[0]+2*(x[0]-x[1])*exp(pow(x[0]-x[1],2));
        g[1]=r*x[1]+q*x[0]+C[1]-2*(x[0]-x[1])*exp(pow(x[0]-x[1],2));
        //探索方向
        d[0]=-g[0];
        d[1]=-g[1];
        norm=sqrt((pow(g[0],2)+pow(g[1],2)));
        f=function(x);
```

```
        alpha=Armijo(x,d,g);
        if(norm<eps)
        {
            break;
        }
        if(k<=100&&k%5==0)
        {
            printf("%d 回目\n",k);
            printf("現 在 の 位 置=(%lf,%lf)\n 目 的 関 数=%lf\n 勾 配 ベ ク ト ル
=(%lf,%lf)\n,
            ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");
        }
        else if(k>100&&k%100==0)
        {
            printf("%d 回目\n",k);
            printf("現 在 の 位 置=(%lf,%lf)\n 目 的 関 数=%lf\n 勾 配 ベ ク ト ル
=(%lf,%lf)\n,
            ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");
        }

        x[0]=x[0]+alpha*d[0];
        x[1]=x[1]+alpha*d[1];
        k++;
        if(k>N)
        {
            break;
        }
    }
    printf("最終結果\n");
    printf("回数=%d\n 現在の位置=(%lf,%lf)\n 目的関数=%lf\n
    勾配ベクトル=(%lf,%lf)\n, ノルム=%lf\n",
    k-1,x[0],x[1],f,g[0],g[1],norm);
    return 0;
}
double function(double *x)
{
    double fun;
    double p,q,r;
    p=M[0][0];
    q=M[0][1];
```

```
    r=M[1][1];
    fun=0.5*(p*pow(x[0],2)+2*q*x[0]*x[1]
    +r*pow(x[1],2))+C[0]*x[0]+C[1]*x[1]+exp(pow((x[0]-x[1]),2));
    return fun;
}
double Armijo(double *x,double *d,double *g)
{
    double alpha=1.0;
    double xi=0.1;
    double tau=0.5;
    double f1,f2,f3;
    double z;
    double p,q,r;
    double y[2];
    p=M[0][0];
    q=M[0][1];
    r=M[1][1];
    while(1)
    {
        y[0]=x[0]+alpha*d[0];
        y[1]=x[1]+alpha*d[1];
        f1=function(y);
        f2=function(x);
        z=xi*(g[0]*d[0]+g[1]*d[1])*alpha;
        if(f1<=f2+z)
        {
            break;
        }
        else
        {
            alpha=tau*alpha;
        }
    }
    return alpha;
}
```

図 A.1:最急降下法 f_1 のソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define N 1000
#define eps 0.00000001
double M[5][3];
double function(double *x);
double Armijo(double *x,double *d,double *g);
int main()
{
    int i,j,k;
    double alpha;
    char fname[128];
    double x[2];
    double p,q,r;
    double g[3],d[2],norm;
    double f;
    FILE *fp;

    printf("input filename:");
    fgets(fname,sizeof(fname),stdin);
    fname[strlen(fname)-1] = '\0';
    fflush(stdin);
    fp = fopen(fname, "r");
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
        {
            fscanf(fp, "%lf", &M[i][j]);
        }
    }
    for(i=0;i<2;i++)
    {
        fscanf(fp, "%lf", &x[i]);
    }
    fclose(fp);
    //初期化表示
    printf("a=\n");
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
```

```
        {
            printf("%5.lf",M[i][j]);
        }
        printf("\n");
    }
    printf("x=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",x[i]);
    }
    printf("\n");
    printf("-----------------------------------\n");
    k=0;
    while(1)
    {
        g[0]=M[1][0]+M[1][1]*x[1]+2*M[2][0]*x[0]+3*M[3][0]
        *pow(x[0],2)+4*M[4][0]*pow(x[0],3);
        g[1]=M[0][1]+2*M[0][2]*x[1]+M[1][1]*x[0];
        norm=sqrt((pow(g[0],2)+pow(g[1],2)));
        f=function(x);
        d[0]=-g[0];
        d[1]=-g[1];
        alpha=Armijo(x,d,g);
        if(norm<eps)
        {
            break;
        }
        if(k<=100&&k%10==0)
        {
            printf("%d回目\n",k);
            printf("現在の位置=(%lf,%lf)\n 目的関数=%lf\n 勾配ベクトル
=(%lf,%lf)\n,
            ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");
        }
        else if(k>100&&k%100==0)
        {
            printf("%d回目\n",k);
            printf("現在の位置=(%lf,%lf)\n 目的関数=%lf\n 勾配ベクトル
=(%lf,%lf)\n,
            ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");
```

```
        }

        x[0]=x[0]+alpha*d[0];
        x[1]=x[1]+alpha*d[1];
        k++;
        if(k>N)
        {
            break;
        }
    }
    printf("最終結果\n");
    printf("回数=%d\n 現在の位置=(%lf,%lf)\n 目的関数=%lf\n
    勾配ベクトル=(%lf,%lf)\n, ノルム=%lf\n",k-1,x[0],x[1],f,g[0],g[1],norm);
    return 0;
}
double function(double *x)
{
    double fun;
    fun=M[0][1]*x[1]+M[0][2]*pow(x[1],2)+M[1][0]*x[0]+M[1][1]*x[0]*x[1]
    +M[2][0]*pow(x[0],2)+M[3][0]*pow(x[0],3)+M[4][0]*pow(x[0],4);
    return fun;
}
double Armijo(double *x,double *d,double *g)
{
    double alpha=1.0;
    double xi=0.1;
    double tau=0.5;
    double f1,f2,f3;
    double z;
    double y[2];
    while(1)
    {
        y[0]=x[0]+alpha*d[0];
        y[1]=x[1]+alpha*d[1];
        f1=function(y);
        f2=function(x);
        z=xi*(g[0]*d[0]+g[1]*d[1])*alpha;
        if(f1<=f2+z)
        {
            break;
        }
        else
```

```
        {
            alpha=tau*alpha;
        }
    }
    return alpha;
}
```

図 A.2:最急降下法 f_2 のソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define inf 10000000000000
#define N 500
#define eps 0.00000001
double M[2][2];
double C[2];
double h[2][2];
double function(double *x);
double h_s(double *x,double *d);//ヘッセ行列を求める
double N_T(double *x,double *d,double *g);
int main()
{
    int i,j,k;
    char fname[128];
    double x[2];
    double p,q,r;
    double g[2],d[2],norm;
    double f;
    FILE *fp;
    //ファイル操作
    printf("input filename:");
    fgets(fname,sizeof(fname),stdin);
    fname[strlen(fname)-1] = '\0';
    fflush(stdin);
    fp = fopen(fname, "r");
```

```
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            fscanf(fp, "%lf", &M[i][j]);
        }
    }
    for(i=0;i<2;i++)
    {
        fscanf(fp, "%lf", &C[i]);
    }
    for(i=0;i<2;i++)
    {

        fscanf(fp, "%lf", &x[i]);
    }
    fclose(fp);
    //初期化表示
    printf("Q=\n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
            printf("%5.lf",M[i][j]);
        }
        printf("\n");
    }
    printf("C=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",C[i]);
    }
    printf("\n");
    printf("x=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",x[i]);
    }
    printf("\n");
    printf("-----------------------------------\n");
    //ここから開始
    k=0;
```

```
    p=M[0][0];
    q=M[0][1];
    r=M[1][1];
    while(1)
    {
        g[0]=p*x[0]+q*x[1]+C[0]+2*(x[0]-x[1])*exp(pow(x[0]-x[1],2));
        g[1]=r*x[1]+q*x[0]+C[1]-2*(x[0]-x[1])*exp(pow(x[0]-x[1],2));
        N_T(x,d,g);
        norm=sqrt((pow(g[0],2)+pow(g[1],2)));
        f=function(x);
        if(norm<eps||d[0]==inf)
        {
            break;
        }

            printf("%d回目\n",k);
            printf("現在の位置=(%lf,%lf)\n 目的関数=%lf\n 勾配ベクトル
=(%lf,%lf)\n
            ,ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");


        x[0]=x[0]+d[0];
        x[1]=x[1]+d[1];
        if(k>N)
        {
            break;
        }
        k++;
    }
    printf("最終結果\n");
    printf("回数=%d\n 現在の位置=(%lf,%lf)\n 目的関数=%lf\n 勾配ベクトル
=(%lf,%lf)\n,
    ノルム=%lf\n",k,x[0],x[1],f,g[0],g[1],norm);
    return 0;
}
double function(double *x)
{
    double fun;
    double p,q,r;
    p=M[0][0];
    q=M[0][1];
```

```
    r=M[1][1];
    fun=0.5*(p*pow(x[0],2)+2*q*x[0]*x[1]+r*pow(x[1],2))
    +C[0]*x[0]+C[1]*x[1]+exp(pow((x[0]-x[1]),2));
    return fun;
}
double h_g(double *x,double *d)
{
    double p,q,r;
    p=M[0][0];
    q=M[0][1];
    r=M[1][1];
    h[0][0]=p+2*exp(pow(x[0]-x[1],2))*(1+2*pow(x[0]-x[1],2));
    h[0][1]=q-2*exp(pow(x[0]-x[1],2))*(1+2*pow(x[0]-x[1],2));
    h[1][0]=h[0][1];
    h[1][1]=r+2*exp(pow(x[0]-x[1],2))*(1+2*pow(x[0]-x[1],2));
}
double N_T(double *x,double *d,double *g)
{
    double h_gg[2][2];
    h_g(x,d);
    if(h[0][0]*h[1][1]-h[0][1]*h[1][0]==0)//正則かどうかの判定
    {
        d[0]=inf;
        d[1]=inf;
        printf("非正則\n");
    }
    else
    {
        h_gg[0][0]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*h[1][1];
        h_gg[1][0]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*-h[1][0];
        h_gg[1][1]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*h[0][0];
        h_gg[0][1]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*-h[0][1];
        d[0]=-g[0]*h_gg[0][0]-g[1]*h_gg[1][0];
        d[1]=-g[0]*h_gg[0][1]-g[1]*h_gg[1][1];
    }
}
```

図 A.3:ニュートン法 f_1 に関するソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define inf 10000000000000
#define N 500
#define eps 0.00000001
double M[5][3];
double h[2][2];
double function(double *x);
void h_s(double *x,double *d);
void N_T(double *x,double *d,double *g);
int main()
{
    int i,j,k;
    char fname[128];
    double x[2];
    double p,q,r;
    double g[2],d[2],norm;
    double f;
    FILE *fp;

    printf("input filename:");
    fgets(fname,sizeof(fname),stdin);
    fname[strlen(fname)-1] = '\0';
    fflush(stdin);
    fp = fopen(fname, "r");
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
        {
            fscanf(fp, "%lf", &M[i][j]);
        }
    }
    for(i=0;i<2;i++)
    {

        fscanf(fp, "%lf", &x[i]);
    }
    fclose(fp);
    //初期化表示
    printf("Q=\n");
```

```
    for(i=0;i<5;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%5.lf",M[i][j]);
        }
        printf("\n");
    }
    printf("x=\n");
    for(i=0;i<2;i++)
    {
        printf("%5.lf",x[i]);
    }
    printf("\n");
    printf("-----------------------------------\n");
    k=0;
    p=M[0][0];
    q=M[0][1];
    r=M[1][1];
    while(1)
    {
        g[0]=M[1][0]+M[1][1]*x[1]+2*M[2][0]*x[0]+3*M[3][0]
        *pow(x[0],2)+4*M[4][0]*pow(x[0],3);
        g[1]=M[0][1]+2*M[0][2]*x[1]+M[1][1]*x[0];
        N_T(x,d,g);
        norm=sqrt((pow(g[0],2)+pow(g[1],2)));
        f=function(x);
        if(norm<eps||d[0]==inf)
        {
            break;
        }
            printf("%d回目\n",k);
            printf("現在の位置=(%lf,%lf)\n 目的関数=%lf\n 勾配ベクトル
=(%lf,%lf)\n
            ,ノルム=%lf\n",x[0],x[1],f,g[0],g[1],norm);
            printf("-----------------------------------\n");

        x[0]=x[0]+d[0];
        x[1]=x[1]+d[1];
        if(k>N)
        {
            break;
```

```
        }
        k++;
    }
    printf("最終結果\n");
    printf("回数=%d\n 現在の位置=(%lf,%lf)\n 目的関数=%lf\n
    勾配ベクトル=(%lf,%lf)\n, ノルム=%lf\n",k,x[0],x[1],f,g[0],g[1],norm);
    return 0;
}
double function(double *x)
{
    double fun;
    fun=M[0][1]*x[1]+M[0][2]*pow(x[1],2)+M[1][0]*x[0]+M[1][1]*
    x[0]*x[1]+M[2][0]*pow(x[0],2)+M[3][0]*pow(x[0],3)
    +M[4][0]*pow(x[0],4);
    return fun;
}
void h_g(double *x,double *d)
{
    double p,q,r;
    h[0][0]=2*M[2][0]+6*M[3][0]*x[0]+12*M[4][0]*pow(x[0],2);
    h[0][1]=M[1][1];
    h[1][0]=M[1][1];
    h[1][1]=2*M[0][2];
}
void N_T(double *x,double *d,double *g)
{
    double h_gg[2][2];
    h_g(x,d);
    if(h[0][0]*h[1][1]-h[0][1]*h[1][0]==0)
    {
        d[0]=inf;
        d[1]=inf;
        printf("非正則\n");
    }
    else
    {
        h_gg[0][0]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*h[1][1];
        h_gg[1][0]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*-h[1][0];
        h_gg[1][1]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*h[0][0];
        h_gg[0][1]=1/(h[0][0]*h[1][1]-h[0][1]*h[1][0])*-h[0][1];
        d[0]=-g[0]*h_gg[0][0]-g[1]*h_gg[1][0];
        d[1]=-g[0]*h_gg[0][1]-g[1]*h_gg[1][1];
```

```
    }
}
```

図 A.4:ニュートン法 f_2 に関するソースコード