情報工学実験 II レポート（探索アルゴリズム 1)

曜日＆グループ番号: 金曜日＆グループ 9

実施日:2017 年 1 月 20 日 (金)、提出日:2017 年 2 月 0 日

グループメンバ

今回の実験はグループメンバ全員で取り組んだが, レポート化にあたって分担を行った．グループメンバと担当した分野は次の通りである．

- 155706J 久場翔悟: 担当 Level2.1
- 155711E 平木宏空: 担当 Level1.1，1.2，1.3
- 155716F 石塚海斗: 担当 Level2.2, 全体フローチャート
- 155730B 清水隆博: 担当 Level2,Level2.3, レポートまとめ

提出したレポート一式について

レポート一式は “shell:/net/home/teacher/tnal/2016-search1-mon/group9/” にアップロードした。提出したファイルのディレクトリ構成は以下の通りである。

```
./reportskel-search1/ #今回のレポートのファイル
    |figs #図のまとめ．尚 level 別にディレクトリを設けた．
./steepestsearch2-1/ #Level2.1 で作成したプログラム一式
./steepestsearch2-2/ #Level2.2 で作成したプログラム一式
./steepestsearch2-3/ #Level2.3 で作成したプログラム一式
```

1 Leve l1: 最適化とは

1.1 Level 1.1: コンピュータと人間の違いを述べよ

1.1.1 課題説明

コンピュータが人間より得意とするモノ、その反対に人間より不得手のモノ、両者について 2 つ以上の視点（立場や観点など）を示し、考察する。

1.1.2 考察

ロボットが得意なもの

- データ処理など, 既存のものをどうにかするという作業
- 同時に複数の処理や, 仕事をすることができる
- 離れていても, ネットワークが繋がっていたら遠くとのデータの処理をすばやくできる
- 演算処理が得意
- 記憶力がある

ロボットが不得意なもの

- 全くの新しいものを一から作る
- 電源が付いてなかったら何もできない
- 運用コストが大きい
- 移動できない
- 欠損の可能性
- 感情に対する行動
- 命令しないと動かない
- 途中結果を飛ばすということができない

やはりコンピュータは人間に比べて, 演算能力ははるかに高く, 膨大な量のデータの処理等は得意であるように感じる. それに関連して, 同時に複数の作業を行うことができるというのも, コンピュータが得意とすることだろう.

それに比べて, コンピュータというものは人間に命令されなければ動くことができず, 一から, 新しいものを考えて作る, もしくは想像するというのは不得意のようだ. 最近ではペッパーなど, 多少移動ができるロボットやコンピュータがあらわれているが, やはりそもそもとして電源が付いてないと何も出来ないというところはロボットの不得意とするところであると

考える.

1.2 Level 1.2: 住宅価格を推定するモデルについて

1.2.1 課題説明

Housing Data Set[2] における RM（平均部屋数）から MEDV（平均価格）を推定するためのモデルについて検討した。

1.2.2 モデル

自分たちの班では一次関数をグラフの中に引くという案が出た. 点が密集している部分を通るように, 一次関数を引くという方法である.

1.2.3 モデルへの入力

平均価格の数値をこの一次関数に入力.

1.2.4 モデルにおける処理内容

一次関数は入力された数値に比例する値を出力する.

1.2.5 モデルの出力

出力された値というのは, 平均価格に対する部屋数を表している.

1.2.6 問題点

この方法はある程度の比較された値を求めることが可能であるが, その一次関数から著しく離れた値を切り捨てることになるのでそのあたりの数値に関して信憑性が保てないという問題点がある.

1.3 Level 1.3: モデルの良さを評価する方法について

1.3.1 課題説明

Level 1.2 で検討したモデルの適切さを評価する指標について検討した。

1.3.2 評価に用いる情報源

評価を用いる情報源としては, 班員で話し合った後, 確かめた.

1.3.3 評価手順

評価手順としては数値が大きいほど, 信憑性が薄く, 数値が小さいほど信憑性が高いものとなっている.

1.3.4 評価に基づいた適切さを計る方法

モデルに対して, それぞれの点から直線に対して直角に引いた差分を平均してその正確さを求めるという方法が出た.

例えば, その平均の値が大きいのならば, モデルに対して差分が大きいということなので, そのモデルの信憑性は薄れるということになり, 逆に小さければそのモデルの信憑性が高いということになる.

2 Level 2: 最急降下法による最適化

2.1 課題説明

3 種類の連続関数 $y = x^2$、$z = x^2 + y^2$、$y = -x \times \cos(x)$ について、最急降下法の適用を通して探索挙動を観察した。以下では, 最急降下法のアルゴリズムについてフローチャートを用いて解説する。その後、3 種類の関数毎にプログラムの変更箇所、観察意図観察方法、観察結果、考察について説明する。

2.2 Level 2 共通部分

2.2.1 最急降下法のアルゴリズム

最急降下法とは

> 微分値を基に x を移動させる幅を決定する際
>
> $$x_{next} = x - \alpha * f'(x)$$
>
> によって移動先を算出するアルゴリズムである．[1]

微分値はその対象のモデル式における切片の傾きに等しい．つまり, ある点から切片を引き, そこに対して学習係数文移動させる方法である．最急降下法の名前の通り, 関数の最小値へと引き寄せられるように移動していく．尚最大値を求める場合は, 学習係数 α を符号反転させれば良い．この際, 探索点が移動した場所を可視化すると, より良い精度の場合はモデル式と同じグラフを辿ると考える．

また最急降下法を用いた今回の C プログラム steepest_decent.c についてフローチャート図を用いて解説を行う． (図:1)

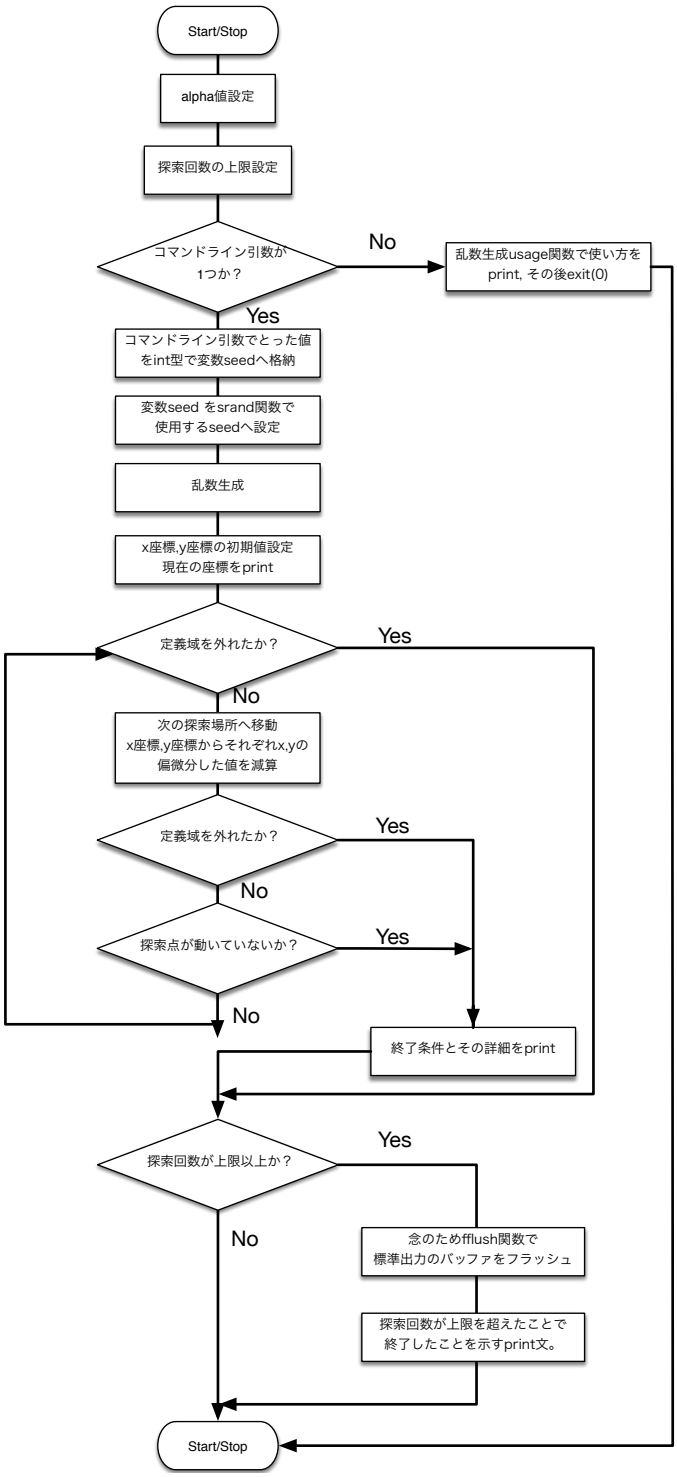

図 1　steepest_decent.c のフローチャート図

この C プログラムは, フローチャート図に示す通り探索場所を移動する度に定義域, mた前回との探索場所の移動について分岐させ, 最適解を見つけ出すプログラムである．最急降下法を用いるにあって, このプログラムではプログラマがソースコード中にモデル式及び偏微分を行った結果を入力する必要がある．今回の実験では level2 の課題はこの C プログラムをそのまま, 又は一部変更して利用した．

2.3 Level2.1: $y = x^2$ について

2.3.1 プログラムソース（変更部分）

steepest_decent.c の一部を以下のように変更した

```
    /** 以下の式を編集して完成させよ (1) **/
    z = x*x;

    /** 以下の式を編集して完成させよ (2-1) **/
    z_dx = 2*x;
```

更に今の alpha の値が見やすいように以下の出力を追加した。

```
     printf("alpha = %.2f\n",alpha);
```

2.3.2 観察意図と観察方法

刻み値をより小さくすると探索点がより細かく移動するため最適性は良くなるだろう。しかし，小さくしすぎると最適性は良いが探索回数が増えてしまい，効率性は悪くなるだろう。

上記のように予想してこれを検証するために，seed 値を 1 に固定し alpha の値を変えることにより探索を行い，結果を観測する。

2.3.3 実行結果

alpha の値を 0.1 から 0.1 刻みに 1 まで実行した。以下に実行結果の一部を示す。

```
alpha = 0.10
~~~~
step 62 x -0.0000072277 y 5.1121064439 f(x,y) 0.0000000001 5.224013e-11
step 63 x -0.0000057822 y 5.1121064439 f(x,y) 0.0000000000 3.343369e-11
~~~~
step 74 x -0.0000004967 y 5.1121064439 f(x,y) 0.0000000000 2.466971e-13
FINISH 3 step 75 x and y were not updated.

alpha = 0.20
~~~~
step 27 x -0.0000075424 y 5.1121064439 f(x,y) 0.0000000001 5.688705e-11
step 28 x -0.0000045254 y 5.1121064439 f(x,y) 0.0000000000 2.047934e-11
~~~~
step 34 x -0.0000002111 y 5.1121064439 f(x,y) 0.0000000000 4.457906e-14
FINISH 3 step 35 x and y were not updated.

alpha = 0.40
~~~~
step 8 x -0.0000188653 y 5.1121064439 f(x,y) 0.0000000004 3.558982e-10
step 9 x -0.0000037731 y 5.1121064439 f(x,y) 0.0000000000 1.423593e-11
~~~~
step 12 x -0.0000000302 y 5.1121064439 f(x,y) 0.0000000000 9.110995e-16
FINISH 3 step 13 x and y were not updated.

alpha = 0.50
step 0 x -7.3692442371 y 5.1121064439 f(x,y) 54.3057606266 5.430576e+01
step 1 x 0.0000000000 y 5.1121064439 f(x,y) 0.0000000000 0.000000e+00
FINISH 3 step 2 x and y were not updated.

alpha = 0.60
~~~~
step 8 x -0.0000188653 y 5.1121064439 f(x,y) 0.0000000004 3.558982e-10
step 9 x 0.0000037731 y 5.1121064439 f(x,y) 0.0000000000 1.423593e-11
~~~~
```

```
step 12 x -0.0000000302 y 5.1121064439 f(x,y) 0.0000000000 9.110995e-16
FINISH 3 step 13 x and y were not updated.

alpha = 0.90
~~~~
step 62 x -0.0000072277 y 5.1121064439 f(x,y) 0.0000000001 5.224013e-11
step 63 x 0.0000057822 y 5.1121064439 f(x,y) 0.0000000000 3.343369e-11
~~~~
step 84 x -0.0000000533 y 5.1121064439 f(x,y) 0.0000000000 2.844223e-15
FINISH 3 step 85 x and y were not updated.

alpha = 1.00
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
```

alpha の値が 0.1 のときから徐々に step 数が減少していき，alpha=0.5 で最も step 数が少なくなった。更に alpha の値を増やしていくと step 数が増加していった。

2.3.4 考察

各 alpha 値で f(x) の値が 0 になる箇所を示したが，f(x) の表示されている範囲の値が 0 になってから実行が終わるまでにかかった step 数が alpha=0.5 から離れるに連れて多くなっていることがわかった。偏微分の値がほぼ 0 になると x の値が 0 に近づいているということになるが，そこから実行されている回数が多いほど更に細かく見ているため最適性が良くなると思った。しかし最も最適性がよかった alpha 値から少なくなるごとに最適性が悪くなっていき，alpha 値が低くても最適性が良いというわけではなかった。しかし効率性に関しては予想通り悪くなっていた (下図 2,3,4)。

逆に alpha 値が大きくなっていくと探索回数が増えていき，alpha=1.0 からは実行が 1000 回を超えてしまった (下図 5)。

探索点の推移のグラフから，alpha=0.2 の際はほぼ 0 になってそこから動くことが少なくなった時点で動作を終了しているが，alpha=0.5 の際は運が良く一度の試行で最小値を導いていることがわかる (下図 6,7)。

alpha の値が小さくなりすぎると効率性に欠けてしまい，alpha の値が大きすぎると探索点が範囲外に出てしまう可能性が高くなってしまうことより，関数 $y = x^2$ に関して学習係数 alpha は 0 より大きく 1 未満である適切な数が好ましいと考察される。今回 seed 値 1 の場合は alpha=0.5 であった。

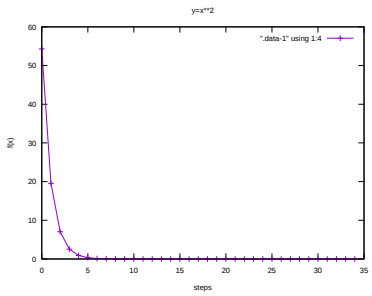

図 2　alpha=0.2 での目的関数の推移

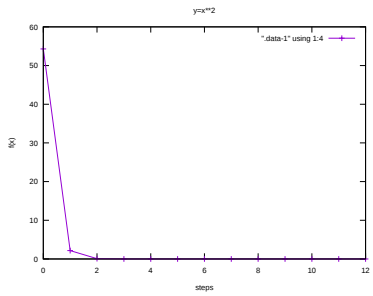

図 3　alpha=0.4 での目的関数の推移

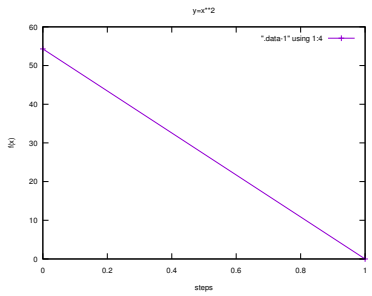

図 4　alpha=0.5 での目的関数の推移

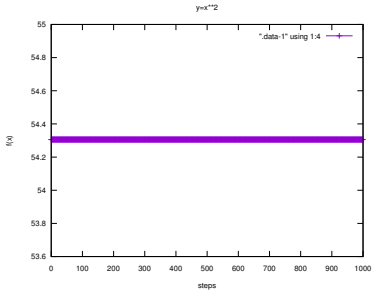

図 5　alpha=1.0 での目的関数の推移

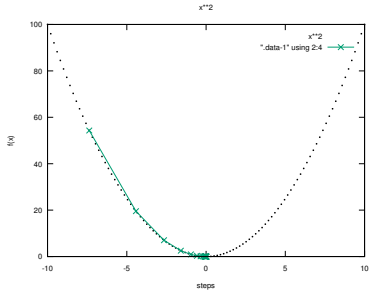

図 6　alpha=0.2 での探索点の推移

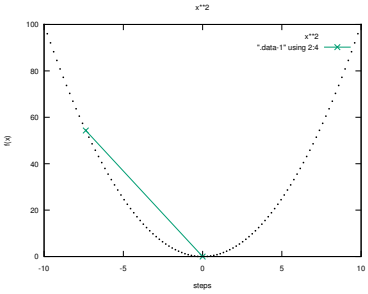

図 7　alpha=0.5 での探索点の推移

2.4　Level2.2: $z = x^2 + y^2$ について

2.4.1　フローチャート共通からの変更点

今回,Level2.2 に取り組むにあたって, プログラムの変更を行ったため, フローチャートにも多少の変更が出た。変更点を以下の図 8 に示す。

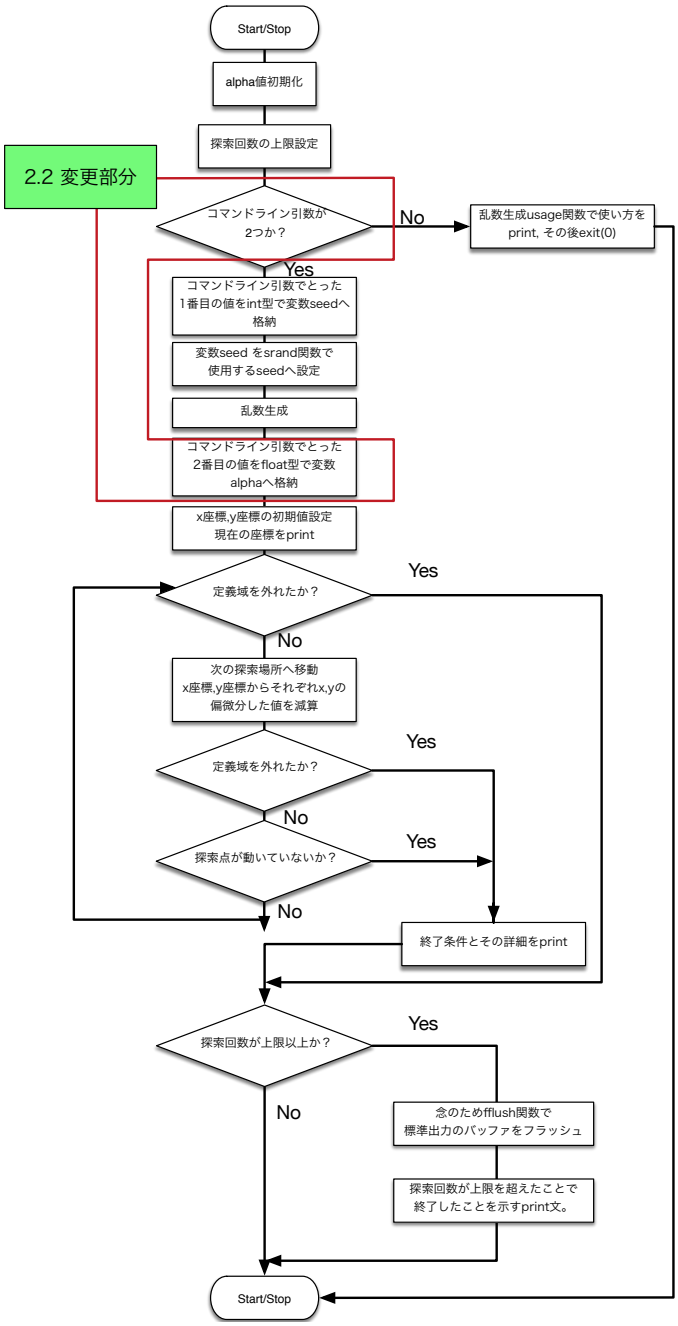

図 8　フローチャート変更点

2.4.2　**プログラムソース（変更部分）**

以下の図 9 に変更部分のみを示す。

図 9　Level2.2 変更点

```
//main 関数直後
f( argc != 3 ){
(略)
}else{
(略)
alpha = atof(argv[2]);
(略)
}

//f 関数内
//  z = x;
  z = x*x + y*y;

//pd_x 関数内
//  z_dx = 1;
  z_dx = 2*x;

//pd_y 関数内
//  z_dy = 0;
  z_dy = 2*y;
```

2.4.3　観察意図と観察方法

seed 値を固定して alpha 値を変動させることで, 探索点の刻み幅による探索の最適性及び効率性を検討する。alpha 値を変更することで探索幅も変動することから, 以下の 2 つのような予想が立てられる。1, alpha 値が大きければ探索点の刻み幅も大きくなり, 効率性を向上できるが, 最適性が低下する。2, alpha 値が小さければ探索点の刻み幅が小さくなるため, 最適性を向上できるが, 効率性が低下する。

効率性の観察方法は,seed 値を固定し alpha 値を変動させた際の step 数の推移を各 seed 値 (範囲 1000-10000 1000 刻み) ごとに表して行う。効率性の検討はグラフより, 各 seed 数で最小 step 数の alpha 値を読み取り, 最も優れた alpha 値を多数決で決定し, それをこのプログラムの最大効率であると決定, 改善点を考察する。

また,srand 関数にコマンドライン引数の seed 値が渡されているため, rand 値は同じ seed 値 を入力している限り, 一定である。よって, 同一 seed 値 において, 乱数を考慮しての複数実行, 実行結果の平均値取得等はしない。

最適性の観察方法は,seed 値を固定し alpha 値を変動させた際の終了時座標, これと傾きが 0 になっている座標 (ここでは 0,0) との差の推移を各 seed 値 (範囲 1000-10000 1000 刻み) ごとにグラフで表す。最適性の検討はグラフからその差が最小の alpha の値を各 seed から読み, 最も優れた alpha 値を多数決で決定する。その alpha 値をこのプログラムの最大最適値 であると決定し, 改善点を考察する。

2.4.4　実行結果

1 効率性について

効率性の観察のための手法として, 各 seed 値 (1000-10000 の 1000 刻み 10 種), 各 alpha 値 (0.001 と 0.1-1.0(0.1 刻み) の計 11 種) ごとに最終 step 数を plot していき, その推移傾向を観察することで行った。

以下の図 10 がその結果である。

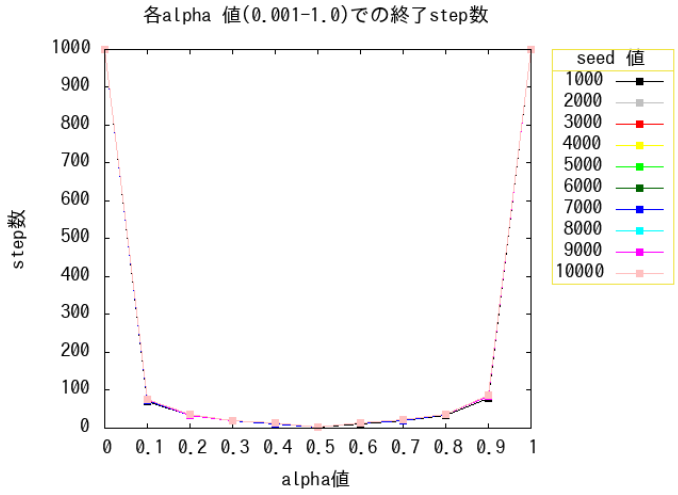

図 10　各 alpha 値 (0.001-1.0) での終了 step 数

alpha 値が 0.001, 1.0 の時に step 数が 1000 となり,alpha 値 0.1-0.9 までの点の差が見えづらい。よって,alpha 値が 0.1-0.9 の範囲の最終 step 数のグラフも図 11 に用意した。

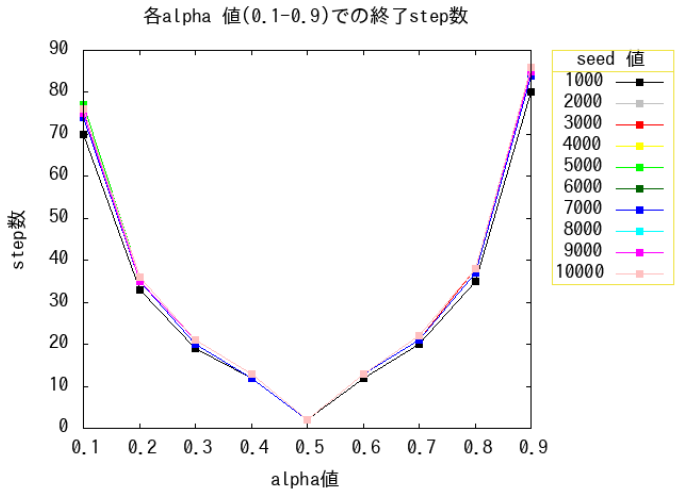

図 11　各 alpha 値 (0.1-0.9) での終了 step 数

2 最適性について

最適性の観察のための手法として, 各 seed 値 (1000-10000 の 1000 刻み 10 種), 各 alpha 値 (0.001 と 0.1-1.0(0.1 刻み) の計 11 種) ごとに終了座標の誤差 (x,y の合計) を plot していき, その推移傾向を観察することで行った。

以下の図 12 がその結果である。

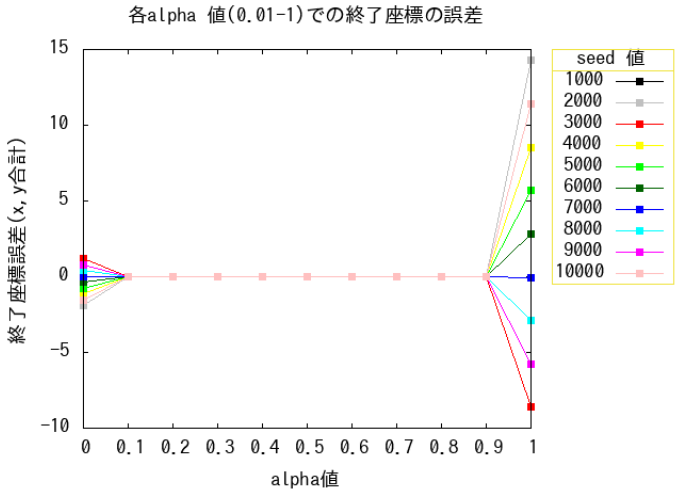

図 12　各 alpha 値 (0.001-1.0) での終了座標誤差 (x,y 合計)

alpha 値が 0.001, 1.0 の時に 最大合計誤差 10 を超え,alpha 値 0.1-0.9 までの点の差が見えづらい。よって,alpha 値が 0.1-0.9 の範囲の合計誤差のグラフも図 11 に用意した。

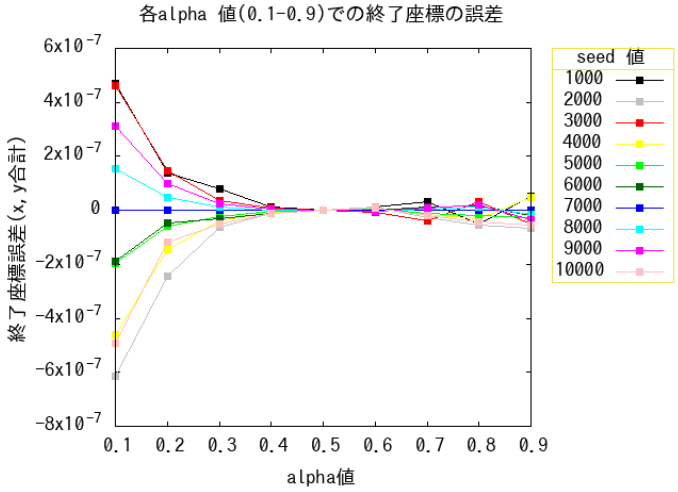

図 13　各 alpha 値 (0.1-0.9) での終了座標誤差 (x,y 合計)

2.4.5　考察

まず, 効率性についての考察を行う。

図 10 より, 最終 step 数が 1000 回以内に収まるのが,0.001 $< alpha <$ 1.0 の範囲であることがわかる。このことから,alpha 値が小さすぎると探索できる範囲が狭まり, 最適解まで届かなかったと推察する。 図 11 からは, 総 step 数が 1 番低い点の alpha 値が 0.5 であり,0.5 から離れるごとに step 数が増加する傾向にあることがわかる。alpha 値が 0.5 の時に総 step 数が最小となった理由を, プログラム内の探索点移動に用いられている式より考察する。

$x = x - alpha * pd_x(x, y);$

この式の pd_x(x,y) は x についての偏微分であるため, 置き換えると

$x = x - alpha * 2x;$

となる。この式に alpha $= 0.5$ を代入すると x には '0' が入ることとなるため, 移動後の x 座標がちょうど傾きが 0 になる点 (最終的に移動したい最適解) になる。y についても同様にして alpha $= 0.5$ の時に 1 回移動後の座標は'0' となる。

このことから, alpha = 0.5 という値は 今回探索した $x^2 + y^2$ の式においてあらゆる座標から最適解の座標を求めることができる値であることがわかる。

次に, 最適性について
効率性の考察にて図 10 より, 最終 step 数が 1000 回となるのが $alpha = 0.001, alpha = 1.0$ で, 言い換えると $alpha = 0.001, alpha = 1.0$ のとき最適解との誤差が大きい。
このことが, 終了座標誤差を表す図 12 からも読み取れる。図 13 からは効率性と同じく alpha=0.5 に最適性が最も高いことがわかる。さらに alpha=0.5 の点へのグラフの収束の度合いにも alpha が 0.5 より小さい時と大きい時で差がある。これは,alpha が 0.5 より小さい場合の最適解からの誤差は探索点が単純に最適解に届かなかったものであるが,alpha が 0.5 より大きい場合では最適解付近まで探索点が到達したが探索点の移動幅の大きさが影響し最適解には至らなかったというもの, という差によるものと推察する。

2.5 Level2.3: $y = x * \cos(x)$ について

Level2.3 では $y = x * \cos(x)$ を最急降下法で探索する. 最急降下法では微分した値をアルゴリズムで必要とする為まずはこのモデル式の微分を導出する.

$$y = x * \cos(x) \tag{1}$$
$$y' = \cos(x) - x * \sin(x) \tag{2}$$

2.5.1 プログラムソース（変更部分）

上記 (2) 式で導出した微分値を利用して steepest-decent.c を以下の様に変更した. (コード:1) 尚変更点の列挙には元のソースコードが入っていた steepestsearch2-2/steepest_decent.c と今回のコードを diff コマンドを用いた. 変更した箇所を + 記号で示している.

ソースコード 1　変更点

```
--- ../steepestsearch2-2/steepest_decent.c 2017-01-20 15:59:51.000000000 +0900
+++ steepest_decent.c 2017-01-23 16:44:10.000000000 +0900
@@ -4,8 +4,8 @@

 #define X_MAX 10.0 /* 定義域の最大値 */
 #define X_MIN -10.0 /* 定義域の最小値 */
-#define Y_MAX 10.0 /* 定義域の最大値 */
-#define Y_MIN -10.0 /* 定義域の最小値 */
+#define Y_MAX 10.0 /* 値域の最大値 */
+#define Y_MIN -10.0 /* 値域の最小値 */
 #define X_RANGE (fabs(X_MAX)+fabs(X_MIN))
 #define Y_RANGE (fabs(Y_MAX)+fabs(Y_MIN))
 #define SAME 0.0000001 /* 探索点の動作チェック */
@@ -18,7 +18,7 @@


 void usage(){
-  fprintf(stderr," Usage : prompt> ./a.out random-seed\n");
+  fprintf(stderr," Usage : prompt> ./a.out random-seed (alpha)\n");
   exit(0);
 }

@@ -28,8 +28,9 @@
 double f(double x, double y) {
   double z;

-  /** 以下の式を編集して完成させよ (1) **/
```

```
28 - z = x;
29 +//モデル式は z=x*cos(x)であるのでここに設定する．
30 +
31 + z = x * cos(x);
32
33   return( z );
34  }
35 @@ -40,8 +41,12 @@
36  double pd_x(double x, double y) {
37    double z_dx;
38
39 - /** 以下の式を編集して完成させよ (2-1) **/
40 - z_dx = 1;
41 + /*
42 +  * z = x * cos(x)をx で微分すると
43 +  * z' = cos(x) - x *sin(x)になるのでここに設定
44 +  * **/
45 +
46 + z_dx = cos(x) - x*sin(x);
47
48   return( z_dx );
49  }
50 @@ -52,7 +57,9 @@
51  double pd_y(double x, double y) {
52    double z_dy;
53
54 - /** 以下の式を編集して完成させよ (2-2) **/
55 + /*
56 +  * z = x * cos(x)では関係がないので 0に設定しておく
57 +  * */
58   z_dy = 0;
59
60   return( z_dy );
61 @@ -71,9 +78,16 @@
62       */
63      int term_cond = 1000; /* 終了条件(繰り返し数) */
64
65 +    //デフォルトでは引数が 1つであるが学習レートを引数で変更出来るようにする
66 +
67      int seed;
68 -    if( argc != 2 ){
69 -        usage();
70 +    if( argc != 2 && argc != 3){
71 +        usage();  //引数が 1つもない or 3つ以上の場合はエラー
72 +    }else if ( argc == 3){
73 +        seed = atoi(argv[1]);
74 +        srand(seed); //引数が 2つの場合は第一引数をseed に設定
75 +        rand();   //第二引数は double 型に変換し alpha に設定
76 +        alpha = atof(argv[2]);
77      }else{
78          seed = atoi(argv[1]);
79          srand(seed);
```

軽微な修正であるが,y 軸に対して設定するは値域であると考えたので, 定義域から値域に変更した．続いて, 後述するが引数を 1 つ以上取るように設定したので,usage のメッセージを変更した．

diff 結果の 29 行目以降では, モデル式の表現として,C 言語の cos 関数と x を乗算し, 結果を z に代入する様にコードを変更した．36 行目から示す関数 pd_x 及び pd_y の変更点であるが, まず $y = x * \cos(x)$ を実験班で偏微分したところ, 得られた解を表現した．実際に x で偏微分をした結果を代入する z_dx には,cos,sin 関数をそれぞれ利用している．pd_y は偏微分した結果が 0 であるので初期状態のまま手を付けなかった．

学習係数 alpha を引数として変更できる用,argv のエラーメッセージを選択する if 文を変更した．コマンドライン引数の総数が 2 つある場合, 第 2 引数として alpha を受取り,char 型から double 型への変換を行っている．

今回の流れをフローチャートに示す．尚赤線部分が変更箇所である．(図:14)

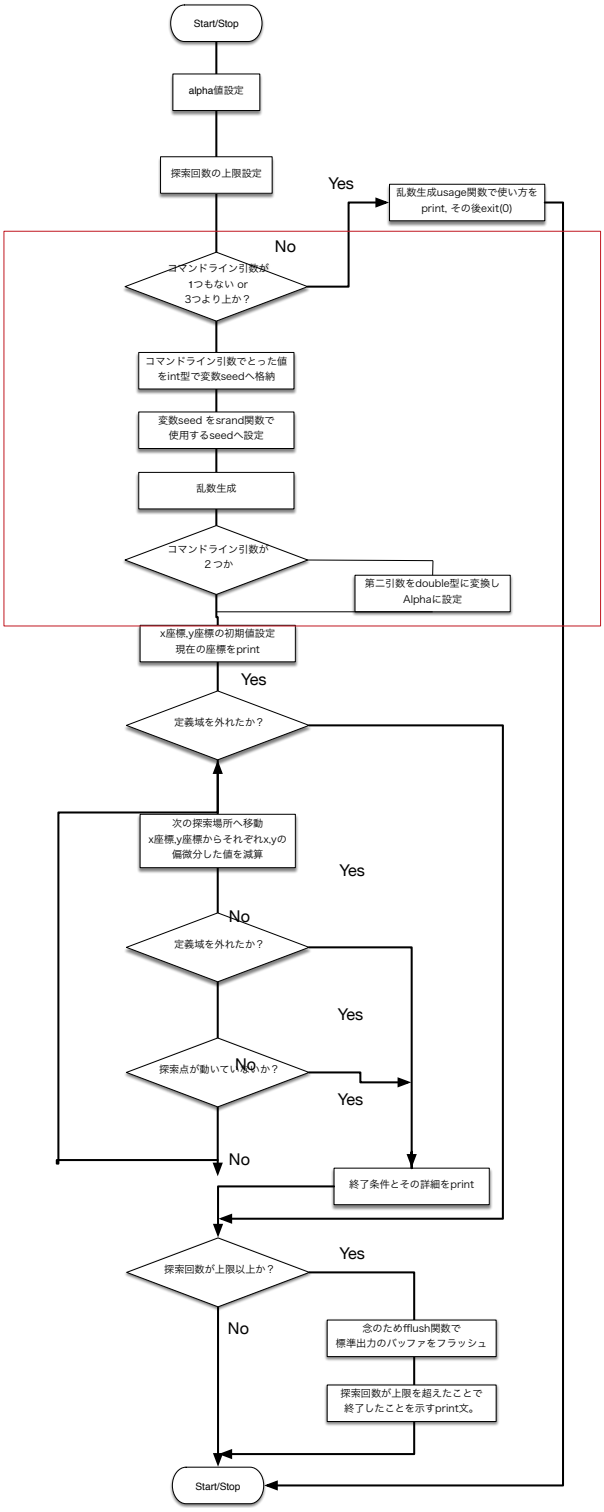

図 14　level2.3 フローチャート図

2.5.2　実験意図と観察方法

最初に学習係数が探索に影響をあたえる物が何かを決定づけるため,seed を固定して学習係数 Alpha のみ変更を行う．実験では, 最初に効率に着目し step の回数を比較する．続いて, 正確性を判断するためにシェルスクリプトを用いてモデル式の上に結果を plot し, 確認を行った．

続いて seed がどのような影響を与えるかを観察する為,Alpha 値を固定し seed を変更して観察を行った．内容は Alpha と同様に,step 回数, 及びモデル式との比較を行った．

2.5.3 Alpha の変更実験

まず Alpha の値を変更した場合,step 数にどう変化があるかを実験する．今回はシェルスクリプトを用いて自動化し, 実験を行った．

スクリプトは run_ave.sh を一部改変した run_ave_alpha.sh を利用した為, 本レポートでは変更点を示す． (コード:2) 尚ファイル名は average が入っているが, 今回は平均値を取ってはおらず, 複数回のループ処理の雛形として流用した．

ソースコード 2　run_ave.sh と run_ave_alpha.sh の差分

```
--- run_ave.sh 2017-01-20 15:59:51.000000000 +0900
+++ run_ave_alpha.sh 2017-02-03 18:08:58.000000000 +0900
@@ -1,38 +1,56 @@
 #!/bin/sh
 set -e

-# steepest_decent をシード値(=初期探索点)を変えて 10回実行し、
+# steepest_decent を Alpha(学習係数)を変更後,20回実行
 # 収束するのに要した平均探索回数を算出 。
+# FINISH2 以外のものでも集計しているので,記録としての正確性にはやや疑問が残る

 exec_file="./steepest_decent"
 average_file="./average.txt"
+result_file="./result.txt"
+pdf_title="./alpha_ave.pdf"
+seed=1000

 if [ -f $average_file ] ; then
     rm $average_file
 fi

-# シード値を下記 10パターンで試す。
-seeds="1000 2000 3000 4000 5000 6000 7000 8000 9000 10000"
+if [ -f $result_file ] ; then
+    rm $result_file
+fi
+
+# 20パターンで試す。
+roops=()
+
+i=0
+
+while [ $i -lt 20 ]
+do
+    i=`expr $i + 1`
+    roops[$i]=$i
+done
+#roops="1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 "
 sim=0
-for seed in $seeds
+alpha=0.01
+
+for roop in ${roops[@]}
 do
     sim=`expr $sim + 1`
-    $exec_file $seed > .archive-$seed
+    $exec_file $seed $alpha > .archive-$seed
     # シミュレーション結果から試行回数 10回分を抜き出す。
     tail -1 .archive-$seed | cut -f2 -d" " >> $average_file
-    /bin/echo -n "sim $sim: seed=$seed -> step="
-    /bin/echo `tail -1 $average_file`
```

```
+    /bin/echo -n "$alpha `tail -1 $average_file`" >> $result_file
+    #改行を挿入
+    /usr/bin/printf "\n" >>$result_file
+    #0.1ずつ足していく
+    alpha=`echo "$alpha"|awk '{print $1 +0.1}'`
 done

-# 平均試行回数を計算
-steps=`cat $average_file`
-sum=0
-for step in $steps
-do
-    sum=`expr $sum + $step`
-done
-/bin/echo -n "average step = "
-bc<<EOF
-scale=2;
-$sum/10
+gnuplot<<EOF
+set term pdf
+set xlabel "Alpha"
+set ylabel "Step"
+set output "${pdf_title}"
+plot "${result_file}" w linespoints
 EOF
+
```

今回は結果を出力するテキストファイルを変数 result_file,gnuplot での出力 pdf 名を pdf_title でそれぞれ設定している．また,seed 値を固定し, マジックナンバーを使用しない為に, 変数 seed に 1000 を設定している．

元のシェルスクリプトを参考にし, 結果を書き出すテキストファイルがあれば削除するように変更した．シェルスクリプト内でループをさせる場合, 全要素を書き込んでループさせる事も考えられる．しかし, 今回は 20 パターン計測をする事を考えている．この場合, 学習係数 Alpha の候補全要素を書き込むのは冗長と判断した．故に while 分で初期化しながら, 配列 roops[3] を利用した．

今回は配列 roops に収納した変数 20 個分 Alpha の初期値を 0.01 とし,0.1 づつインクリメントしながら計測を行う．計測をした結果を result_file で設定したテキストファイルに書き込み,gnuplot を用いて pdf 形式でグラフ化している．

このスクリプトを用いてグラフ化したものと, その際の実行結果を示す. (グラフ:図 15 , 実行結果:コード 3)

ソースコード 3　run_ave.sh の実行結果

```
$sh run_ave_alpha.sh
FINISH 3 step 724 x and y were not updated.
FINISH 3 step 72 x and y were not updated.
FINISH 3 step 36 x and y were not updated.
FINISH 3 step 22 x and y were not updated.
FINISH 3 step 15 x and y were not updated.
FINISH 3 step 11 x and y were not updated.
FINISH 3 step 16 x and y were not updated.
FINISH 3 step 24 x and y were not updated.
FINISH 3 step 44 x and y were not updated.
FINISH 3 step 118 x and y were not updated.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 2 step 13 x_rearched_to -13.8494155810 y_reached_to 2.1064439007 f(x,y) -3.9304192505
FINISH 2 step 33 x_rearched_to -13.4946134992 y_reached_to 2.1064439007 f(x,y) -8.0865337472
FINISH 2 step 20 x_rearched_to -13.6502008678 y_reached_to 2.1064439007 f(x,y) -6.3875677218
```

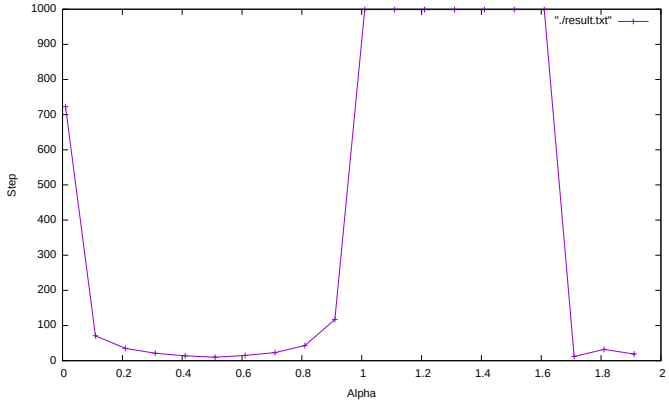

図 15　run_ave.sh の実行結果

step 数という観点から見ると,2 箇所で如実な減少が見られた．しかし, 今回のプログラムでは探索が正常に終了していない場合も含めている．実行結果を見ると,FINISH 2 で終了しているものは後半の 3 つしかなく, その中でも良い値は Alpha の値を 1.71 にした場合であった．

続いて実際に探索点がどのように動いたのか, 精度の面から確認する．まずは探索結果を可視化するために, スクリプトを生成した．今回は trans_x_vs_func.sh を参考に,run_ave_alpha.sh を改良した.(ソースコード:4)

ソースコード 4　trans_alpha.sh の変更点

```
--- run_ave_alpha.sh 2017-02-03 18:08:58.000000000 +0900
+++ trans_alpha.sh 2017-02-03 19:50:24.000000000 +0900
@@ -29,19 +29,16 @@
     i=‘expr $i + 1‘
     roops[$i]=$i
 done
-#roops="1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 "
 sim=0
 alpha=0.01

 for roop in ${roops[@]}
 do
     sim=‘expr $sim + 1‘
-    $exec_file $seed $alpha > .archive-$seed
-    # シミュレーション結果から試行回数 10回分を抜き出す。
-    tail -1 .archive-$seed | cut -f2 -d" " >> $average_file
-    /bin/echo -n "$alpha ‘tail -1 $average_file‘" >> $result_file
-    #改行を挿入
-    /usr/bin/printf "\n" >>$result_file
+    $exec_file $seed $alpha > .archive-$sim
+    archive_file=.archive-$sim
+    data_file=.data-$alpha.txt
+    cat $archive_file | cut -f4,8 -d" " > $data_file
     #0.1ずつ足していく
     alpha=‘echo "$alpha"|awk ’{print $1 +0.1}’‘
 done
@@ -51,6 +48,5 @@
 set xlabel "Alpha"
 set ylabel "Step"
 set output "${pdf_title}"
-plot "${result_file}" w linespoints
+plot ".data-0.41.txt" w linespoints title "0.41",".data-0.51.txt" w linespoints title "0.51", ".data-0.61.txt" w linespoints title "0.61", ".data-1.71.txt" w linespoints title "1.71",".data-1.81.txt" w linespoints title "1.81",".data-1.91.txt" w linespoints title "1.91", x*cos(x) w linespoints
 EOF
-
```

trans_x_vs_func.sh からは,cut コマンドで分割しテキストファイルに plot するデータを使用する部分のみに改良した．

大まかな処理は run_ave_alpha.sh と同様である.

20 件のデータをまず plot することを考え, ソースコード (4) に一部加筆を行い, グラフ化を行った. ソースコード (4) との変更点は,gnuplot の plot 呼び出しに対して, 収集した全データを記述した点である.

全データを plot した結果を示す. (図:16) また, プロンプトに表示された実行結果も併記する. (ソースコード:5)

ソースコード 5　trans_alpha.sh の実行結果

```
$sh trans_alpha.sh
FINISH 3 step 724 x and y were not updated.
FINISH 3 step 72 x and y were not updated.
FINISH 3 step 36 x and y were not updated.
FINISH 3 step 22 x and y were not updated.
FINISH 3 step 15 x and y were not updated.
FINISH 3 step 11 x and y were not updated.
FINISH 3 step 16 x and y were not updated.
FINISH 3 step 24 x and y were not updated.
FINISH 3 step 44 x and y were not updated.
FINISH 3 step 118 x and y were not updated.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 1 step 1000 this trial couldn't be search enough under the term_cond=1000.
FINISH 2 step 13 x_rearched_to -13.8494155810 y_reached_to 2.1064439007 f(x,y) -3.9304192505
FINISH 2 step 33 x_rearched_to -13.4946134992 y_reached_to 2.1064439007 f(x,y) -8.0865337472
FINISH 2 step 20 x_rearched_to -13.6502008678 y_reached_to 2.1064439007 f(x,y) -6.3875677218
```

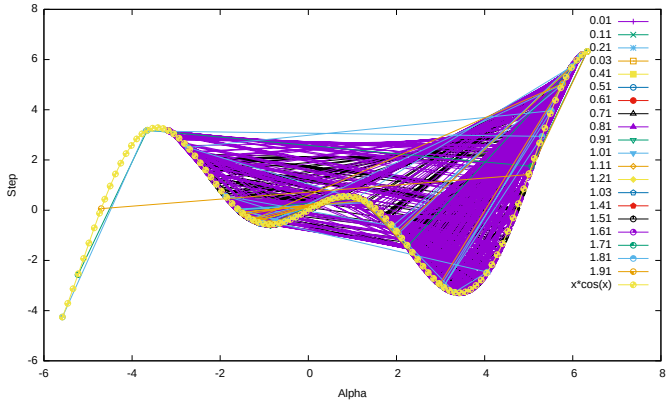

図 16　trans_alpha.sh を用いた全データの実行結果

データ件数が 20 件と多かった事と, 動きがやや複雑なものがあり, 動きを推測出来ないグラフとなってしまった. そこで, 先程 step 数が少なかった 0.41,0.51,0.61,1.71,1.81,1.91 とモデル式のみをを plot し確認を行った.

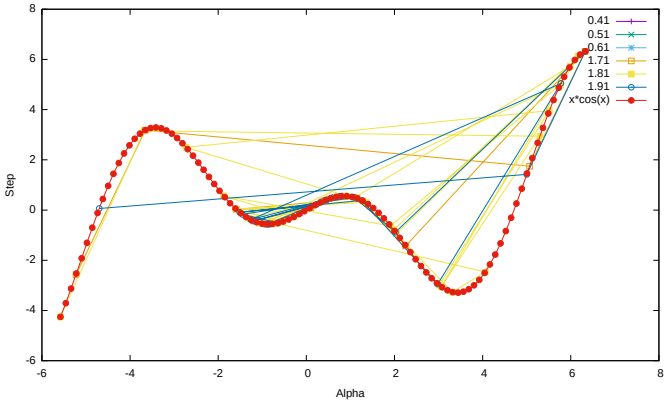

図 17　trans_alpha.sh を用いたデータの実行結果

動きとしてはやや難があるが,1.81 が比較的モデル式に沿って移動していると判断できる．この事から, 今回の結果では 1.81 が学習係数 Alpha としてふさわしいと考える．

2.5.4　seed の変更実験

続いて seed について変更した結果どう変わるかを実験する．今回は先程作成した run_ave_alpha.sh を seed 用に変更した．(ソースコード:6)

ソースコード 6　run_seed.sh の変更点

```
--- run_ave_alpha.sh 2017-02-03 18:08:58.000000000 +0900
+++ run_seed.sh 2017-02-03 21:01:49.000000000 +0900
@@ -1,15 +1,15 @@
 #!/bin/sh
 set -e

-# steepest_decent を Alpha(学習係数)を変更後,20回実行
+# steepest_decent を Seed 変更後,20回実行
 # 収束するのに要した平均探索回数を算出。
 # FINISH2 以外のものでも集計しているので,記録としての正確性にはやや疑問が残る

 exec_file="./steepest_decent"
-average_file="./average.txt"
-result_file="./result.txt"
-pdf_title="./alpha_ave.pdf"
-seed=1000
+average_file="./seed_average.txt"
+result_file="./seed_result.txt"
+pdf_title="./seed_ave.pdf"
+Alpha=0.1

 if [ -f $average_file ] ; then
     rm $average_file
@@ -31,7 +31,7 @@
 done
 #roops="1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 "
 sim=0
-alpha=0.01
+seed=1000

 for roop in ${roops[@]}
 do
@@ -39,16 +39,16 @@
     $exec_file $seed $alpha > .archive-$seed
     # シミュレーション結果から試行回数 10回分を抜き出す。
     tail -1 .archive-$seed | cut -f2 -d" " >> $average_file
```

```
-    /bin/echo -n "$alpha `tail -1 $average_file`" >> $result_file
+    /bin/echo -n "$seed `tail -1 $average_file`" >> $result_file
     #改行を挿入
     /usr/bin/printf "\n" >>$result_file
     #0.1ずつ足していく
-    alpha=`echo "$alpha"|awk '{print $1 +0.1}'`
+    seed=`echo "$seed"|awk '{print $1 +1000}'`
 done

 gnuplot<<EOF
 set term pdf
-set xlabel "Alpha"
+set xlabel "Seed"
 set ylabel "Step"
 set output "${pdf_title}"
 plot "${result_file}" w linespoints
```

今回は seed 値の挙動を観察するため, 初期値として 1000 を与え, ループのたびに 1000 インクリメントしていく方法に変更した．実際にこのスクリプトを実行した結果得られたグラフを示す． (図:18)

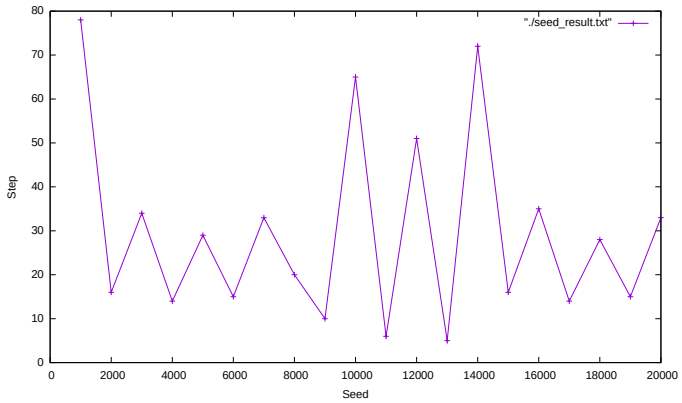

図 18　run_seed.sh の実行結果

このグラフから,seed 値を 1000 区切りで変更すると step 値が上昇と下降を繰り返す結果が読み取れた．この中では 11000 から 13000 ほどの間が尤も短い step 数であった為, ここに絞ってより細かく調査を行った．

調査用スクリプトとして範囲とループ数を変更したものを用意した． (ソースコード:7)

ソースコード 7　run_seed_seed.sh の変更点

```
--- run_seed.sh 2017-02-03 21:01:49.000000000 +0900
+++ run_seed_seed.sh 2017-02-03 21:05:22.000000000 +0900
@@ -6,9 +6,9 @@
 # FINISH2 以外のものでも集計しているので,記録としての正確性にはやや疑問が残る

 exec_file="./steepest_decent"
-average_file="./seed_average.txt"
-result_file="./seed_result.txt"
-pdf_title="./seed_ave.pdf"
+average_file="./see_seed_average.txt"
+result_file="./see_seed_result.txt"
+pdf_title="./see_seed_ave.pdf"
 Alpha=0.1

 if [ -f $average_file ] ; then
@@ -24,14 +24,14 @@

 i=0

```

```
-while [ $i -lt 20 ]
+while [ $i -lt 40 ]
 do
     i=‘expr $i + 1‘
     roops[$i]=$i
 done
 #roops="1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 "
 sim=0
-seed=1000
+seed=11000

 for roop in ${roops[@]}
 do
@@ -43,7 +43,7 @@
     #改行を挿入
     /usr/bin/printf "\n" >>$result_file
     #0.1ずつ足していく
-    seed=‘echo "$seed"|awk ’{print $1 +1000}’‘
+    seed=‘echo "$seed"|awk ’{print $1 +500}’‘
 done

 gnuplot<<EOF
```

今回は初期値を 11000 に設定し,500 ずつ 40 回インクリメントを行い挙動を確認した．このスクリプトを実行した結果,得られたグラフを示す．

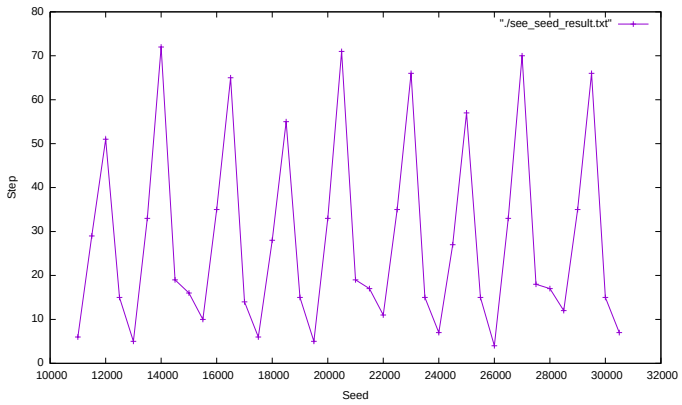

図 19　run_seed_seed.sh の実行結果

先程と同様に,STEP 数が上昇と下降を繰り返しをしていることが確認出来た．この結果から,Seed 値 26000 が今回では有益なものであると考える．尚このスクリプトは終了条件を特に設けていない為, 得られた値が正しいものとは限らない．

では, 今回 seed 値に応じてどのような探索経路を辿ったかを確認する．今回は Alpha の実験同様にシェルスクリプトを用いて実験を行った．利用したシェルスクリプトを示す．(ソースコード:8)

ソースコード 8　truns_seed.sh

```
#!/bin/sh
set -e

# steepest_decent を Seed を変更後,20回実行
# 収束するのに要した平均探索回数を算出。
# FINISH2 以外のものでも集計しているので,記録としての正確性にはやや疑問が残る

exec_file="./steepest_decent"
average_file="./seed_trans.txt"
result_file="./seed_trans.txt"
pdf_title="./seed_total.pdf"
alpha=0.1
```

```
13
14 if [ -f $average_file ] ; then
15     rm $average_file
16 fi
17
18 if [ -f $result_file ] ; then
19     rm $result_file
20 fi
21
22 # 20パターンで試す。
23 roops=()
24
25 i=0
26
27 while [ $i -lt 20 ]
28 do
29     i=‘expr $i + 1‘
30     roops[$i]=$i
31 done
32 sim=0
33 seed=1000
34
35 for roop in ${roops[@]}
36 do
37     sim=‘expr $sim + 1‘
38     $exec_file $seed $alpha > .archive-$sim
39     archive_file=.archive-$sim
40     data_file=.data-$seed.txt
41     cat $archive_file | cut -f4,8 -d" " > $data_file
42     #1000ずつ足していく
43     seed=‘echo "$seed"|awk ’{print $1 +1000}’‘
44 done
45
46 gnuplot<<EOF
47 set term pdf
48 set xlabel "Step"
49 set ylabel "f(x)"
50 set output "${pdf_title}"
51 plot ".data-1000.txt" w linespoints title "1000",".data-6000.txt" w linespoints title "6000", ".data
       -11000.txt" w linespoints title "11000", ".data-12000.txt" w linespoints title "12000",".data
       -13000.txt" w linespoints title "13000",".data-17000.txt" w linespoints title "17000", x*cos(x) lc
       rgb "red"w lines
52 EOF
```

ほぼ構造はこれまで使ってきたシェルスクリプトと同じであり, 今回は全てのデータではなく, ピックアップした幾つかの結果をグラフ化した．このシェルスクリプトを実行した結果, 出力されたグラフを示す． (グラフ:20)

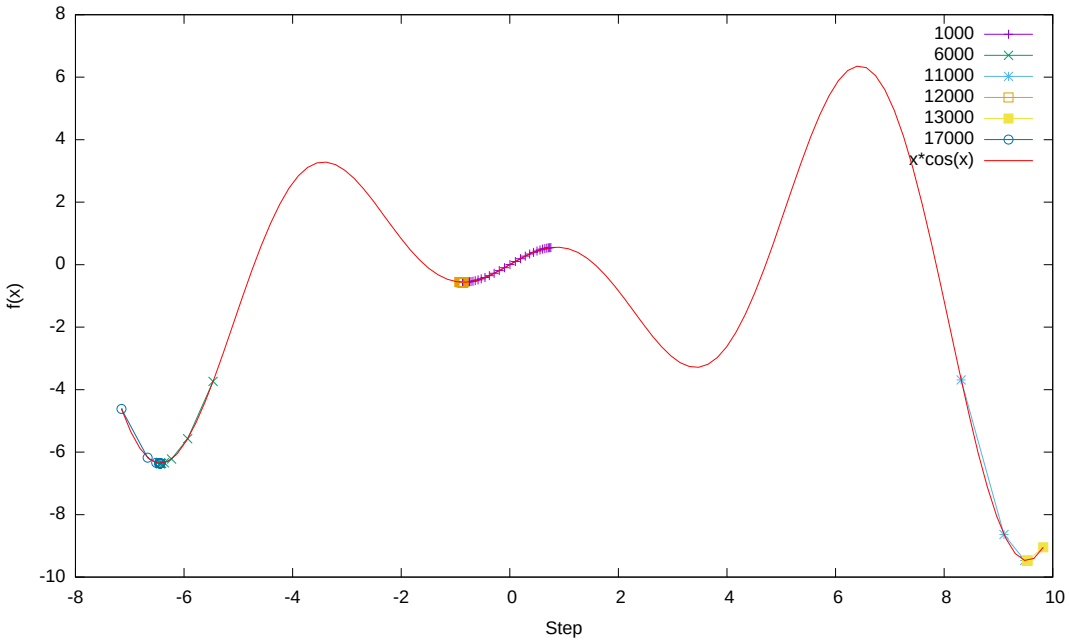

図 20　seed 値による探索点の推移

結果から,seed 値によっては, 最終的に探索を終了する最小値に行き着いていない箇所があることが解る．今回の範囲での最小値 (Step が 8 から 9 の間) に収まったのは,Seed 値 11000 と 1300 のみであった．

2.5.5　考察

今回の実行結果を踏まえると,Alpha の値を変えた場合, 探索点がたどる経路が異なってくると推測できる．Alpha の値が小さいほど, モデル式と類似した経路を辿り最小値に行き着く可能性が高い．ただし,Step 数と探索自体の正確性では, やや疑問の残る結果となっしまった．今回の結果では, 極端に Step 数が多い箇所, 少ない箇所が観察できた．従って, 正確性を保ちつつ Step 数を少なくするには極めて細やかなチューニングが必要であるとも考える．

seed 値の変更では, 最終的に得られる f(x) の値が異なる事が読み取れた．これは最急降下法のアルゴリズムが, 一度最小値と思われる部分．つまり切片の傾きが 0 の所に来てしまった場合, そこを最小値と認識し探索を終了することが原因していると考える．今回のモデル式である $f = x * \cos(x)$ は, この切片の傾きが 0 である箇所が複数ある．その為, 最急降下法を用いて正確な実験を行う場合初期探索位置を複数の場所に設定し, 探索をすることが有効であると考える．また, 一度探索が終了する条件になった際に分岐させ, 現在の x 座標からある一定の数座標軸をずらし探索を行う．その結果得られた解と分岐前の解を比較し, より良い結果を採用するなどの工夫が必要になってくると考えた．

参考文献

[1] 情報工学実験 2: 探索アルゴリズムその 1（當間）
http://www.eva.ie.u-ryukyu.ac.jp/~tnal/2016/info2/search1/

[2] Housing Data Set
http://archive.ics.uci.edu/ml/datasets/Housing

[3] シェルスクリプトの基礎知識まとめ
http://qiita.com/katsukii/items/383b241209fe96eae6e7#%E9%85%8D%E5%88%97