情報工学実験II レポート（探索アルゴリズム 1）

月曜日＆グループ 7

2014 年 12 月 15 日

グループメンバ

- 135711F 屋比久祐樹: 担当 Level1.1, 2.2
- 135713B 天願寛之: 担当 Level2.1
- 135717E 岡田和也: 担当 Level3.1, 4.1
- 135761B 大城海斗: 担当 Level1.2,2.3

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/2014-search1-mon/group7/” 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。

```
./          #レポート等
./figs/     #図表や実験に必要なプログラムなど.
```

1 Leve l1: 探索とは

1.1 Level1.1: 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。

- コンピュータはパターン化されたものを決まった動作を繰り返す。
- コンピュータは大量の情報を取り扱いや特定のパターンに基づいてまとめたりする。

- コンピュータの不得意なことは、感情移入判断することが出来ない、決まったことしか出来ない。
- コンピュータと人との違いは、パターン化されていないこともできる。

1.1.2 考察

- 視点 1:
  コンピュータは、パターン化されたことが得意

- 視点 2:
  人間は、パターン化されないことでも出来る

1.2 Leve1.2: 評価方法（目的関数の設計指針や方法）について

1.2.1 課題説明

Amazon における書籍検索時に「ファンタジー作品で泣ける作品」を探し出すためのアイテム集合 x と目的関数 $f(x)$ について検討した。

1.2.2 アイテム集合 x について

我々のグループでは，ファンタジー作品のレビューに書いてあるワードをアイテム集合 x とすることを考えた．次のようなレビューを例にとると，レビューに書いてあるすべての単語がアイテム集合 x になる．

> いやぁ、おもしろい。異世界ファンタジーと言うのは、最初の数ページでこれはちょっと、と思う場合と、そこで引き込まれてあっという間と言う場合の二種類がある気がするけど、本書はもちろん後者の方。ありえなぁいと叫ぶだけのような、そんな浅いファンタジーではなく、とてもとても重厚な、しっかり細部も練り込まれた、実に味わい深い作品でした。

1.2.3 目的関数について

目的関数 $f(x)$ はすべてのアイテム集合 x(レビュー内容の単語) に対して重み付けをし，それを足し合わせた合計を返す関数とする．先ほどのレビューを例にとると,「おもしろい」や「重厚な」,「味わい深い」などの作品を高く評価する単語に対しては他のワードより大きな値を付加する．そして探索目的は値が最大になるような $f(x)$ である (図 1 参照).

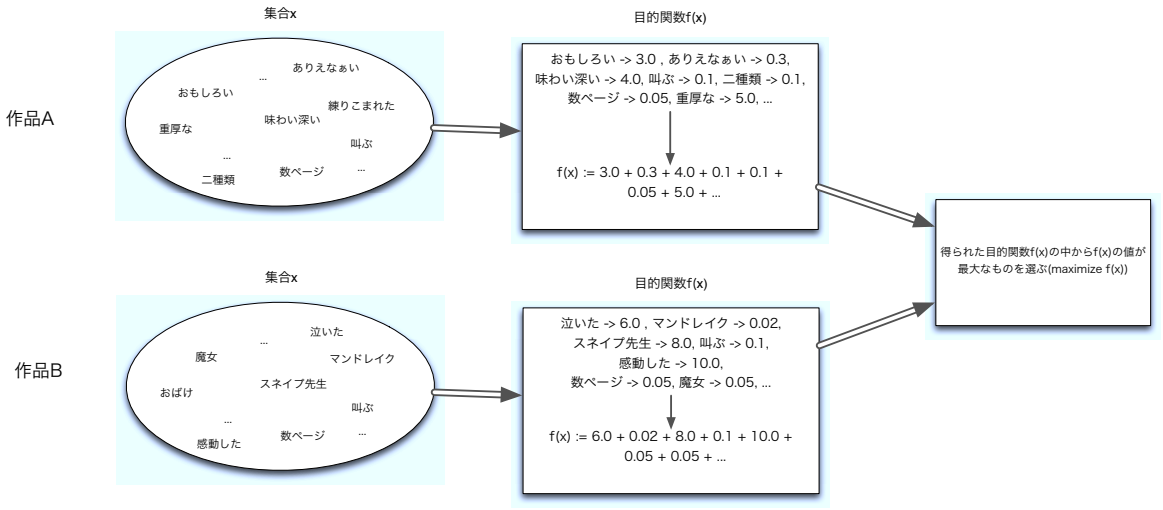

図 1: 泣ける作品を探し出すまでの流れ

1.2.4 目的関数の欠点

この目的関数の設計上の問題点はあらかじめワードとワードに対応する重みの値を定義しておく必要があるということである.

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 $y = x^2$、$z = x^2 + y^2$、$y = -x \times sin(x)$ 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。以下ではまず共通部分である最急降下法の探索手続きについて、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 探索の手続きとフローチャート（共通部分）

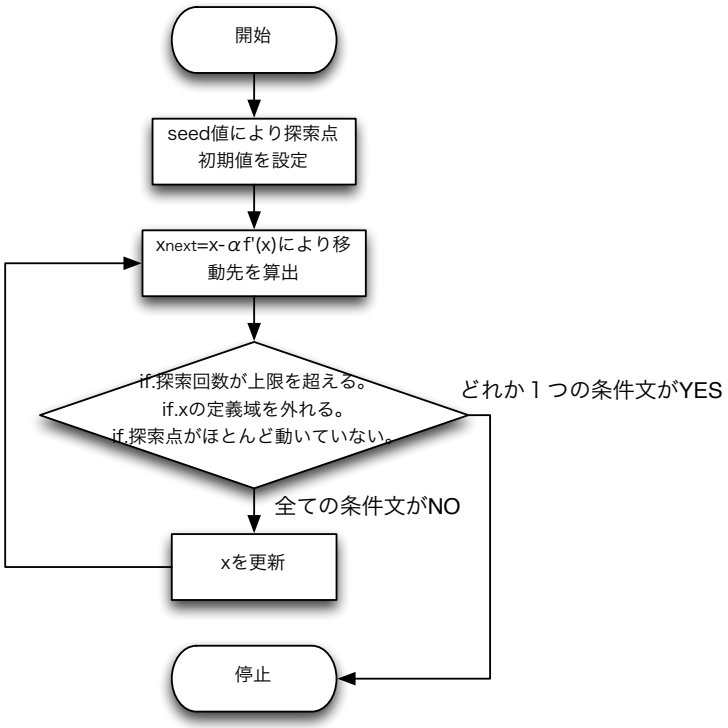

図 2: 探索手続きのフローチャート

2.3 Level2.1: $y = x^2$

2.3.1 プログラムソース

ソースコード 1: 変更された関数 f(x y)

```
double f(double x, double y) {
  double z;

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

  return( z );
}
```

ソースコード 2: 変更された f(x y) の微分値

```
double pd_x(double x, double y) {
  double z_dx;

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

  return( z_dx );
}
```

2.3.2 観察意図と観察方法 (実験計画)

変更したプログラムに対して、シード値を 1、α 値を 0.1 として実行する。run_ave.sh ファイルの実行結果により、シード値が及ぼす挙動についてみれるはずなので、その結果をもとにシード値の考察を行う。次に α 値により探索点の推移距離が変わるので、シード値を固定したまま α 値をより大きく、または小さくした値に変更して実験を行うことで α 値が探索挙動に及ぼす影響を考察する。この時、関数に対して谷が 1 つなので x の解や α 値が 1 つに定まるはずなのでそれを意識する。

2.3.3 実行結果

ソースコード 3: ˙steepest_decent 1 の実行結果

```
step 0 x -7.3692442371 y 5.1121064439 f(x,y) 54.3057606266 5.430576e+01
step 1 x -5.8953953897 y 5.1121064439 f(x,y) 34.7556868010 3.475569e+01
.
.
.
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.
```

シード値を 1、α 値を 0.1 に設定して実行した結果、x の値が-7.3692442371 から始まり、step63 で f(x) の値が 0、探索点がほとんど動かなくなった為 step75 で終了した。
./run_ave.sh を用いてシード値を千刻み 10 パターン（1000〜10000）で実行し平均試行回数を計測した。1000〜10000 までそれぞれの試行回数は順に 64、75、69、74、71、73、73、72、74、70 であり、平均試行回数は 71.5step であった。

図 1 と図 2 はシード値 1、α を 0.1 に設定して実行された最急降下法アルゴリズムにおける点の step あたりの移線グラフと関数上での推移グラフである。

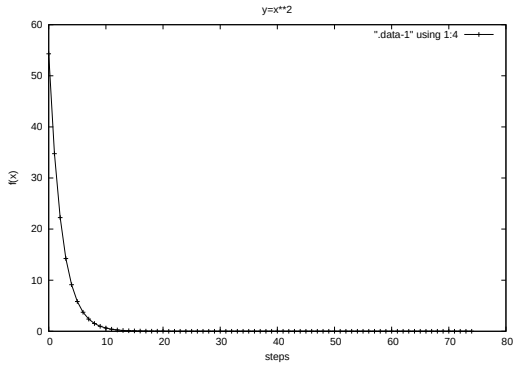

図 1 step あたりの目的関数推移線グラフ

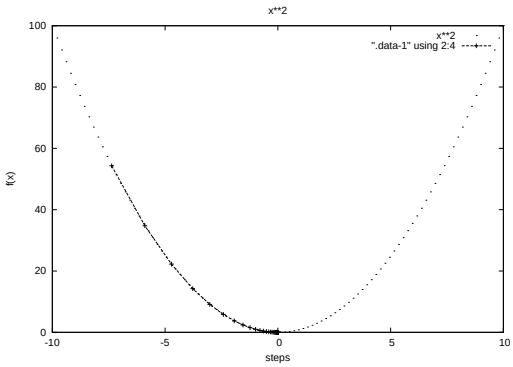

図 2 探索点が関数上をどのように推移したかを示すグラフ

2.3.4 考察

シード値は探索点の初期値の設定に用いるものである。実験結果よりシード値による高さの決定はランダムであるが、α値を固定した場合は、決定した高さがより低いほうが step 数は少ない事が分かった。さらにコマンド「tail -1 .archive-*」を実行した結果、高さと得られる解の質は関係がないと考えられた。
α値に関しての考察を得るためシード値 1、α値 0.1 における実験に対してα値を変更した 2 つの実験を行う。シード値を固定したままα値を 0.3、0.007 に変更する。シード値は固定であるため開始地点は同じになる。

ソースコード 4: α値 0.3、0.007 に置ける˙steepest_decent 1 の実行結果

```
1 α値
2  0.3
3 step 20 x -0.0000000810 y 5.1121064439 f(x,y) 0.0000000000 6.565164e-15
4 FINISH 3 step 21 x and y were not updated.
5
6 α値 0.007
7 step 983 x -0.0000070539 y 5.1121064439 f(x,y) 0.0000000000 4.975774e-11
8 FINISH 3 step 984 x and y were not updated.
```

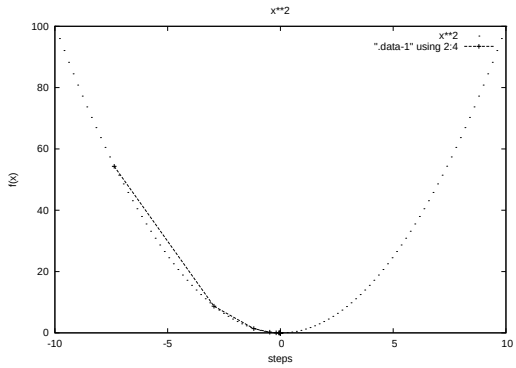

図 3 探索点の関数上での推移を示すグラフ (α値 0.3)

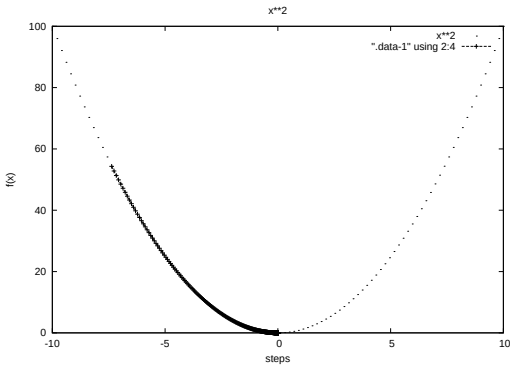

図 4 探索点の関数上での推移を示すグラフ (α値 0.007)

α値を変更する事で推移距離が変わる。α値を 0.3 にすると推移距離が長くなり、小さくすると短くなる。α値を 0.3 にすると、α値 0.007、0.1 に比べ、step21 で探索が済んでいるため効率性がよく、得られる解の質に関しても、誤差が小さくなっていることが実行結果より判断出来る。α値を小さくすると誤差がより大きくなってしまったことに関しては、小さいと探索点から動いていないと判断されてしまい、探索を打ち切られてしまうからと考えられる。
これらの実行結果より、3 つのα値の内 0.3 付近がより、探索における有用性が見いだせる。今回の場合、最小値の値が 0 であり、傾きの値が 0 となる点であることが自明である。よって移動先の算出は式から見いだせ、その値はα値が 0.5 のときである。つまりα値が 0.5 の時 step2 で最適値を見つけ出す事ができ、解の質においても誤差はない。今回の実験で、0.3 付近でより有用性が見いだせたのは、より 0.5 に近かったからと判断できる。

2.4 Level2.2: $z = x^2 + y^2$ について

2.4.1 プログラムソース（変更部分）

ソース変更：
/** 以下の式を編集して完成させよ (1) **/
$z = x * x + y * y$;
/** 以下の式を編集して完成させよ (2-1) **/
$z\_dx = 2 * x$;
/** 以下の式を編集して完成させよ (2-2) **/
$z\_dy = 2 * y$;
この三カ所である。

2.4.2 観察意図と観察方法

alpha の数字を変えてみて、どのようになるのかを確認して行く。
./trans_xy_vs_func.sh”x**2+y**2”1 を用いて、グラフの図を作成しどのようになっているのかを確認した。

2.4.3 実行結果

alpha = 0.1 の時
/Users/e135711/実験 2/steepestsearch ./trans_xy_vs_func.sh”x**2+y**2” 1
FINISH 3 step 75 x and y were not updated.

alpha =1.0 の時
/Users/e135711/実験 2/steepestsearch ./trans_xy_vs_func.sh ”x**2+y**2” 1
FINISH 1 step 1000 this trial couldn’t be search enough under the term_cond=1000.

2.4.4 考察

x 軸と y 軸から見て“ .data-1 ” using2:3:4 を確認すると、alpha の数を変えてみた結果、数字を上げて行くと計算回数が少なくなり、数字を下げると計算回数が多くなった。

2.5 Level2.3: $y = -x * sin(x)$ について

2.5.1 プログラムソース（変更部分）

Level2.3 では steepest_decent.c の f 関数と pd_x 関数を一部以下のように変更した.

f(x,y) 関数の変更点

```
double f(double x, double y) {
    double z;

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

    return( z );
}
```

pd_x(x,y) 関数の変更点

```
double pd_x(double x, double y) {
    double z_dx;

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

    return( z_dx );
}
```

2.5.2 観察意図と観察方法 (実験計画)

最急降下法は，$y = x^2$ のような「極小値=関数全体の最小値」となる関数に対しては非常に有効なアルゴリズムである．しかし，今回の $y = -xsin(x)$ のように極小値が複数ある場合は，探索開始地点から最寄りの極小値に収束しようとする．これは必ずしも関数の最小値だとは限らない．そのため，複数の初期値を与えて実行し，それぞれの収束したときの y の値の中から最も小さいものを選択すれば，それが関数の最小値となるだろうと考えた．

run_ave.sh をベースに試行回数 10 回分の結果の中から y の最小値を抜き出すスクリプト (ソースコード 5) を作成した．また学習係数 α を変えることにより，どのように step 数が変動するかを観察する．

ソースコード 5: y の最小値を求めるプログラム

```
#!/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、
# 最小値を算出

exec_file="./steepest_decent"
smallest_file="./smallest.txt"

if [ -f $smallest_file ] ; then
    rm $smallest_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"
for seed in $seeds
do
    $exec_file $seed > .archive-$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 -f8 -d" " >> $smallest_file
done

m=10
cat $smallest_file | awk '{if(m>$0) m=$0}{print m}' > hogefile
tail -1 hogefile
```

2.5.3 実行結果

今回試した学習係数 α は 0.075, 0.2 の 2 つである．まず，ソースコード 5 を実行した結果を示す．

$\alpha = 0.075$ の時に求まった $-xsin(x)$ の最小値

```
/Users/e135761/search/steep/steepestsearch% ./alter_run_ave.sh
FINISH 3 step 72 x and y were not updated.
FINISH 3 step 17 x and y were not updated.
FINISH 3 step 59 x and y were not updated.
FINISH 3 step 19 x and y were not updated.
FINISH 3 step 67 x and y were not updated.
FINISH 3 step 23 x and y were not updated.
FINISH 3 step 24 x and y were not updated.
FINISH 3 step 68 x and y were not updated.
FINISH 3 step 19 x and y were not updated.
FINISH 3 step 61 x and y were not updated.
-7.9167273716
/Users/e135761/search/steep/steepestsearch%
```

$\alpha = 0.2$ の時に求まった $-xsin(x)$ の最小値

```
/Users/e135761/search/steep/steepestsearch% ./alter_run_ave.sh
FINISH 3 step 24 x and y were not updated.
FINISH 3 step 36 x and y were not updated.
FINISH 3 step 19 x and y were not updated.
FINISH 3 step 35 x and y were not updated.
FINISH 3 step 22 x and y were not updated.
FINISH 3 step 38 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 34 x and y were not updated.
FINISH 3 step 20 x and y were not updated.
-7.9167273716
/Users/e135761/search/steep/steepestsearch%
```

特に α による最小値の違いは見られなかった．ここで，得られた最小値がどれほどの精度なのかを知るため，方程式の解の近似値を求める手法であるニュートン法 [3](ソースコード 6) で同様のことを行った．ここでいう方程式は，$y = -xsin(x)$ の導関数が 0 になる式，すなわち $y' = -sin(x) - xcos(x) = 0$ のことである．この方程式を解くことで関数が最小値をとるときの x を求めることができる．

ソースコード 6: ニュートン法による最小値算出

```
1  import scala.math._
2  object Newton {
3    val epsilon = 0.000001  /*誤差の許容範囲*/
4    def newton(a:Double, f:Double=>Double, f_dash:Double=>Double):Double = {
5      val b = a - f(a) / f_dash(a) /*次の移動先を求める*/
6      if (abs(b-a) < epsilon) b else newton(b,f,f_dash) /*許容誤差の範囲内なら移動先のx座標を表示．そうでなければ探索を続行
           */
7    }
8    def main(args: Array[String]) {
9      val x = newton(8.0, x=>x+tan(x), x=>1+(1/pow(cos(x),2))) /*解に近いであろう8.0を初期値として与える*/
10     println("x is " + x)
11     println("the minimum value is " + -x*sin(x))
12   }
13 }
```

ニュートン法による最小値の算出結果

```
/Users/e135761/scala% scala Newton
x is 7.978665712413194
the minimum value is -7.916727371587782
/Users/e135761/scala%
```

この実行結果は上記のようになり，最急降下法で得られた最小値はほぼ正確なものだと言える．

次に，run_ave.sh を用いて α が 0.075 と 0.2 のときのそれぞれの平均 step 数を調べたところ，$\alpha = 0.075$ のとき平均 step 数は 41.90 で，$\alpha = 0.2$ のときのそれは 27.60 であった．

2.5.4 考察

実験結果より，2 つの学習係数 α 間で最小値の精度に違いは見られず，かつ，ニュートン法との比較でそれは正確なものだと判断できるため，得られた解は解の質として好ましいものである．効率性に関しては，少ない step 数で最小値に収束している $\alpha = 0.2$ の方が効率的であることが分かる (図 3，図 4)．学習係数 α にあまりに小さい値をとると，解を求めるファクターとしての最適性は十分に有するものの，効率性に欠けるという欠点が生じる．反対に大きすぎる数値を学習係数に設定すると，定義域を外れたり，指定した探索回数内で終了しない原因になる．したがって学習係数 α には 0.2 付近の値を設定するのがよい．

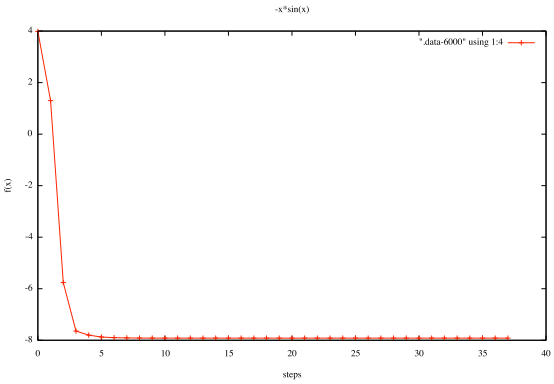

図 3: $\alpha = 0.2$ のときの目的関数推移図

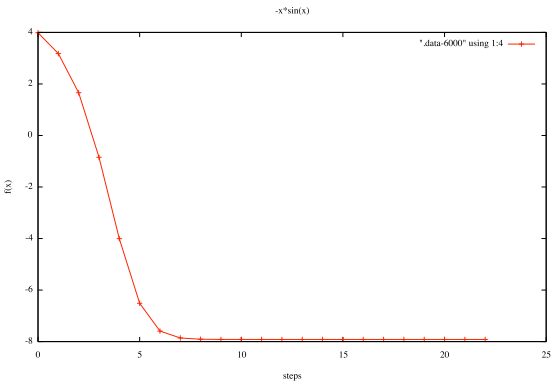

図 4: $\alpha = 0.075$ のときの目的関数推移図

3　Level 3: 最急降下法が苦手とする状況

3.1　最急降下法が苦手とする状況についてその理由を解説し、検討した改善方法について解説する。

3.1.1　原因

傾きが小さくなるほど移動距離が小さくなって検索点が多くなっている. 谷に直線的に向かわない.

3.1.2　改善方法

alpha を大きくとり, 移動距離を大きくする. 今居る場所より次に移動する場所を移動する前に比較して高かったら,α の値を下げもう一度比較していく.

4 Level 4: モデル推定時における目的関数の設計

Housing Data Set[2] を例に、モデルの適切さを図るための目的関数に付いて設計した。

4.1 目的関数について

設計した目的関数は最小化である. 点と線の最短距離の和が小さければ小さいほど適切. 点と線の横軸と縦軸の距離の和が小さければ小さいほど適切.

4.2 設計理由について

モデルと全ての点の差を出して差が小さい方がモデルとして適切なので, モデルとの差が分かるような目的関数にした.

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/2013/info2/search1/

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

[3] Newton 法による方程式の近似解法
http://www.math.u-ryukyu.ac.jp/~suga/C/2004/7/node9.html