情報工学実験II レポート（探索アルゴリズム 1）

金曜日＆グループ 7

平成 27 年 1 月 9 日

グループメンバ

- 135719 澤崎夏希: 担当 Level2.3, 3.1

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/2013-search1-mon/group0/” 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。

自分たちでやりやすいように Level 毎に整理しても構わない）

```
./scr/      # 作成したプログラム一式
./report/   # レポート関係ファイル．図ファイルを含む．
```

1 Leve l1: 探索とは

以下は『人工知能研究 [2]』を参考に記述した.

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。

1.1.2 考察

- 視点 1: 計算、処理速度
  数学などの単純計算において、人間は算盤などの道具を使っても、計算速度はコンピュータが勝る。
- 視点 2: 学習、推測
  コンピュータはある物事について完全に正しいデータを与えられないと、正しいルールを得ることができない。人間には勘や感情があるため、論理的に説明できないこともこなすことができる。
- 視点 3: 作業効率
  ベルトコンベアなどで行われる単純作業は、コンピュータ (ロボット) でも人力でも行われているが、人間には集中力などの観点から、長時間の作業は難しい。よって、生産量などの結果はコンピュータが勝る。

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

私達は、映画や書籍、ゲームからファンタジー作品について検討した。まずファンタジー作品とは、モンスターや魔法など現実に存在しないものが登場することや、想像上の物語が展開されていることが挙げられると考えた。

1.2.3 目的関数について

レビューから、「泣く」や「感動する」といった単語を抽出し、さらに、現実に存在しないものが登場することが目的関数になると考えた。これらに点数をつけ、どれほどファンタジー作品に近いかを点数から評価することができる。
しかし、「涙を流す」や「心が震える」など、「泣く」や「感動する」といった意味の単語が多く、「(感動する映画) を見ているようだった」など、遠回しに感動などを伝える表現なども存在する。これら全てを認識し、抽出することは難しい。

2 Level 2: 最急降下法による最適化

以下は『最急降下法 [3]』を参考に記述した.

2.1 Level2.1,2.2 共通部分 観察意図と観察方法

最急降下法の学習レートである alpha を任意で初期値, 刻み幅, 最大値を決め連続関数における探索プログラムを実行し, シード値 1000〜10000 における step 数とその平均を出力してくれるスクリプト alpha.sh を使用し,alpha を 0.01〜0.99 と刻み幅 0.01 で変更しながら step 数の平均を観察する. これらの方法により効率的に最適解 (最小値となる解) を求められる,alpha の値の発見および alpha の変化による探索への影響を考察することを目的とする.

2.2 Level2.1: $y = x^2$ について

2.2.1 プログラムソース（変更部分）

ソースコード 1: 探索プログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  ————————————省略————————————————
6
7  double f(double x, double y) {
8    double z;
9
10   /** 以下の式を編集して完成させよ(1) **/
11   z = x*x;
12
13   return( z );
14 }
15
16 /* f(x,y)/dx
17 *     z=f(x,y) の微分値(偏微分値)を求め, 返す.
18 */
19 double pd_x(double x, double y) {
20   double z_dx;
21
22   /** 以下の式を編集して完成させよ(2-1) **/
23   z_dx = 2*x;
24
25   return( z_dx );
26 }
27
28 /* f(x,y)/dy
29 *     z=f(x,y) の微分値(偏微分値)を求め, 返す.
30 */
31 double pd_y(double x, double y) {
```

```
32      double z_dy;
33
34      /** 以下の式を編集して完成させよ(2-2) **/
35      z_dy = 0;
36
37      return( z_dy );
38    }
39
40  ----------省略----------
41
42      return 0;
43    }
```

2.2.2 実行結果

ソースコード 2: シェルスクリプト alpha.sh 実行結果

```
1   alpha = 0.01
2   FINISH 3 step 592 x and y were not updated.
3   sim 1: seed=1000 -> step=591
4   FINISH 3 step 712 x and y were not updated.
5   sim 2: seed=2000 -> step=711
6   FINISH 3 step 646 x and y were not updated.
7   sim 3: seed=3000 -> step=645
8   FINISH 3 step 702 x and y were not updated.
9   sim 4: seed=4000 -> step=701
10  FINISH 3 step 671 x and y were not updated.
11  sim 5: seed=5000 -> step=670
12  FINISH 3 step 539 x and y were not updated.
13  sim 6: seed=609000 -> step=538
14  FINISH 3 step 688 x and y were not updated.
15  sim 7: seed=7000 -> step=687
16  FINISH 3 step 674 x and y were not updated.
17  sim 8: seed=8000 -> step=673
18  FINISH 3 step 701 x and y were not updated.
19  sim 9: seed=9000 -> step=700
20  FINISH 3 step 650 x and y were not updated.
21  sim 10: seed=10000 -> step=649
22  average step = 656.50
23
24  ----------省略----------
25
26  alpha = .50
27
28  FINISH 3 step 2 x and y were not updated.
29  sim 1: seed=1000 -> step=1
30  FINISH 3 step 2 x and y were not updated.
```

```
sim  2: seed=2000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  3: seed=3000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  4: seed=4000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  5: seed=5000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  6: seed=609000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  7: seed=7000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  8: seed=8000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  9: seed=9000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim 10: seed=10000 -> step=1
average step = 1.00

----------省略----------

alpha = .99
FINISH 3 step 819 x and y were not updated.
sim  1: seed=1000 -> step=818
FINISH 3 step 939 x and y were not updated.
sim  2: seed=2000 -> step=938
FINISH 3 step 874 x and y were not updated.
sim  3: seed=3000 -> step=873
FINISH 3 step 929 x and y were not updated.
sim  4: seed=4000 -> step=928
FINISH 3 step 899 x and y were not updated.
sim  5: seed=5000 -> step=898
FINISH 3 step 766 x and y were not updated.
sim  6: seed=609000 -> step=765
FINISH 3 step 916 x and y were not updated.
sim  7: seed=7000 -> step=915
FINISH 3 step 901 x and y were not updated.
sim  8: seed=8000 -> step=900
FINISH 3 step 928 x and y were not updated.
sim  9: seed=9000 -> step=927
FINISH 3 step 877 x and y were not updated.
sim 10: seed=10000 -> step=876
average step = 883.80
finish
```

ソースコード 3: 探索プログラム実行結果

```
%./steepest_decent2_1 1000
```

```
step 0 x 0.7557628633 y 2.1064439007 f(x,y) 0.1775056 5.711775e-01
step 1 x 0.0000000000 y 2.1064439007 f(x,y) 0.0000000000 0.000000e+00
FINISH 3 step 2 x and y were not updated.
```

※シード値:1000 学習レート α:0.5(最も平均 step 数が少なかった α の探索結果)

2.3 Level2.2: $z = x^2 + y^2$ について

2.3.1 プログラムソース（変更部分）

ソースコード 4: 探索プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

————————————省略————————————

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+y*y;

  return( z );
}

/* f(x,y)/dx
*     z=f(x,y) の微分値(偏微分値)を求め, 返す.
*/
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 );
}

/* f(x,y)/dy
*     z=f(x,y) の微分値(偏微分値)を求め, 返す.
*/
double pd_y(double x, double y) {
  double z_dy;

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

```

```
37      return( z_dy );
38    }
39
40    ――――――――――――――――省略――――――――――――――――
41
42      return 0;
43    }
```

2.3.2 観察意図と観察方法

2.3.3 実行結果

ソースコード 5: シェルスクリプト alpha.sh 実行結果

```
1   alpha = 0.01
2   FINISH 3 step 643 x and y were not updated.
3   sim 1: seed=1000 -> step=642
4   FINISH 3 step 712 x and y were not updated.
5   sim 2: seed=2000 -> step=711
6   FINISH 3 step 697 x and y were not updated.
7   sim 3: seed=3000 -> step=696
8   FINISH 3 step 702 x and y were not updated.
9   sim 4: seed=4000 -> step=701
10  FINISH 3 step 717 x and y were not updated.
11  sim 5: seed=5000 -> step=716
12  FINISH 3 step 690 x and y were not updated.
13  sim 6: seed=6000 -> step=689
14  FINISH 3 step 688 x and y were not updated.
15  sim 7: seed=7000 -> step=687
16  FINISH 3 step 701 x and y were not updated.
17  sim 8: seed=8000 -> step=700
18  FINISH 3 step 701 x and y were not updated.
19  sim 9: seed=9000 -> step=700
20  FINISH 3 step 714 x and y were not updated.
21  sim 10: seed=10000 -> step=713
22  average step = 695.50
23
24  ――――――――――――――――省略――――――――――――――――
25
26  alpha = .50
27  FINISH 3 step 2 x and y were not updated.
28  sim 1: seed=1000 -> step=1
29  FINISH 3 step 2 x and y were not updated.
30  sim 2: seed=2000 -> step=1
31  FINISH 3 step 2 x and y were not updated.
32  sim 3: seed=3000 -> step=1
33  FINISH 3 step 2 x and y were not updated.
```

```
sim  4: seed=4000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  5: seed=5000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  6: seed=6000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  7: seed=7000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  8: seed=8000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim  9: seed=9000 -> step=1
FINISH 3 step 2 x and y were not updated.
sim 10: seed=10000 -> step=1
average step = 1.00

----------省略----------

alpha = .99
FINISH 3 step 870 x and y were not updated.
sim  1: seed=1000 -> step=869
FINISH 3 step 939 x and y were not updated.
sim  2: seed=2000 -> step=938
FINISH 3 step 924 x and y were not updated.
sim  3: seed=3000 -> step=923
FINISH 3 step 929 x and y were not updated.
sim  4: seed=4000 -> step=928
FINISH 3 step 944 x and y were not updated.
sim  5: seed=5000 -> step=943
FINISH 3 step 917 x and y were not updated.
sim  6: seed=6000 -> step=916
FINISH 3 step 916 x and y were not updated.
sim  7: seed=7000 -> step=915
FINISH 3 step 928 x and y were not updated.
sim  8: seed=8000 -> step=927
FINISH 3 step 928 x and y were not updated.
sim  9: seed=9000 -> step=927
FINISH 3 step 942 x and y were not updated.
sim 10: seed=10000 -> step=941
average step = 922.70
```

ソースコード 6: 探索プログラム実行結果

```
%./steepest_decent2_2 1000
step 0 x 0.7557628633 y 2.1064439007 f(x,y) 5.0082834122 5.008283e+00
step 1 x 0.0000000000 y 0.0000000000 f(x,y) 0.0000000000 0.000000e+00
FINISH 3 step 2 x and y were not updated.
```

※シード値:1000 学習レート α:0.5(最も平均 step 数が少なかった α の探索結果)

2.4 Level2.1,2.2 共通部分 考察

作成したシェルスクリプト alpha.sh を実行した結果, 当初の予想と同じように学習レートアルファが小さければ詳細に探索でき, アルファが大きくなれば大雑把だが効率よく最適解付近にたどり着くことができることがわかった. 今回は,alpha を初期値, 刻み幅, 最大値をきめ動かしたがしらみつぶしに探索したが, 何カ所かの異なる alpha においての探索を行うことにより, ある程度効率の良い探索ができる学習レートを予想することができる.

2.5 Level2.3: $y = -x * sin(x)$ について

この関数はいくつかの峰と谷を持つ複雑な関数であり，谷を見つけてもそこが最低値かどうかは判断出来ない．そこで以下のようのソースコードを修正した.

2.5.1 プログラムソース（変更部分）

ソースコード 7: 変更部分

```
1  ----------...
2  83        n=4;
3  84        d=X_RANGE/n;
4  85
5  86        for (j=0; j < n; j++){
6  87          x = X_MIN+d*j + (X_RANGE/n) * (double)rand()/RAND_MAX;
7  ...----------
```

2.5.2 観察意図と観察方法

これは，最急降下法で観察する部分を分割することで，様々な場所の最小値を発見しようとしている．変更したのは初期値だけであるが，最急降下法は初期値の発生位置によって結果が大きく左右されるため十分であると判断した．観察は当初出力結果を観察していたが，データの推移が判断しにくいため gnuplot でのグラフとの比較を見て判断した．グラフの作成にはシェルスクリプト（スクリプト 2.3 参照）を用いた.

2.5.3 実行結果

2.5.4 考察

上の結果が示すように分割した位置で最急降下法を実行することにより，それぞれの谷の位置を発見することが出来た．今回は谷の数を考慮した n(分割数)=4 でプログラムを実行したが，現実的には n もパラメータとして定義し，都合の良い値を推測べきである.

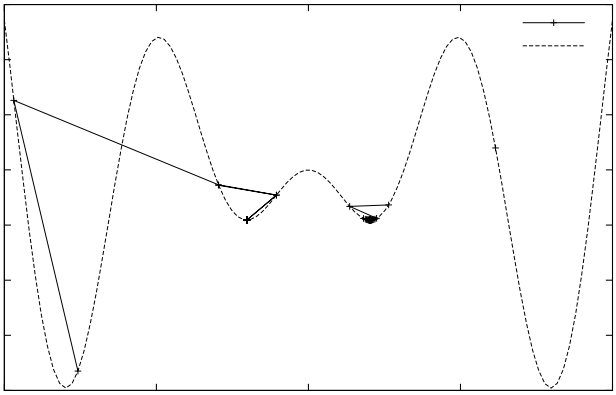

図 1: 修正されたプログラムによる出力結果

3 Level 3: 最急降下法が苦手とする状況

3.1 Level 3.1: $y = x^2 + (y^2)/10$ について

この関数は 1 つの谷を持つ関数であるが，最急降下法では上手く最小値を見つける事ができない．

3.1.1 原因

最急降下法は勾配ベクトルの逆を進み，山を下るような感覚で最小値を発見しようとするものである．しかし勾配ベクトルには，実際の最小値への方向とは誤差が含まれている．それを刻み値 α で修正するが，方向にずれが生じているため結果的にジグザグに動くことになる．これは x 方向と y 方向の勾配に差があるためであり，結果的には勾配が大きい方に重みが生じていることになる．

3.1.2 改善方法

この欠点を改善するためには「共役勾配法」という，前回の進行方向を加味して次の方向を決めるようなアルゴリズムである．これは単純に計算量が増えるものの比較的効率よく最小値を探索することができる．

4 Level 4: モデル推定時における目的関数の設計

以下は『最適化問題 [4]』参考に記述した．Housing Data Set[?] 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

モデルの関数を f(x)=10*x-40
とし、
幅を 25 と仮定すると
g(x)=<f(x+5)+5
g(x)=>f(x-5)-5
の範囲内にある赤点の個数が多い程モデルは適切である。これは最大化を利用した目的関数である。

また、最小化を利用する場合は、以下のように
g(x)=>f(x+5)+5
g(x)=<f(x-5)-5
とすると、範囲内にある赤点の個数が少ない程モデルは適切であると言える。

4.2 設計理由について

自分たちのグループではモデルと各赤点までの距離で評価する目的関数とモデルからある一定の距離ないにある赤点の数で評価する目的関数の二つを考えましたが、後者の方が評価が簡潔で分かりやすいとの理由から、後者を選択しました。
なぜ、このような目的関数が候補に挙がったのかというと、最適なモデルは全ての赤点を通るモデルがいいのですが、今回の場合では、そのようなモデルではわかりづらいので、グラフ全体の傾向を元にしたモデルの方が好ましい。このモデルの評価方法では、最適とされるモデルが全ての赤点がモデルから距離 0 の場所にある、またはモデルから距離 0 内に全ての赤点が存在するという状態をヒントに冒頭で述べた二つの目的関数が候補としてあがりました。

4.3 Level4.2: 嗜好モデルの構築方法

4.3.1 嗜好モデルの説明

4.3.2 上記モデルの利点および欠点

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] 人工知能研究
http://www.ai-gakkai.or.jp/whatsai/AIresearch.html

[3] 最急降下法
http://dsl4.eee.u-ryukyu.ac.jp/DOCS/nlp/node4.html

[4] 最適化問題
http://www.wikiwand.com/ja/%E6%9C%80%E9%81%A9%E5%8C%96%E5%95%8F%E9%A1%8C

[5] シェルスクリプト リファレンス
http://shellscript.sunone.me/