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。

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

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 $\nabla f(\boldsymbol{x})$ 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。

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

勾配ベクトルとヘッセ行列はそれぞれ 1 変数関数 $f(\boldsymbol{x})$ 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、$\boldsymbol{x} = \boldsymbol{a}$ の周りで 2 次の項まで求めると、以下のようになる。

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

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、$\forall \boldsymbol{d} \in \mathbb{R}^n$, $\boldsymbol{d}^T X \boldsymbol{d} \geq 0$ が成り立つことを言い、
- X が正定値とは、$X^T = X$ と、$\forall \boldsymbol{d} \in \mathbb{R}^n \backslash \{\boldsymbol{0}\}$, $\boldsymbol{d}^T X \boldsymbol{d} > 0$ 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) $\boldsymbol{x}^* \in \mathbb{R}^n$ 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.$\boldsymbol{x}^* \in \mathbb{R}^n$ 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 $f : \mathbb{R}^n \to \boldsymbol{R}$ 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　実験方法

次の各項目に取り組む。

(1) 最適化問題とみなせる現実問題の例を挙げる。ただし、目的関数、制約条件、決定変数を大雑把に述べる程度で良いものとする。

(2) 問 1 から問 6 に答えよ。

問 1. 例 2 の問題を定式化する。

問 2. 例 2 の定数に具体的な数値を適当に代入し、そのときの最適解と最適値を求める。ただし、(x_1, x_2)-平面を描いて制約条件を満たす解の領域を図示すること。

問 3. 例 3 の問題の最適解と最適値を求める。

問 4. 例 3 の定式化に以下の 2 つの制約条件を追加したいとする。これらがどのように数式で表現できるかを述べ、その妥当性を説明する。

A. グミとガムの両方は買わない
B. ポテチを買うならガムも買う

問 5. (例 4 の問題の定式化の準備)。決定変数 x と y を考える。x はバイナリ変数、y は連続変数とする。$x = 1$ のときに y は a 以上 b 以下の値をとることができ、$x = 0$ のときには $y = 0$ でないといけないようにしたい。この制約条件を数式で表現し、その妥当性を説明する。

問 6. 例 4 の問題を定式化する。

(3) 最適化問題とみなせる現実問題の例を挙げ、実際に定式化する。(1) の問題と同じでも良い。

5　実験結果・検討・考察

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: saiteki.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
template <class T>
inline bool chmax(T &a, T b)
{
    if (a < b)
    {
        a = b;
        return 1;
    }
    return 0;
}
template <class T>
inline bool chmin(T &a, T b)
{
    if (a > b)
    {
        a = b;
        return 1;
    }
    return 0;
}

using pll = pair<long long, long long>;
pll sub(long long h, long long w)
{
    if (h % 2 == 0 || w % 2 == 0)
        return {h * w / 2, h * w / 2};
    if (h > w)
        swap(h, w);
    return {h * (w + 1) / 2, h * (w - 1) / 2};
}

int main()
{
    long long H, W;
    cin >> H >> W;
    long long res = H * W;

    for (long long h = 1; h < H; ++h)
    {
        vector<long long> a(3);
        a[0] = h * W;
        auto p = sub(H - h, W);
        a[1] = p.first, a[2] = p.second;
        sort(a.begin(), a.end());
        chmin(res, a.back() - a[0]);
    }
```

```
49          for (long long w = 1; w < W; ++w)
50          {
51              vector<long long> a(3);
52              a[0] = H * w;
53              auto p = sub(H, W - w);
54              a[1] = p.first, a[2] = p.second;
55              sort(a.begin(), a.end());
56              chmin(res, a.back() - a[0]);
57          }
58          cout << res << endl;
59      }
```