2009年度応用マクロ経済学　講義ノート
数値計算入門　最適化

Naohito Abe

平成 21 年 10 月 26 日

1　本章であつかう問題

$$f : R^n \Longrightarrow R, x \in R^n$$

この関数 f について

$$\min f(x) \tag{1}$$

の問題を解くことを考える。経済学では、効用最大化、あるいは利潤最大化問題を考えることが多いが、最小化問題の目的関数にマイナス 1 を乗じれば最大化問題となるので、最大化と最小化は同じ問題とみなすことができる。また、制約つき最適化問題についても若干議論する。

1.1　最適化について

ここでは、最適化手法の概観を説明する。最適化は、計量分析における非線形推定や経済モデルにおける主体均衡の導出、厚生経済学における First Best, Second Best の導出など、経済学のほぼ全分野において使用されている。無論、経済学に関わらず、何かを最小化する必要があるときは常に最適化アルゴリズムが利用されている。例えば、自動車やビルの振動を最小化させるためには、最適制御技術が用いられているが、その中には多くの最適化アルゴリズムが利用されている。非線形関数の最小値を求める手法には長い歴史があり、現在でも常に進歩が進んでいる分野である。

最適化問題は、非線形関数のゼロ点を見つけることにも応用可能である。非線形関数の二乗を最小化させれば、その非線形関数のとりうる最小の絶対値となり、ゼロ点が存在すればゼロがその解となる。また、複数の連立非線形関数の解を求める場合も、ゼロ点からの乖離の二乗和を最小化させれば、やはりゼロ点を得ることができる。最適化問題を解く強力なアルゴリズムをもてば、多くの分野に応用することが可能である。

最適化問題を数学的に描写することは極めて簡単である。目的関数が連続で、定義域がコンパクトであれば、その関数には Weierstrass の定理より、最大値および最小値が存在することが知られている。また、目的関数および制約式が微分可能であれば、最大値や最小値の必要十分条件は、クーンタッカーの最適化条件で表現できることは、大学院向けのミクロ経済学における基本知識となっている。しかしながら、こうした理論的知識は、実際に数値解を求める際にはほとんど役に立たない[1]。最適化アルゴリズムの研究には非常に長い歴史があるにも関わらず、未だに我々は汎用的な、一般的に利用可能なアルゴリズムというものを持っていない。成功する可能性が高いとされている Quasi-Newton 法も、なぜそれが他の手法よりもうまく行くのか、理論的な解明は行われてない。実際、関数の形状により、望ましい最適化アルゴリズムは異なることが多く、それ故、様々な最適化ルーチンのオプションを持っていることは、応用する際には極めて重要なのである。グリッドサーチ、シンプレックス、Quasi-Newton の三種類をデータや推計モデルにより使い分けることで、「プログラムを回したけど収束しませんでした」という報告を避けることができるかもしれない[2]。

実際に数値計算を行う際に、すぐに直面する問題は、「微分」である。微分の計算は、理論上、解析的に行うのは極めて簡単である。しかし、control の微小な変化と目的関数の変化の比率という数値は、分子における桁オチの問題に加え、微小な値を分母に持ってくることになり、数値計算における誤差が極めて大きくなってしまう。また、最適化理論において重要な目的関数の二次導関数の符号、あるいはヘシアンの符号は二階微分となり、ただでさえ誤差の大きな微分操作を二回行うことになる。MDE(Minimum Distance Estimator) を用いる者や比較的大型の経済モデルを分析する者にとり、最適化アルゴリズムに関して知るということは、いかにしてこの数値微分と対処するかを知ることと等しいと言う事もできるだろう。

一つのアプローチは、数値微分を全く使用しないことである。例えば、一変数の最適化問題であれば、その変数がとりうる値をひたすら目的関数に代入し、最小の値を与えるものを探すことが出来る。Grid Search と呼ばれるこの手法はかなり強力であり、一変数、あるいは二変数で、かつ、とりうる値 (定義域) がわかっている場合はかなり応用可能性が高い。無論、計算は極

[1] 一般均衡解も、存在を示すことは不動点定理で証明できるが、実際に均衡解を導出する際には全く異なる手法が必要である。この授業では一般的な数値解析の問題を取り上げるが、分析対象が静学モデルの均衡であれば、Scarf による素晴らしいアルゴリズムが存在する。Scarf アルゴリズムに関しては Scarf 本人のものも含め多くの論文が書かれているが、興味のある者は Bryan Ellickson による *Competitive Equlibrium: Theory and Application* (1993) Cambridge University Press を読むことをお勧めする。一般均衡分析の基本や Aumann による貢献等の内容に加え、Scarf アルゴリズムに関してもかなりのページを割いて紹介している。

[2] 実のところ、問題が収束しない時、一番役にたつ情報は、最適化アルゴリズムよりも、モデルとデータの構造に関する知識である。最適化させる目的関数に、経済理論が示す特徴をもたせたり、簡単化させることで、複雑な非線形問題を往々にして簡易化させることが可能である。また、ほとんど変動がない変数、あるいは、強い相関をもつ二つの変数を同時にいれないよう、データの特徴を踏まえたうえで、どのような情報を用いてパラメターを識別しているのか、それを推計前に考えることは極めて重要である。

めて非効率であり、目的関数の傾き情報等を一切利用しない primitive な手法であるため時間がかかる。多変数の場合は、Simplex 法と呼ばれる、もう少し効率的な手法が存在する。Simplex 法は Matlab の最適化アルゴリズムの default であり、また目的関数に微分可能性を要求しないため、応用範囲は広い。実際、経済モデルでは流動性制約等、微分可能でない状況を考えることが近年増えており、Simplex 法を利用することは思いのほか多い。もっとも、多くの変数を取り込む最適化問題では収束するまで信じられないくらい長い時間がかかることがある。並列処理による数値解析が普及すれば、Feed Back を利用しない Grid Search の応用範囲は一気に広がる可能性はあるが、並列処理が経済学の応用で主流になるにはもう少し時間がかかりそうである。

微分情報を利用すれば、どちらに移動すれば目的関数が増えるか、減少するか知ることになるため、最適化させる際に有利である。数値微分を利用するアプローチでは、二階微分をどう処理するかでアルゴリズムが分かれる。二階微分を数値的に計算する Newton 法と、なんらかの近似を利用する Quasi-Newton 法に大別される。ただ、二階微分の計算を回避する場合でも、計量経済学で使用する場合は標準誤差を計算するためにヘシアンを計算する必要はいずれ出てくる (近年では Bootstrap 法など、数値ヘシアンに頼らずに標準誤差を求める手法も開発されており、非線形推計で標本数が多くない場合にはよく利用されている)。

1.2 微分を用いない最適化: Simplex 法

多変数関数の最適化において、微分情報を用いない手法には、もっとも基本的な Grid Search 法のほかに、Simplex 法があり、Grid Search 同様に非常に popular である。Matlab の fminsearch コマンドのアルゴリズムでもある。

Nelder-Mead algorithm とも呼ばれるこの手法は、n 次元上に n+1 次元の多面体 (Simplex) を最小点まで転がしていくやり方である。

まず、初期点として、n 次元空間上で n+1 次元のベクトル $\left\{x^1, x^2, ..., x^{n+1}\right\}$ を考える。この多面体は full dimension であり、n-1 次元の hyperplane にこのベクトルは存在しないとする[3]。もっとも単純な simplex は、原点および、各軸上に {1,0,0,...,0} のような点をとってくことで作ることができる。

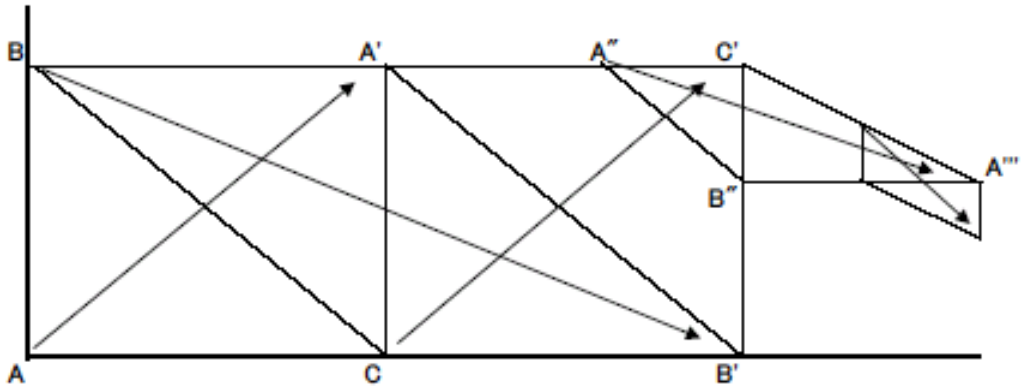
つぎに、最小化させる関数を各点で評価し、小さい順に並べて番号をつける。すなわち、

$$f\left(x^i\right) \geq f\left(x^{i+1}\right), i = 1, ..., n$$

我々の目的は、関数 f の最小点を求めることであり、一つの自然な発想は、最も大きな値をとる x^1 を何か他の点で置き換えて新しい Simplex を作って

[3]ようは、多面体の異なる面が、同一平面にない、ということであり、多面体が平面上を転がすときに二つの面が同時に接してしまうと、多面体が無駄に大きくなってしまうので、この仮定をおいている。実際には、あまり意識する必要はない。無駄な多面体だと収束するのに時間がかかってしまい非効率になってしまう。

図1 Judd (1998)による説明



いくことで、Simplex のとる f の値を小さくしていくことができるだろう。

x^1 は Simplex の中で最も大きな値なので、最も小さな値は x^1 の反対方向にあると考えるのは自然である。そこで、 x^1 に関して、Simplex の反射点である y^1 をもってきて、もしも $f\left(y^1\right) < f\left(x^1\right)$ になっていれば、この作業は成功したことになる。そして、また Simplex 上の各点における f の値で並びなおし、同じ作業を繰り返していく。$f\left(y^1\right) > f\left(x^1\right)$ であるときは、x^1 ではなく $x2$ に関する反射点を考える。同様にして、$f\left(y^i\right) < f\left(x^i\right)$ になるような x^i が存在すれば、また並べなおしていく。もしもそのような x^i が存在しない場合、我々の Simplx は大きすぎると考えることができる。その場合は Simplex 全体を、$f\left(x^{n+1}\right)$ の方向に縮小させて、また同様のプロセスを繰り替えす。

二次元の最適化を考え、f が示す二次元曲面を考える。Simplex は二次元曲面上の三角形となる。すると、上記のプロセスにしたがうと、三角形を下に向かって転がしていき (実際には三角形は平面なので自然には転がっては行かないが ⋯)、もしも大きすぎて転がらなくなったら三角形を小さくして、また転がしていく。そのうち三角形は小さくなり、いずれは最小点に収束していくのである。

例えば、上記の図 1 を用いて Judd (1998) に従い説明すると、最初の Simplex が三角形 ABC で、A 点が ABC の中で最も大きな値になっていれば、BC 線をもとに三角形をひっくり返し、A'BC の三角形とする。つぎに B 点が最も

大きければ、A'C に沿ってひっくり返し、A'B'C つぎ C 点に関してひっくり返して A'B'C' となる。全体を縮小させ、A"に関してひっくり返し、さらに縮小させ、という作業が上記の図では説明されている。

この手法は、微分情報を用いず、ただ Simplex 上での関数の値を比較して転がしていくだけなので、関数が連続である必要すらない。ただし、最適化させる変数が多いと非常に長い時間がかかる。

1.3 微分情報を用いる最適化 1: Newton 法

最適化させる関数が二階微分可能であるとすると、非常に高速に最小点を求める Algorithm が存在する。目的関数を x^k で Taylor 展開すると

$$f(x) = f\left(x^k\right) + \nabla f\left(x^k\right)\left(x - x^k\right) + \frac{1}{2}\left(x - x^k\right)' H\left(x^k\right)\left(x - x^k\right)$$

ただし、$\nabla f\left(x^k\right)$ は Gradient, H は Hessian である。もしも f が Convex であれば H は正値定符号となり、この二次関数の最小点は

$$x = x^k - H\left(x^k\right)^{-1} \nabla f\left(x^k\right)'$$

となる。Newton 法とは x^k を上記の式で update していく方法であり

$$x^{k+1} = x^k - H\left(x^k\right)^{-1} \nabla f\left(x^k\right)'$$

とし、繰り返していく。

Newton 法は、関数のヘシアンが既知であり、かつ Convex であるときは非常に高速に収束することが知られている。実際、もしも関数が二次関数であれば、Newton 法は一回の操作で最適点に達することが出来る。しかしながら、一般に Newton 法が用いられることは少ない。なぜなら、コンピューターが苦手とする数値微分の回数が多いからである。Newton 法では、Gradient のみでなく、Hessian およびその逆行列の計算も必要となっている。数値 Hessian は、Gradient に比較しても計算数が極めて多く、かつ不正確である可能性が高い。また、もしも関数が Global に Convex でない場合、初期点が Concave の箇所にある場合はあっという間に発散してしまうことがある。実際に用いられる手法は、数値 Hessian の計算回数を極力減らす Quasi-Newton 法である。

1.4 Line Search

一般に、x の update rule を下記のように書くことにする。

$$x^{k+1} = x^k + \lambda^k \Delta^k$$

ただし、Δ^k は x を変化させる方向を示す傾きベクトルであり、λ^k はスカラーである。傾きが一定の場合、λ^k の値を増減させることで、最小点に達す

る速度を早くさせることが出来る。最適化 Algorithm の内部の操作の一つとして、全体の高速化のために、一次元の最適化問題を取り込むのである。一次元の最適化であるため、最適な λ^k を求めるのは簡単であるし、また範囲も非負領域で探せばよい (別に負の値で探しても構わない)。この作業を Line Search と呼び、多用されている。

Newton 法は、

$$\Delta^t = -H\left(x^k\right)^{-1}\nabla f\left(x^k\right)'$$

$$\lambda = 1$$

の特殊な line search を行っているケースと考えることも可能である。

もしも f の Convexity が十分に確保できない場合、Newton 法は収束しないケースが多いが、Line Search を組み合わせることで収束する確率を高めることができる。

1.5 Quasi-Newton 法

Quasi-Newton 法は Newton 法に Line Search を組み合わせ、Hessian を他の $n \times n$ の正値半定符号行列で置き換える手法である。利点は、Hessian を計算する必要がなく計算速度が向上することと、常に正値半定符号行列を用いることで収束する確率を高めていることである。よく用いられるのは BFGS(Broyden-Fletcher-Goldfab-Shanno) 法である。

アルゴリズム 1 *BFGS*

1. $\Delta^k = -H\left(x^k\right)^{-1}\nabla f\left(x^k\right)'$ を計算する。
2. $f\left(x^k + \lambda^k\Delta^k\right)$ を最小化させるような λ^k を計算する *(Line Search)*。
3. $x^{k+1} = x^k + \lambda^k\Delta^k$
4. $z^{k+1} = x^{k+1} - x^k$
5. $y^k = \nabla f\left(x^{k+1}\right)' - \nabla f\left(x^k\right)'$
6. $H^{k+1} = H^k - \frac{H^k z^k z^{k\prime} H^k}{z^{k\prime} H^k z^k} + \frac{y^k y^{k\prime}}{y^{k\prime} z^k}$
7. *Stop if* $\left\|x^{k+1} - x^k\right\| < \varepsilon\left(1 + \left\|x^k\right\|\right)$, *else go to step 1.*

なお、Hessian の update formula を

$$H^{k+1} = H^k + \frac{\left(y^k - H^k z^k\right)y^{k\prime} + y^k\left(y^k - H^k z^k\right)'}{y^{k\prime} z^k} - \frac{z^{k\prime}\left(y^k - H^k z^k\right)y^k y^{k\prime}}{\left(y^{k\prime} z^k\right)^2}$$

とすると、DFP(Davidson-Fletcher-Powell) 法と呼ばれるものになる。

BFGS の法が経験的には速く収束すると言われているが、理論的根拠は知られておらず、実際、DFP の法が速いときもある。また、収束に成功するか失敗するかも、両者で異なることがある。したがって、実際にはまず BFGS で行い、うまく行かない場合は DFP、さらには Simplex 法を試すことになる。なお、H の初期値としては、単位行列を用いるのが普通である。単位行列は正値定符号であり、この条件をみたす。なお、Greene(2004) や他の教科書では最大化アルゴリズムのための BFGS とし、Hessian の逆行列 (の符号をひっくり返したもの) を update する formula を説明している。基本的には同じだが、最小化と最大化の違いのため、update する Hessian は正値ではなく負値になるようになっているので注意すること。

もう一つ注意するべきことは、BFGS も DFP も、update される Hessian の代替物は真の Hessian に収束するとは限らないことである。したがって、Information matrix を用いて標準誤差を求める場合は、収束した後に数値 Hessian を改めて計算する必要がある。Matlab では Optimization Toolbox の fminunc が対応している。

1.6 Matlab Code

BFGS を実施するには (1) 数値 Gradient の計算 (2)Line Search(3) 逆行列の計算が必要である。逆行列に関しては Matlab のコマンドを利用することにし、まず数値 Gradient の計算を行う Code を紹介する。

Gradient の計算を行うには、finite difference、すなわち微分の定義に従い差分を計算する方法と、目的関数を多項式で近似し、その解析的な微分を計算する方法がある。また、他にも多くの手法があるが、ここでは finite difference 法の code を紹介する。

下記のファイルを numgradient.m として保存する。なお、他のセクションもそうだが、ここで紹介する Matlab Code は教育目的のために書かれており、実際に分析に使用するのにはお勧めできない。実用的な Code にするためには、様々な例外処理をする必要があるからである。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function G = numgradient(f,x)
% Computes numerical gradient
% ——————————————————-
% Usage: G = numgradient(func,x)
% Where: func = function name, fval = func(x)
% x = vector of parameters (1 x n)
%
% ——————————————————-
```

```
% RETURNS:
% G = finite differnce gradient (n x 1)
% ------------------------------------------------------------
%Naohito Abe
%October 27, 2006
%nabe@ier.hit-u.ac.jp
eps = 1e-10;
[n1 n2] = size(x);
fx = feval(f,x);
% Compute the stepsize (h)
h = 0.000001*(max(abs(x),eps));
% Compute forward and backward step
ho=zeros(n2,n2);
h1=zeros(n2,n2);
h2=zeros(n2,n2);
GO=zeros(n2,1);
G1=zeros(n2,1);
for i=1:n2 % Creating the evaluation points
    h1(i,i)= h(i);
    ho(i,:)= x+h1(i,:);
    h2(i,:)= x-h1(i,:);
end
for i=1:n2 % Taking the differences
    GO(i)= feval(f,ho(i,:)) - fx;
    G1(i)=-feval(f,h2(i,:)) + fx ;
end
G=zeros(n2,1);
for i=1:n2
    G(i) =(GO(i)+G1(i))/(2*h(i));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

上記で注意する点は、(1) 微分を計算する際に forward step と backward step の平均を使用していること (2)input を横ベクトルで与えること、である。(1) を forward step のみにした場合は、convex のときには過大に、concave のときには過小になる。backward step との平均値を用いる方がより適切なはずである。実際に、強く曲がっている、傾きが既知の関数 (例えば指数関数とか) を用いて、forward step と backward step の誤差、およびその平均値の誤差を計算してみるのは良い練習になる。

BFGS は下記のように実行できる。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [z, H,xopt]=bfgsalg2(f,x)
%
% Computes BFGS
% ---------------------------------------
% Usage: [z, H,xopt] = bfgsalg2(func,x)
% Where: func = function name, fval = func(x)
% x : vector of parameters (n x 1)
% ---------------------------------------
% RETURNS:
% xopt: x that achives the minimum f
% z: the minimized value H: Hessian
% ---------------------------------------
% Based on Judd (1998)
% Naohito Abe
% October , 2009
% nabe@ier.hit-u.ac.jp
[n1,n2]=size(x); % the number of lines
maxit = 100; % the maximum number of iterations
delta = 1e-15; % the torelence level
H=eye(n2); % The initial Inverse Hessian
d=ones(1,n2); % The initial delta
i=1;
% The main loop
while max(abs(d))>=delta
    % Calling numerical gradient
    grad1=numgradient(f,x);
    % the direction for update
    B=-inv(H);
    s=B*grad1;
    % Obtaining the optimal ramda by grid search (line search)
    ramda =0:0.01:3; % limiting the range for ramda
    [num1, num2]=size(ramda');
    mat1=zeros(1,num1); % initializing (can be moved outside the loop)
    for i=1:num1
        mat1(i)= feval(f,x+s'*ramda(i));
    end
    mat2=[ramda;mat1];
    [fval1, index]=min(mat1);
```

```
    optramda=ramda(index);
    % updating x
    x2=x+optramda*s';
    d=x2-x;
    % for BFGS update formula
    y=numgradient(f,x2)-grad1;
    %w=(optramda*s')'-B*y;
    if y'*d' <0.0001*delta % Avoiding zero division
        H1=H;
    end
    if y'*d'>=0.0001*delta % BFGS Update
        H1=H+ (y*y')/(y'*d') -(H*d'*d*H)/(d*H*d');
    end
    H=H1; % update
    x=x2; % update
    i=i+1; % for counting the iteration
    % terminating the loop
    if i == maxit
        d=delta/2;
        warning('maximum number of iteration achived')
    end
    % evaluating the function
    z=feval(f,x);
    xopt=x;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

この code のなかで、上記の数値 gradient の関数を呼び出していることに注意せよ。また。line search は 0 から 3 の値の中で行っているが、別に負の値からはじめても、もっと大きな上限を与えても構わない。ただ、あまり大きな range を与えると使用メモリーサイズがその分増えるので注意すること。

BFGSupdate の方法を DFP の公式に変更すれば、これは DFP の Code となる。Matlab の練習に良いので、自分で DFP を書いてみることをお勧めする。

1.7 制約つき最適化

経済学の問題では、最適化を行う際に、操作変数がある集合内に入っていることを条件として課す事がある。たとえば、予算制約をみたす、資源制約

をみたす、誘引両立条件をみたす、あるいは非負制約を満たす、等である。ある種の状況下では制約を除いて制約無しの最適化問題に変換することが可能であるが、多くの場合はそうい簡便法は不可能である。

このようなときには、クーンタッカーの一階条件を用いて、非線形連立方程式の解を解くか、penalty 関数を用いるかの二種類の方法がある。クーンタッカーは単なる非線形連立方程式の問題となるが、綺麗な関数形でない限り、あまり実用的ではない。

Penalty 関数のアイディアは極めて簡単である。

$$\min f(x)$$

$$st.g(x) = a, h(x) \leq b0$$

を考えよう。このとき、Penalty 関数を用いると、上記を制約無しの最適化問題と考え、下記のような問題で近似する。

$$\min f(x) + P\left(\sum_i (g_i(x) - a_i)^2 + \sum_j (\max[h_j(x) - b_j, 0])^2\right), P > 0.$$

実際には、P も、最適化の iteration の中で update していくことが多い。最初は小さく、そして徐々に大きくしていくのである。Judd は $P^{k+1} = 10P^k$ としている。また、ヘシアンが奇妙な挙動を示すときには、目的関数を二次関数で、制約を一次式で近似することも考えられる。

例えば、

$$Min \quad x^2$$

$$s.t.3x - 2e^x = 0$$

を考える。この解は 0.4053 であるが、この程度の問題であれば update しなくとも一瞬で収束し、P=500 程度でほぼ正しい値になっている。

x	P=10	P=100	P=500	P=1000	P=10000
0.37	32.0159	318.9269	1594.087	3188.037	31879.14
0.38	31.99127	318.6131	1592.488	3184.832	31847.02
0.39	31.97729	318.404	1591.412	3182.671	31825.34
0.4	31.9741	318.301	1590.863	3181.565	31814.21
0.41	31.98166	318.3037	1590.85	3181.52	31813.7
0.42	32.00022	318.4146	1591.367	3182.558	31823.99
0.43	32.02984	318.6343	1592.432	3184.679	31845.12
0.44	32.07065	318.9641	1594.046	3187.898	31877.24
0.45	32.12277	319.4052	1596.216	3192.229	31920.47