2009年度応用マクロ経済学講義ノート
DP(2)

阿部修人

平成 21 年 12 月 10 日

概 要

1 数値計算: Discretization

これまで紹介した Value Function Iteration、Guess and Verify、および Policy Function Iteration は、いずれも正しい解を手計算で得ることができた。しかしながら、これが可能だったのは各ステップで Value Function や Policy Function の候補を State Variable の関数として、Closed Form で表現することができたためである。残念ながら一般に Closed Form の解は存在せず、実際に計算する際にはなんらかの形で Closed Form の形で関数を表現せねば計算できない。

Closed Form が存在しない関数を既知の関数で近似する場合、大きく分けて 2 つの手法が考えられる。一つは多項式、あるいはスプライン関数等で近似する手法であり、もう一つは関数ではなく、点の集合として扱う手法である。前者の中には良く知られている線形近似も含まれる。前者の場合、我々の既知のある種の多項式の集合の中で、問題となっている価値関数に最も近いものを探すことになる。換言すれば、多項式のつくる空間に直行写像を作るということであり、それゆえ Projection Methods と言われる。

後者は一般に離散近似法と呼ばれるものであり、多項式という制約を置かずに解くことが可能であるため、非常に強力な手法であると言える。特に、数値計算に用いられる多項式の多くが連続微分可能な Class に限定され、それゆえ Policy Function も連続微分可能なものになるのに対し、離散近似の場合連続微分可能とは限らない Policy Function も扱うことが出来る。これは、非可逆的投資や借り入れ制約など、今日の経済にとり興味深い問題も分析可能になるという大きな利点がある。一方、離散近似は数値計算に必要な Computation の負荷が極めて大きく、実際に計算可能なモデルの範囲は必ずしも広くはない。とくに国際貿易や多部門モデルへの応用は現時点ではまず不可能である。

本節では、離散近似法を紹介する。多項式、あるいはスプラインによる近似は後の講義ノートで説明する。

「離散近似」という言葉に含まれる“離散”は、State Variable を連続な集合ではなく、有限個の点の集合とみなすという意味である。これまでの最適成長の例では資本が State Variable であり、(0,∞) の実数値をとると仮定していたが、離散近似の場合、例えば {0.1, 0.2, 0.3, ...,0.9, 1} の値しかとらないと仮定するのである。無論、この区間を広く取れば取るほど、また点を細かく取れば取るほど理論的に正しい解に近くなっていく。一方、点を多く取れば取るほど計算にかかる時間とメモリーの量が膨大になっていく。これは Curse of Dimension と呼ばれる現象であり、後で説明する。

前節までと同様に、単純な最適経済成長モデルを考えよう。ただし、今回は対数効用ではなく、CRRA を仮定し、Closed Form での解が存在しないモデルを考える。単純化のため、減価償却のない世界を考えるが、これは全く本質的なものではない。

Social Plannar の問題として、下記のようなものを考える。

$$\max_{\{c_t,k_{t+1}\}_o^\infty} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{\gamma+1}}{\gamma+1} \tag{1}$$

subject to

$$k_{t+1} = Ak_t^\alpha + k_t - c_t \tag{2}$$

定常状態での資本ストックが 1 になるようにパラメター A を調整する。定常状態においては

$$Aak^{\alpha-1} + 1 = \frac{1}{\beta} \tag{3}$$

であるから

$$A = \frac{1-\beta}{\alpha\beta} \tag{4}$$

とする。これは、単に、GDP の単位を調整したにすぎない。

ここでは、Judd [1998] の例に従い、パラメターを $\gamma = -2, \alpha = 0.25, \beta = 0.96$ とする。

1. まず、資本ストックをどのような集合で近似するかを決定する。ここでは、定常状態の水準を 1 としたので、その区間を含む集合を考える。例えば、0.2 と 1.8 の間を 0.001 の間隔で埋めると[1]、資本ストックは 1600 個の点の集合となる。

2. 次に、来期の資本ストックと今期の資本ストックの組み合わせを考える。予算制約より

$$c_t = Ak_t^\alpha + k_t - k_{t+1} \tag{5}$$

[1]この例は非常に多くのメモリーを要求するが、それでも 3GHz の Core 2 Duo に 8GB のメモリーを積んだ Note PC では三十秒程度で計算を終えることが出来る。

であり、今期の資本ストックと来期の資本ストックの全ての組み合わせを考えることで、消費量を決定することが出来る。各期の効用関数を資本ストックで定義し

$$u = u\left(k_t, k_{t+1}\right) \tag{6}$$

と考えることと等しい。予算制約を使って、今期の資本ストックと来期の資本ストックの可能な組み合わせ、ここでは 1600×1600 の行列の各要素に消費水準を割り当てる。なお、グリッドの数を増やすと、より広い範囲で、またはより詳細な形状を知ることが可能であるが、計算の時間の増大もまた著しい。1600 というのは、現在のコンピューターにとっては決して大きいサイズではないが、グリッドの数を倍、3200 にすると、必要とするメモリー量や計算時間はその二乗の Order で増える。ここで、新たに、もうひとつの状態変数、例えば人的資本を加えてみよう。すると、一つの人的資本のグリッドに対し、3200 の資本ストックのグリッドが対応する。もしも人的資本ストックのグリッドが 3200 であれば、$3200 \times 3200 \times 3200 \times 3200$ の情報が必要となる。これが Curse of Dimension と呼ばれる現象であり、離散近似手法の最大のもんだいてんである。コンピュテーションの限界が、スピードではなく、メモリーによるものとなり、たとえ大型の汎用機や Super Computer を用いたとしても解決できない深刻な問題となっている。

3. Value Function Iteration を行う。Value Function の初期値として、例えば zero 関数を用いる。無論、どんな関数でも構わないが、もっともプログラムしやすいものを選ぶのが間違いが少ない。

$$V_1 = \max_{k_{t+1}} \left(u\left(k_t, k_{t+1}\right) + \beta V_0\left(k_{t+1}\right)\right) \tag{7}$$

を考える際に、$V_0 = 0$ を初期値とする。ここで、$u\left(k_t, k_{t+1}\right)$ は 1600×1600 の行列であり、max をとるということは、今期の資本ストックの水準、ここでは 1600 個の水準ごとに、右辺の値を最大にするような来期の資本ストックを 1600 の中から一つ選ぶことに等しい。最初のステップでは Value Function の初期値が zero であるから単に効用関数を最大化する組み合わせが選ばれ、次のステップでは、得られた Value Function を右辺に移動して、また今期の各資本ストックごとに、右辺全体を最大化させる来期の資本ストックが選ばれる。

4. Value Function が収束するまで Iteration を繰り返す。ここでは、Value Function の存在や一意性は問題にならない。なぜなら資本ストックの水準に上限および下限を外生的に与えているため、各期の効用関数は有界となっており、さらに凹関数と仮定している、前回の講義ノートで触れたように、定理より Bellman Equation は縮小写像となり、Value Function は Unique に決まり、かならず収束する。

5. 得られた Value Function から Policy Function を計算する。ここでいう Policy Function は、Iteration の最後で選ばれた今期の資本ストックと来期の資本ストックの組み合わせに他ならない。

では、具体的に Matlab のプログラムを見てみよう。

```
% パラメターの設定
%
alpha = 0.25; % production parameter   Judd より。
beta = 0.9; % subjective discount factor
delta = 1; % 1 -depreciation rate
gam = -2; % preference parameter
%
%
% 資本ストック水準の離散化
%
mink = 0.2; % minimum value of the capital grid
maxk = 1.8; % maximum value of the capital grid
ink = 0.001; % size of capital grid increments
nk = round((maxk-mink)/ink+1); % number of grid points
%
% 各期の効用関数の作成
%
util = -1000000*ones(nk,nk);
%
% 初期値をこのようにするのは、最適な来期の資本ストック
% 消費水準を選ぶときの選択肢の中で、消費水準がゼロや負の
% 値が生じることがあるが、そのような数値が決して選択されないように
% するためである。このプログラムでは不要な一行であるが、後に不確実性
% を含むコードを書くときに、必要になる。
%
% 今期の資本ストックに関するループ
%
for i=1:nk;
    k=(i-1)*ink + mink;
    %
    % 来期の資本ストックに関するループ
    %
    for j=1:nk;
        kprime = (j-1)*ink + mink;
```

```
            invest = kprime - delta*k;
            cons = ((1-beta)/(alpha*beta))*(k^(alpha)) - invest;
            if cons > 0;
                util(j,i)= (cons^(gam+1))/(gam+1);
                % 今期の資本が i で、来期資本が j の効用水準
            end;
        end;
    end;
    %
    % 各関数の初期化
    %
    v = zeros(nk,1); % Value Function の初期値
    % Value Function の定義域は今期の資本ストックの各グリッドである
    % ことに注意。
    %
    decis = zeros(nk,1);
    iter = 0;
    metric1 = 10;
    metric2 = 10;
    [rs,cs] = size(util);
    %
    % Bellman Equation を iterate する。
    %
    while metric1 ~= 0 | metric2 > 0.001 ;
       %    metric2 を 0.01 とすると 60 回で、0.001 とすると
        %    76 回の iteration で、metric2 ~= 0 とすると 381
        %    回で収束する。なお、0.01 以下にしてもあまり結果に
        %    違いは出ない。
        %
        % すべての state に関してループを走らせる。
        %
        for i=1:cs;
            r(:,i) = util(:,i) + beta*v;
        end;
        [tv,tdecis] = max(r); % tv is 1*cs vector with max of each cols of r
        % tdecis は 1*cs ベクトルであり、この要素は r の値を最大にするもの
である。
        % この関数は各状態において Value Function を所与として
        % Bellman Equation を最大にする来期の資本ストック水準を与える。
```

```
    % tv はそのときの value の値である。
    %
    tdecis = tdecis';
    tv = tv';
    metric1 = max(any(tdecis-decis)); % test nonzeros (1 if yes, 0 OW)
  % Policy Function の違いのチェック
    metric2 = max(abs(v-tv));
  % Value Function の違いのチェック
    v = tv;
    decis = tdecis;
    vfor8 = v(1); % value of v for k=kmin=0.2
    ufor8 = (decis(1)-1)*ink + mink; % value of control for k=0.2
    iter = iter+1;
    disp([ iter metric1 metric2 vfor8 ufor8]);
   % iteration の様子の表示
end;
%
%Policy Function と Value Function を計算する。
%
policy = (decis-1)*ink + mink; % policy function
%
p = -10000*ones(nk,1); % utility under the optimal policy
for i=1:cs;
    k = (i-1)*ink + mink;
    invest = policy(i) - delta*k;
    cons = ((1-beta)/(alpha*beta))*(k^(alpha)) - invest;
    if cons > 0;
        p(i)= (cons^(gam+1))/(gam+1);
    end;
end;
betam = beta*ones(cs,1);
value = p/(ones(cs,1)-betam);
%
% 結果を出力する。
%
disp('PARAMETER VALUES');
disp('');
disp(' alpha beta gamma ');
disp([ alpha beta gam ]);
```

```
disp('');
%
%
% 結果をグラフに出力する。
%
kgrid = [(mink:ink:maxk)']; % capital grid
%
figure(1)
plot(kgrid',value);
title('DETERMINISTIC GROWTH MODEL: VALUE FUNCTION');
%
figure(2)
plot(kgrid',policy,'.',kgrid',kgrid','-');
title('DETERMINISTIC GROWTH MODEL: POLICY FUNCTION');
```

図 1 はこの結果得られる Value Function を示している。Value Function は単調増加の凹関数になっており、Lucas, Stokey の結果と整合的である。また、図 2 で示される Policy Function は極めて線形に近く、わずかに 45 度線よりも小さい傾きになっていることがわかる。また、定常状態が 1 であることも、Policy Function が 1 で 45 度線と交わっていることから確認できる。

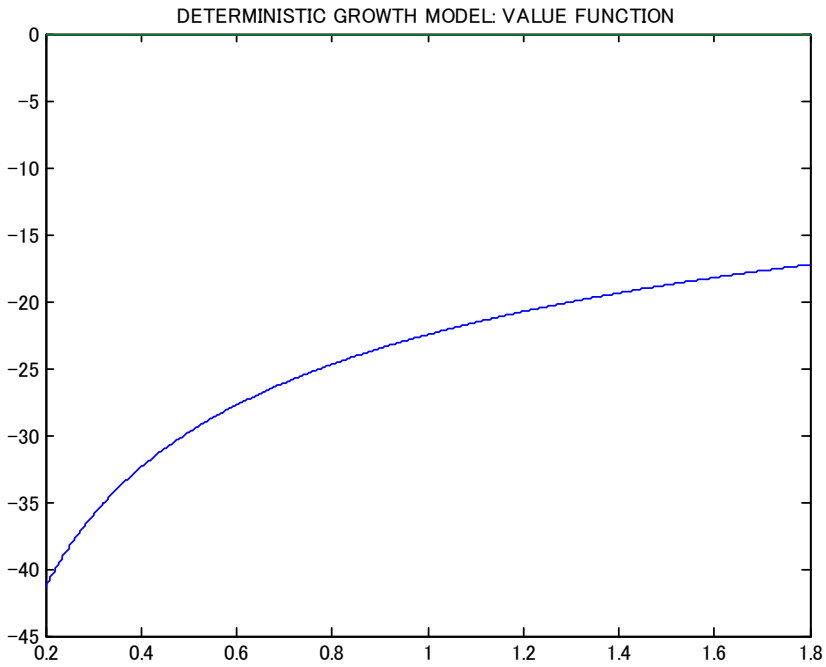
一つ興味深いのは、この離散近似の結果と線形近似の結果との比較であろう。線形近似の場合、Value Function も Policy Function も線形と仮定されている。離散近似の場合の Value Function が凹になっているため、定常状態から乖離した点、とくに左側では線形近似と離散近似の乖離は大きくなるであろう。また、線形近似の場合はテイラーの定理より、定常状態付近では両者はかなり近くなることが予想される。図 3 は両者の乖離を示している。予想通り、資本ストックがゼロに近いところでは両者の乖離は大きくなっている。縦軸のスケールは 0.001 であり、Value Function の値は-40 近くであるから、両者の乖離はそれほど大きくはない。また、興味深いのは、両者の乖離が波を打っていることである。これは離散近似の場合常に生じる現象であり、これを回避するには grid をさらに細かくとっていくしかないように思われる。

下記のコードを前記の離散近似 DP の後半に置くことで、線形近似と離散近似の解を比較することが出来る。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cons = (1-beta)/(alpha*beta);
lam = cons^gam;
%
```

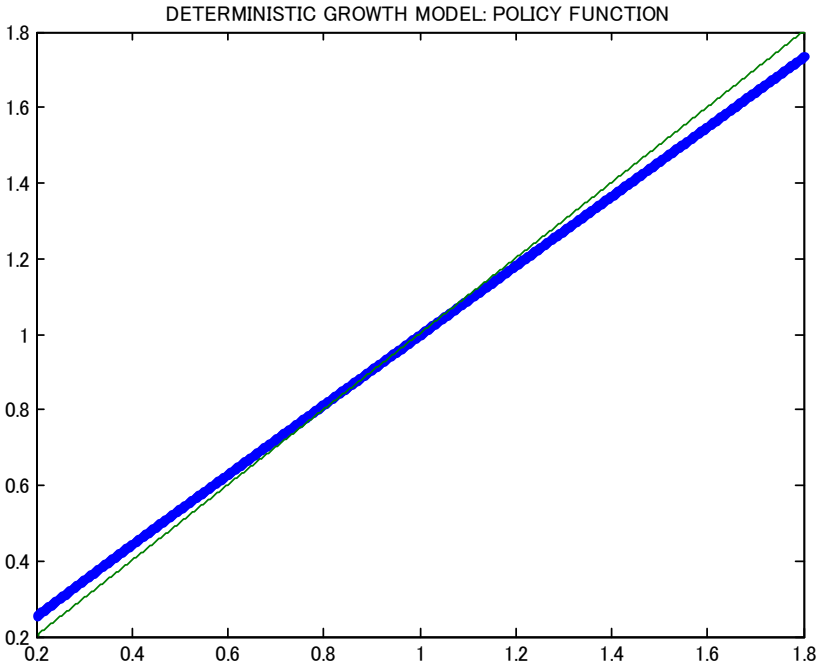
```
n=1; % The number of the predtermined variables
%
%
%
%
% Matrices For subroutine to solve dynamic optimization problem
%
%
% MCC matrix
%
mcc=zeros(1,1);
mcc(1,1) = gam * consˆ(gam-1);
%
%
% MSC Matrix
%
mcs = zeros(1,2);
mcs(1,2) = 1;
%
%
% MCE Matrix - no stochastic elements
%    不確実性はないからゼロ行列
%
mce = zeros(1,1);
%
%
% MSS0 Matrix
%
mss0 = zeros(2,2);
mss0(1,1) = lam*(1-beta)*(alpha-1);
mss0(1,2) = 1;
mss0(2,1) = 1;
%
%
% MSS1 Matrix
%
mss1 = zeros(2,2);
mss1(1,2) = -1;
mss1(2,1) = -1/beta;
```

```
%
%
% MSC0 Matris
%
msc0 = zeros(2,1);
%
%
% MSC1 Matrix
%
msc1 = zeros(2,1);
msc1(2,1) = -1;
%
%
% MSE0 Matrix
%
mse0 = zeros(2,1);
%
% MSE1 Matrix
%
mse1 = zeros(2,1);
%
%
% PAI Matrix
%
pai = zeros(1,1);
%
%
%
[GXX,GXZ,GUX,GUZ,M,Psi,V] = burns6(n,mcc,mcs,mce,mss0,mss1,msc0,msc1,mse0,mse1,pai);
%
%
linpol = zeros(cs, 1);
for i = 1:cs
      linpol(i) = 1-GXX +GXX*(mink+ink*(i-1));
end
diff = policy-linpol;
%
figure(3)
plot(kgrid',diff);
```

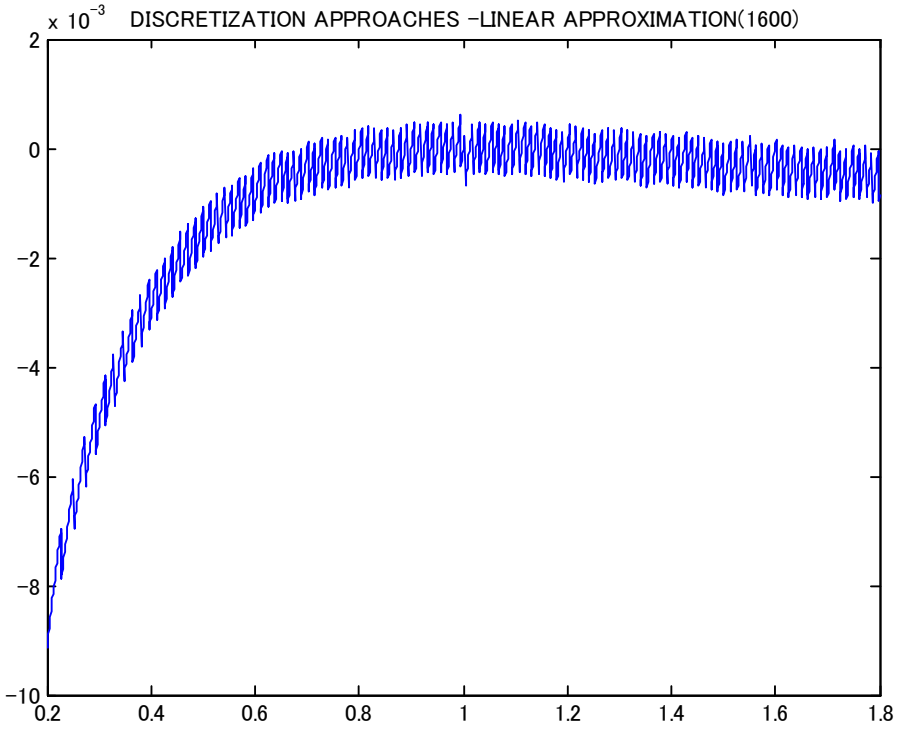


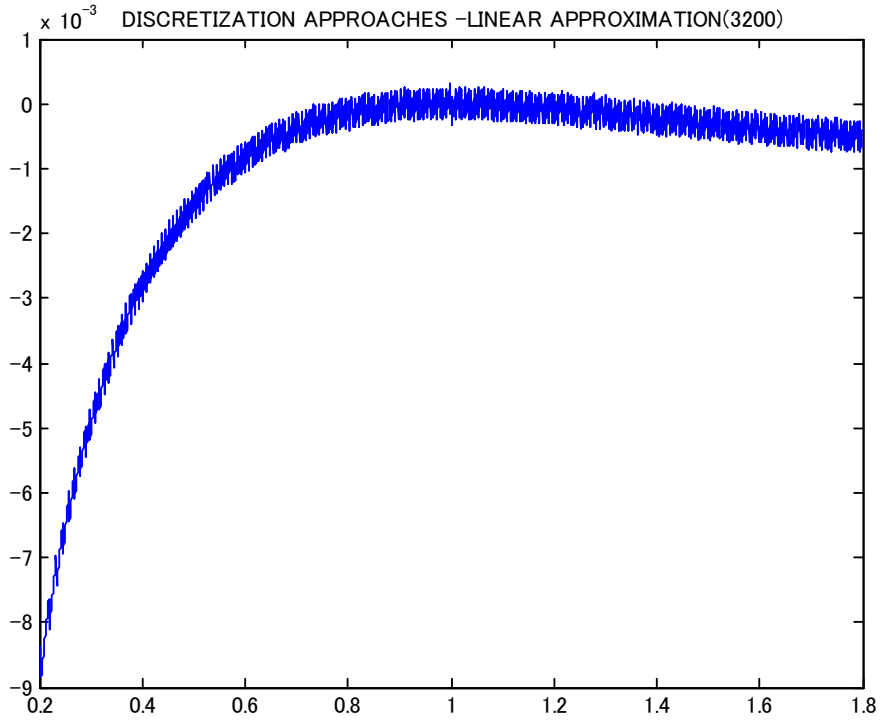
title(' DISCRETIZATION APPROACHES -LINEAR APPROXIMATION(400)');

% % % % % % % % % % % % % % %5



なお、グリッドのとり方を細かくし、3200 点にした場合の、線形モデルと離散近似の乖離を最後に示している。グリッドが細かいため、1600 ケースとは異なるパターンとなるが、奇妙なジグザグ運動は残ってしまう。これは metric を細かくしてもその幅はほとんど変わらない。1600 ケースに比べると、その幅は若干低下しており、グリッドを増加させることによる計算精度向上をみることができる。


x 10^-3
DISCRETIZATION APPROACHES -LINEAR APPROXIMATION(1600)
2
0
−2
−4
−6
−8
−10
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8


x 10^-3
DISCRETIZATION APPROACHES -LINEAR APPROXIMATION(3200)
1
0
−1
−2
−3
−4
−5
−6
−7
−8
−9
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8