最小絶対偏差推定量のための効率的算法

近藤 康之
早稲田大学政治経済学術院
ykondo@waseda.jp

2005 年 8 月

1 はじめに

本稿の目的は，線形回帰モデルにおける最小絶対偏差推定量（least absolute deviations estimator, LADE）[1, §4.6] を計算するための算法の効率性について検討することである．LADE を求めるための算法としては，線形計画問題（linear program, LP）によるものが標準的であり，そこでは単体法（simplex method）に基づく算法が使用されることが多い．これまでに提案された方法は，LADE を定義する最小化問題を LP に書き換えた上で，一般的な LP のための単体法を LADE の特殊な構造を利用して改善したものである [2, 3]．本稿では，LADE のための特別な算法を開発するよりも，広く普及している算法（主単体法，双対単体法，有界変数法）を比較することで，それらの特徴を明らかにする．

数値計算（シミュレーション）に基づく比較により，主単体法よりも双対単体法に基づく有界変数法の方が効率的であること，主単体法に基づく算法の非効率さは初期可能解の与え方に大きく依存すること，が明らかになった．

2 最小絶対偏差推定量を定義する線形計画問題

次の回帰モデルを考える．

$$y_i = x_i'\beta + \varepsilon_i \quad (i = 1, \ldots, n). \tag{1}$$

ここで，y_i はスカラの被説明変数，x_i は $k \times 1$ の説明変数ベクトル，β は $k \times 1$ のパラメタ・ベクトル，ε_i はスカラの攪乱項である．また，上付き添字 $'$ は行列・ベクトルの転置

て主可能基底を求める必要があるが，これは容易には求められない．したがって，２段階単体法と同様の方法で初期可能基底を求める [4, pp. 141–142]．他方，双対単体法に基づく算法のための初期基底としては，次に述べるように，比較的容易に双対可能基底を求めることができるので，これを用いる．

X' のある基底集合を $B \subset \mathcal{N} = \{1, \ldots, n\}$ とする．すなわち，$|B| = k$ であり，k 個の列ベクトル $x_i\ (i \in B)$ を並べた基底行列 X'_B は非特異である．以下では，文脈から判断して誤解の生じない限り，基底集合 B と基底行列 X'_B の両方を基底と呼ぶ．非基底 $N = \mathcal{N} \setminus B$ の $n-k$ 個の列を並べた $k \times (n-k)$ 行列を X'_N とする．$\mathcal{N}$ の基底 B と非基底 N への分割に対応して，

$$X' = \begin{bmatrix} X'_B & X'_N \end{bmatrix}, \quad y' = \begin{bmatrix} y'_B & y'_N \end{bmatrix}, \quad \zeta = \begin{bmatrix} \zeta_B \\ \zeta_N \end{bmatrix}, \quad w = \begin{bmatrix} w_B \\ w_N \end{bmatrix}$$

と分割する．

LP (6) の制約式 $X'\zeta = X'w$ と目的関数 $y'\zeta$ は，この分割に沿って次のように書き換えられる．

$$\begin{aligned} X'\zeta &= X'_B\zeta_B + X'_N\zeta_N = X'w \\ X'_B\zeta_B &= X'w - X'_N\zeta_N \\ \zeta_B &= (X'_B)^{-1}(X'_B w_B + X'_N w_N - X'_N\zeta_N) \\ &= w_B + (X'_B)^{-1}X'_N(w_N - \zeta_N), \qquad (7) \\ y'\zeta &= y'_B\zeta_B + y'_N\zeta_N \\ &= y'_B\{w_B + (X'_B)^{-1}X'_N(w_N - \zeta_N)\} + y'_N\zeta_N \\ &= y'_B\{w_B + (X'_B)^{-1}X'_N w_N\} + \{y'_N - y'_B(X'_B)^{-1}X'_N\}\zeta_N. \qquad (8) \end{aligned}$$

また，被約費用ベクトル d'_N を，

$$d'_N = y'_N - y'_B(X'_B)^{-1}X'_N \qquad (9)$$

と定義する．基底 X'_B に対する基底解は，

$$\zeta_B = w_B + (X'_B)^{-1}X'_N(w_N - \zeta_N), \quad \zeta_N = \text{要素毎に } 0 \text{ または } 2w_N, \qquad (10)$$

とあらわされる．非基底変数 ζ_N について，

$$\zeta_i = \begin{cases} 0 & d_i \le 0 \text{ のとき} \\ 2w_i & d_i > 0 \text{ のとき} \end{cases} \quad (i \in N) \qquad (11)$$

とすると，この基底解は双対可能である．

ある．繰り返しを許した標本抽出なので，1回だけ抽出されている観察もあれば，複数回抽出される観察，あるいは1回も抽出されない観察がある．1つのブートストラップ標本中での抽出回数を数え上げたのが，表1の第2列である．

以上のように，1つのブートストラップ標本中での抽出回数 $w_1, w_2, \ldots, w_n$ を定義することで，ブートストラップ繰り返し計算の各回に解くべき LP は，これまでと同様に (6) としてあらわされる．制約式の右辺ベクトル $X'w$ と有界変数の上限 $2w$ がブートストラップ標本に依存して変化するので，元の標本に基づいて得られた最適基底解は，一般に主可能ではなく，また，双対可能でもない．したがって，ブートストラップ繰り返し計算の各回毎に，前節で述べた方法により初期可能基底解を求める必要がある．

■Bootstrap based on residuals　残差についてのブートストラップでは，元の標本に基づいて得られた残差から再抽出を行う．元の標本に基づいて得られた推定量を b_{LAD}，残差を $\hat{\varepsilon}_i = y_i - x_i' b_{\mathrm{LAD}}$ $(i = 1, \ldots, n)$ とする．この残差 $\hat{\varepsilon}_1, \hat{\varepsilon}_2, \ldots, \hat{\varepsilon}_n$ から繰り返しを許して抽出された大きさ n の標本を $\{\hat{\varepsilon}_1^*, \hat{\varepsilon}_2^*, \ldots, \hat{\varepsilon}_n^*\}$ とすると，ブートストラップ標本は $\{(x_1, x_1' b_{\mathrm{LAD}} + \hat{\varepsilon}_1^*), (x_2, x_2' b_{\mathrm{LAD}} + \hat{\varepsilon}_2^*), \ldots, (x_n, x_n' b_{\mathrm{LAD}} + \hat{\varepsilon}_n^*), \}$ である．すなわち，ブートストラップ繰り返し計算の各回に解くべき LP は，これまでと同様に (6) としてあらわされる．目的関数ベクトル y' のみが変化し，制約式と有界変数の上限は元の標本の場合と等しいので，元の標本に基づいて得られた最適基底解は，一般に双対可能ではないが，主可能である．したがって，ブートストラップ繰り返し計算の各回毎に2段階単体法を適用する必要はなく，主可能基底を初期基底解として，主単体法に基づく算法を適用することができる．他方，双対単体法に基づく算法では，前節末に述べた方法で可能基底解を求めれば良い．あるいは，元の標本に基づいて得られた最適基底に基づいて，非基底変数の値を変更することで可能基底解を求めても良い．

以上の考察から，算法という観点からは paired bootstrap を考察する必要の無いことが明らかとなったので，次節の計算では bootstrap based on residuals のみを考察の対象とする．

表 1: Paired bootstrap の例：標本の大きさが 5 の場合

ブートストラップ標本	$(w_1, w_2, w_3, w_4, w_5)$
(2, 1, 4, 2, 1)	(2, 2, 0, 1, 0)
(1, 4, 3, 5, 3)	(1, 0, 2, 1, 1)
(3, 5, 3, 2, 4)	(0, 1, 2, 1, 1)

表 2: 計算実験の結果：主単体法と双対単体法に基づく算法の比較

算法	n	k	計算時間	繰り返し回数	枢軸演算回数
主	100	5	0.221	153.1	111.7
双対	100	5	0.043	18.0	18.0
主	200	5	0.476	294.0	208.4
双対	200	5	0.062	22.7	22.7
主	500	5	1.528	713.0	499.0
双対	500	5	0.163	42.9	42.9
主	100	10	0.301	188.4	148.8
双対	100	10	0.085	35.1	35.1
主	200	10	0.666	361.0	279.2
双対	200	10	0.140	47.5	47.5
主	500	10	2.203	873.1	663.8
双対	500	10	0.309	73.4	73.4
主	100	20	0.444	233.8	193.4
双対	100	20	0.168	59.7	59.7
主	200	20	0.991	454.1	370.4
双対	200	20	0.268	82.0	82.0
主	500	20	3.407	1091.2	875.8
双対	500	20	0.783	160.7	160.7

表 3 の通りである．比較対象の 3 つの算法のうち，算法「双対」は上で検討したものと同じである．算法「主-1」は，主単体法に基づく算法であるが，２段階法ではなく，元の標本から得られた最適基底解を初期可能基底解として用いる点が，算法「主」との相違点である．算法「双対-1」では，算法「主-1」と同じ基底を初期値として用いるが，双対可能とするために算法「双対」と同様に非基底変数の値を定める．

算法「主-1」と「双対-1」とを比較すると，両者の効率が最も近い場合（$n = 100$ 最小値，$k = 20$ 最大値），「主-1」による枢軸演算回数は，「双対-1」の約 1.2 倍であり，最も差の大きい場合（$n = 500$ 最大値，$k = 5$ 最小値）では，約 1.7 倍である．上で述べた「主」と「双対」との比較と同様に，標本の大きさが小さいほど，説明変数の個数が大きいほど，「主-1」と「双対-1」の効率は接近する．しかし，効率の相違は高々 1.7 倍程度であり，「主」と「双対」の相違が最大で 18 倍であったことと比べると，驚くほど近いと言える．参考値として載せた計算時間については，説明変数の個数 k の増加に伴って効率が逆転す

ある．このことから，期待値に相当する元の標本から得られたLADEよりも，各ブートストラップ標本から得られるOLSEの方が，効率向上に資する初期値の基準であることが分かる．

6 まとめ

計算実験により，最小絶対偏差推定量を求めるための算法としては主単体法よりも双対単体法の方が効率的であること，最小２乗残差に基づく双対可能基底解は初期値として振る舞いの良いこと，が分かった．この結果は，攪乱項が自由度３のt分布に従うものとして行った計算実験から得られたものである．より裾の厚い分布，歪みのある分布など，最小２乗推定量と最小絶対偏差推定量との乖離が大きいと予想されるとき，どのような結果が得られるかを検討することは，興味深い課題である．てこ値を考慮するなどして，より振る舞いの良い初期値が与えられるかもしれない．

謝辞

本研究の遂行に際し，日本学術振興会科学研究費補助金（奨励研究(A)12730020）の助成を受けた．記して感謝いたします．

参考文献

[1] Amemiya, T. (1985) *Advanced Econometrics.* Cambridge, MA: Harvard University Press.

[2] Charnes, A., W. W. Cooper, and R. Ferguson (1955) "Optimal Estimation of Executive Compensation by Linear Programming," *Mathematical Science* 1(2), 138–151.

[3] Armstrong, R. D., and M. T. Kung (1982) "A Dual Algorithm to Solve Linear Least Absolute Value Problems," *Journal of Operational Research Society* 33, 931–936.

[4] 今野浩 (1987) 『線形計画法』日科技連.

[5] Maros, I. (2003) "A Generalized Dual Phasee-2 Simplex Algorithm," *European Journal of Operational Research* 149(1), 1–16.

[6] Shao, J., and D. Tu (1995) *The Jackknife and Bootstrap.* Springer-Verlag, New York.

[7] Greene, W. H. (2003) *Econometric Analysis, 5th ed.* Upper Saddle River, NJ: Prentice-Hall.

```
slope0 = X \ y;    % 係数ベクトルのＯＬＳ推定量
e = y - X * slope0;    % ＯＬＳ残差
[s,i] = sort( abs( e ) );
IdB = i(1:k);  IdB = IdB(:)';
slope = X( IdB, : ) \ y( IdB );
e = y - X * slope;
IdU = setdiff( find( e > eps ), IdB );
%
[ v3, x3, pi3, eflg3, cnt3, np3, IndBas3, IndNonU3 ] ...
  = dsimplexbv( X', RHS, y', UB2, IdB, IdU );
%
t21 = cputime;
% ========================================
r = r + 1;
LT( r, 1, id_n, id_k0 ) = t11 - t10;
NI( r, 1, id_n, id_k0 ) = cnt1 + cnt2;
NP( r, 1, id_n, id_k0 ) = np1 + np2;
LT( r, 2, id_n, id_k0 ) = t21 - t20;
NI( r, 2, id_n, id_k0 ) = cnt3;
NP( r, 2, id_n, id_k0 ) = np3;
%
% ========================================
% ブートストラップ
slope = pi3(:);
fit = X * slope;
resid = y - fit;
b = 0;
while ( b < NB );
  % データ行列（ブートストラップ標本）の作成
  yb = resid( unidrnd( n, n, 1 ) );
  %
  % ========================================
  % 主単体法
  t1b0 = cputime;
  [ v1b, x1b, pi1b, eflg1b, cnt1b, np1b ] ...
    = psimplexbv( X', RHS, yb', UB2, IndBas3, IndNonU3 );
  %
  t1b1 = cputime;
  % ========================================
  % 双対単体法（主単体法と同じ初期値から）
  t2b0 = cputime;
  % ~~~~ 初期値の作成
  slope = X( IndBas3, : ) \ yb( IndBas3 );
  e = yb - X * slope;
  IdU = setdiff( find( e > eps ), IndBas3 );
  %
  [ v2b, x2b, pi2b, eflg2b, cnt2b, np2b ] ...
    = dsimplexbv( X', RHS, yb', UB2, IndBas3, IdU );
  %
  t2b1 = cputime;
  % ========================================
  % 双対単体法（ＯＬＳ残差に基づく初期値から）
```

```
    cB = c( IndBas );  cN = c( IndNon );      % partition OBJ vector
20  BA = B \ A;  bb = B \ b;
    pi = cB / B;                              % simplex multiplier
    cost = cN - pi * N;                       % reduced cost vector
    pivot = 1;  DictToBeUpdated = 0;

25  xB = bb;
    if ( ~isempty( IndNonU ) );
      xB = xB - BA( :, IndNonU ) * u( IndNonU );  % basic solution
    end
    xN = zeros( n, 1 );  xN( IndNonU ) = u( IndNonU );
30  xN = xN( IndNon );                        % non-basic solution
    %
    while ( ~ExitFlg );
      % Basis, Matrices, and Vectors
      if ( DictToBeUpdated );
35      B = A( :, IndBas );  N = A( :, IndNon );  % partition LHS matrix
        cB = c( IndBas );  cN = c( IndNon );      % partition OBJ vector
        BA = B \ A;  bb = B \ b;
        pi = cB / B;                              % simplex multiplier
        cost = cN - pi * N;                       % cost vector
40      pivot = pivot + 1;
        DictToBeUpdated = 0;
      end;
      xB = bb;
      if ( ~isempty( IndNonU ) );
45      xB = xB - BA( :, IndNonU ) * u( IndNonU );  % basic solution
      end
      xN = zeros( n, 1 );  xN( IndNonU ) = u( IndNonU );
      xN = xN( IndNon );                          % non-basic solution
      %
50    % Optimality Checks
      cost_ab = cost .* ( ( xN == 0 ) - ( xN ~= 0 ) )';
      %% 被約費用を，ゼロの非基底はそのまま，上限値の非基底は符号を逆に
      % Find Max-Cost Column in Non-Basis
      [ MaxCost, JinNon ] = max( cost_ab );
55    JinAll = IndNon( JinNon );
      %
      % Optimality Checks
      if ( MaxCost < eps );
        ExitFlg = 1;
60    else;
        % Find Min-Ratio Row in Basis
        NN = BA( :, JinAll );
        IndPosRow = find( NN > eps );
        IndNegRow = find( NN < -eps );
65      %
        if ( ismember( IndNon( JinNon ), IndNonZ ) )
          [ MinRatPos, IinPos ] = min( xB( IndPosRow ) ./ NN( IndPosRow ) );
          [ MinRatNeg, IinNeg ]
            = min( ( xB( IndNegRow ) - u( IndBas( IndNegRow ) ) ) ./ NN( IndNegRow ) );
70        ubJ = u( JinAll );
```

```
    x( IndNonU ) = u( IndNonU );
    val = c * x;
125
    varargout = { val, x, pi, ExitFlg, cnt, pivot, IndBas, IndNonU };
```

A.3 双対単体法に基づく有界変数法のための M-function

```
 1  function  varargout = dsimplexbv( A, b, c, u, IndBas, IndNonU );

    [ m, n ] = size( A );

 5  % Making Index Vectors
    IndAll = ( 1:n );
    IndNon = setdiff( IndAll, IndBas );
    IndNonZ = setdiff( IndNon, IndNonU );
    % IndBas = 基底
10  % IndNon = 非基底
    %   IndNonU = 非基底のうち上限に一致するもの
    %   IndNonZ = 非基底のうちゼロのもの

    ExitFlg = 0;  cnt = 0;  pivot = 0;
15  MaxIter = max( 1000, 100 * m );

    % Basis, Matrices, and Vectors
    B = A( :, IndBas );  N = A( :, IndNon );  % partition LHS matrix
    cB = c( IndBas );  cN = c( IndNon );      % partition OBJ vector
20  BA = B \ A;  bb = B \ b;
    pi = cB / B;                              % simplex multiplier
    cost = cN - pi * N;                       % reduced cost vector
    pivot = 1;
    DictToBeUpdated = 0;
25
    xB = bb;
    if ( ~isempty( IndNonU ) );
      xB = xB - BA( :, IndNonU ) * u( IndNonU );  % basic solution
    end
30  xN = zeros( n, 1 );  xN( IndNonU ) = u( IndNonU );
    xN = xN( IndNon );                        % non-basic solution
    cost_ab = cost .* ( ( xN == 0 ) - ( xN ~= 0 ) )';
    %
    % Feasibility Checks
35  if ( any( cost_ab > eps ) );
      error( 'Initial Basis is NOT Primal Feasible!' );
    end;

    while ( ~ExitFlg );
40    % Basis, Matrices, and Vectors
      if ( DictToBeUpdated );
        B = A( :, IndBas );  N = A( :, IndNon );  % partition LHS matrix
        cB = c( IndBas );  cN = c( IndNon );      % partition OBJ vector
        BA = B \ A;  bb = B \ b;
```

```
      end
    end
    cnt = cnt + 1;
    if ( cnt >= MaxIter );
      ExitFlg = -2;
    end
  end

  % Making Solution Vector
  x = zeros( n, 1 );
  x( IndBas ) =  xB;
  x( IndNonU ) = u( IndNonU );
  val = c * x;

  varargout = { val, x, pi, ExitFlg, cnt, pivot, IndBas, IndNonU };
```