5.13　モンテカルロシミュレーション Bootstrap の適用例 ver.0.1

Bootstrap や Jackknife をリサンプリングの大きな枠組みとしてとらえて、データを分割して（例えば２分割したり逐次計算をしたり）分析をするクロスバリデーションの考え方と同じ類の分類ができるとすれば、不均一分散を調べる Goldfeld-Quandt テストも、構造変化を調べる CUSUM テストも同じような考え方をしているものと言えます。ここでは、具体的な Bootstrap の適用例の代表的なものをプログラムしていきます。

不均一分散のケース(Wu's Weighted Bootstrap)

この方法は、不均一分散の問題を解消する第３の方法です。まず通常の OLS で残差を求めます。これをもとに、残差を適当にウエイトして標準化させたものに標準正規乱数をかけあわせて、修正された残差 u*をもとめ、ここから Residual Bootstrap を times 回繰り返すものです。すなわち、

$$\hat{u} = y - X\hat{\beta}$$

$$u_i^* = \frac{\hat{u}_i}{\sqrt{1-\omega_i}} N(0,1), \quad where \quad \omega_i = x_i (X'X)^{-1} x_i'$$

$$y^* = X\hat{\beta} + u^*$$

の作業から、y*とＸを用いて通常の OLS を times 回だけ繰り返し計算します。

プログラム

```
new; cls;
rndseed 911;
load data[31,2]=d:datafile12.txt;
y=data[.,1]; x=data[.,2];
x_1=lagn(x,1); x_2=lagn(x,2); y_1=lagn(y,1);
x=ones(rows(data),1)~x_1~x_2~y_1;
y=y[3:31,.]; x=x[3:31,.];
call weightedboot(y,x,2000);

proc weightedboot(y,x,times);
    local n,k,b,e,s2hat,varb,se,w,i,ustar,ystar,bootb,bootmean,bootse,name;
    n=rows(x); k=cols(x);
/* OLS Estimates */
    b=inv(x'x)*x'y;
    e=y-x*b;
```

```
    s2hat=e'e/(n-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
/* Omega Computation */
    w=zeros(n,1);
    i=1;
    do while i<=n;
        w[i]=x[i,.]*inv(x'x)*x[i,.]';
        i=i+1;
    endo;
/* Parametric Bootstrap */
    bootb=zeros(times,k);
    i=1;
    do while i<=times;
        ustar=(e./sqrt(1-w)).*rndn(n,1);
        ystar=x*b+ustar;
        bootb[i,.]=( inv(x'x)*x'ystar )';
        i=i+1;
    endo;
    bootmean=meanc(bootb);
    bootse=stdc(bootb);
/* Comment and Display */
    name={"OLS Beta","OLS SE","Wu Beta","Wu SE"};
    print $name';;
    print b~se~bootmean~bootse;
    print/lz "# of data=" n;
    print/lz "# of repetition=" times;
    retp(b~se~bootmean~bootse);
endp;
```

画面表示

OLS Beta	OLS SE	Wu Beta	Wu SE
-7.8922653	2.6077444	-7.9135448	2.6284796
0.69276561	0.13253255	0.69396394	0.13440286
-0.63412016	0.12615757	-0.63472782	0.12417737
0.71261457	0.094276392	0.70915461	0.11428001

# of data= 29

# of repetition= 2000

時系列データのケースその１(Moving Block Bootstrap)

今までの Bootstrap では、データの並びは考慮に入れない非時系列のデータについての分析でした。しかしながら、時系列データでは、データの並びというのが重要な意味を持ってきます。これを可能な限り考慮に入れた Bootstrap がこの Moving Block です。すなわち、データの並びを Rolling するようにある一定の長さだけとっていき、それをそれぞれの塊として Bootstrap して、もとのサンプルの長さの系列を復元しようというものです。すこしわかりにくいかもしれませんので、とりあえずそのプロトタイプのプログラムを見てください。

プログラム

```
new; cls;
rndseed 911;
data={78,68,89,92,60,42,74,80,78}; times=4; block=3;
print/rz blockboot(data,times,block);

proc blockboot(data,times,block);
    local n,max,bn,index,i,j,temp,boot;
    n=rows(data);
    max=n-block+1;
    bn=ceil(n/block);
    index=zeros(n,times);
    i=1;
    do while i<=times;
        j=1; temp=seqa(ceil(rndu(1,1)*max),1,block);
        do while j<=bn-1;
            temp=temp|seqa(ceil(rndu(1,1)*max),1,block);
            j=j+1;
        endo;
        index[.,i]=temp[1:n];
        i=i+1;
    endo;
    @           print/rz index;      @
    i=1; boot=zeros(n,times);
    do while i<=cols(index);
        boot[.,i]=data[index[.,i]];
```

```
        i=i+1;
    endo;
    retp(boot);
endp;
```

画面表示

5	3	1	7
6	4	2	8
7	5	3	9
7	4	2	1
8	5	3	2
9	6	4	3
5	6	7	5
6	7	8	6
7	8	9	7

60	89	78	74
42	92	68	80
74	60	89	78
74	92	68	78
80	60	89	68
78	42	92	89
60	42	74	60
42	74	80	42
74	80	78	74

上のように、index も画面表示すると、サンプルの長さ n=9 に対して、Block の長さを仮に３として考えると、３つの続き番号を Bootstrap して、それを９÷３＝３の３個並べてもとのサンプルの長さを復元していることがわかると思います。上のプログラムでは、基本的に、並びの最初の点は１から max の n-block+1 までをとり得ると考えて、プログラム上、その始点の index 番号を一様乱数から取り出して、以下シークエンスで１つずつ増やして Block の長さ分だけ index の temp 番号をつくり､それをもとのサンプルが復元される長さまで繰り返しています。上の procedure を使って、１列ごとに時系列の統計量、例えば単位根計算やトレンド計算を行なって、それを並び替えて Percentile を取ってやると簡単にその統計量に対する Bootstrap 信頼区間が得られます。

時系列データのケースその２(Sieve Bootstrap)

Moving Block の考え方はデータを並びを極力保存して Boostrap しようというものでし

たが、この Sieve の考え方は、AR や ARMA などモデルが既知なものとして、そのモデルで一度パラメータを求めてしまってから、モンテカルロシミュレーションの設定の全く同じように、その基底となるランダムタームになるところまで戻って、そのパラメトリックな設定にもとづいてランダムタームを Bootstrap の回数だけスクランブルしてシミュレーションをしようというものです。モンテカルロシミュレーションの節の前半で取り上げたような正規乱数にまで立ち戻れるものは基本的に何でもこの方法が適用できます。

プログラム

```
new; cls;
rndseed 911;
load data[50,3]=d:datafile16.txt;
y=data[.,1]; x=ones(50,1)~data[.,2:3];
call sievehildlu(y,x,2000,0.05);

proc sievehildlu(y,x,times,pp);
    local n,k,bhat,bstar,u,bboot,i,rho,s2hat,se,serho;
    local rhohat,rhob,e,index,estar,j,ustar;
/* Estimate original parameters */
    {bstar,rho,s2hat,se,serho}=hildlu(y,x);
    n=rows(x); k=cols(x);
    bhat=bstar;
    u=y-x*bhat;
    rhohat=rho;
    e=u[2:n]-rhohat*u[1:n-1];
/* Residual Bootstrap */
    bboot=zeros(times,k); rhob=zeros(times,1);
    i=1;
    do while i<=times;
        index=ceil((n-1)*rndu(n-1,1));
        estar=e[index];
        ustar=u[1]|zeros(n-1,1);
        j=2;
        do while j<=n-1;
            ustar[j]=rhohat*ustar[j-1]+estar[j];
            j=j+1;
        endo;
        y=x*bhat+ustar;
```

```
        {bstar,rho,s2hat,se,serho}=hildlu(y,x);
        bboot[i,.]=bstar';
        rhob[i,.]=rho;
        i=i+1;
    endo;
/* Sort Bootstrap Estimates */
    i=1;
    do while i<=k;
        bboot[.,i]=sortc(bboot[.,i],1);
        i=i+1;
    endo;
    rhob=sortc(rhob,1); print rhob;
/* Comment and Display */
print/rz "        Bias beta          E(beta)" (1-pp)*100 "% ConfidenceInterval";;
print bhat~meanc(bboot)~bboot[round(pp/2*times),.]'~bboot[round((1-pp/2)*times),.]';
print/rz "        Bias rho           E(rho) " (1-pp)*100 "% ConfidenceInterval";
print rhohat~meanc(rhob)~rhob[round(pp/2*times),.]'~rhob[round((1-pp/2)*times),.]';
    retp(bboot);
endp;

proc (5) = hildlu(y,x);
    local n,k,b,e,rho,iter,length,start,finish,rhomax1,step,grid;
    local i,j,ystar,xstar,bstar,rhomax,estar,s2hat,varb,se,serho;
    n=rows(x); k=cols(x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    rho=e[1:n-1,.]'e[2:n,.]/e[1:n-1,.]'e[1:n-1,.];
    @   print "Iteration:      #          rho"; @
    @ print "                    1 ";; print rho; @
    iter=100; length=18;
    start=-0.9; finish=0.9;
    i=1; rhomax1=0;
    do while i<=iter;
        step=(finish-start)/length;
        grid=(-1e256).*ones(length+1,1);
        rho=start;
```

```
        j=1;
        do while j<=length+1;
            ystar=sqrt(1-rho^2)*y[1,.]|y[2:n,.]-rho*y[1:n-1,.];
            xstar=sqrt(1-rho^2)*x[1,.]|x[2:n,.]-rho*x[1:n-1,.];
            bstar=inv(xstar'xstar)*xstar'ystar;
            e=ystar-xstar*bstar;
            grid[j]=e'e;
            j=j+1; rho=rho+step;
        endo;
        rhomax=start+(minindc(grid)-1)*step;
        if abs(rhomax-rhomax1)<1e-8;
            break;
        endif;
        rhomax1=rhomax;
        start=rhomax-step; finish=rhomax+step;
        i=i+1; @ print/rz i;;print rhomax; @
    endo;
    rho=rhomax1;
    ystar=sqrt(1-rho^2)*y[1,.]|y[2:n,.]-rho*y[1:n-1,.];
    xstar=sqrt(1-rho^2)*x[1,.]|x[2:n,.]-rho*x[1:n-1,.];
    bstar=inv(xstar'xstar)*xstar'ystar;
    estar=ystar-xstar*bstar;
    s2hat=estar'estar/(n-k);
    varb=s2hat*inv(xstar'xstar);
    se=sqrt(diag(varb));
    serho=sqrt((1-rho^2)/(n-k));
    retp(bstar,rho,s2hat,se,serho);
endp;
```

画面表示

Bias beta	E(beta)	95 % ConfidenceInterval	
1.0117231	1.0120334	0.97412248	1.0499826
-0.98837295	-0.98895297	-1.0320830	-0.94705840
1.9896097	1.9903007	1.9515731	2.0298679
Bias rho	E(rho)	95 % ConfidenceInterval	
0.42222222	0.37446021	0.085091478	0.61111111

上の結果で E(beta)のところは、2000 回の基底ランダムタームの残差 Bootstrap を行なっ

て求めた係数を平均したものです。その Bootstrap 係数に対する Percentile による信頼区間をあわせて表示させています。Sieve は、このように、モデルに依存した形で Bootstrap する基底となるランダムタームをスクランブルして、既知のρから残差 u*を求め、これから既知のパラメータと u*からｙ*を復元して、これともともとのｘとをあらためて回帰するものです。Moving Block はモデルによらないものですが、この Sieve はモデルが正しく既知であることが前提で、そのパラメータを固定したものとして使います。上の計算では、AR の計算に grid search 法による Hildreth-Lu の方法を採用していますが、第３節で扱ったその他の方法の procedure に該当箇所を置き換えてもそのまま計算できます。

通常の非時系列の Likelihood による推定

5.11 では OLS についての３つの Bootstarp についてプログラムしましたが、そのうちの Parametric Bootstrap を残差に適用してスクランブルして、もう一度従属変数を復元してその復元された従属変数ともとの独立変数、そして最初の推定のパラメータ推定値を用いて、逐次的に Likelihood を最大化することによって、線形および非線形の Bootstrap 係数を推定しようというものです。

プログラム[要 CML]

```
new; cls;
library cml;
cmlset;
load data[29,2]=d:datafile1.txt;
data=data[2:29,.];
_cml_Bounds={-1e256 1e256,
                    -1e256 1e256,
                      0.001 1e256};
_cml_Algorithm = 1;
start={0,0,1};
rndseed 911;
call paraboot(data,200,0.05);

proc paraboot(data,times,pp);
    local bhat,f,g,cov,retcode,x,n,b,e,y,i,index,estar,ystar,bboot,bhatb,sortb,low,up;
    screen off;
    {bhat,f,g,cov,retcode}=cmlprt(cml(data,0,&ll,start));
    y=data[.,1]; x=ones(28,1)~data[.,2]; b=bhat[1:2]; n=rows(x);
    e=y-x*b;
    i=1; bboot=zeros(times,rows(bhat));
```

```
    do while i<=times;
        index=ceil(n*rndu(n,1));
        estar=e[index];
        ystar=x*b+estar;
        data=ystar~data[.,2];
        {bhatb,f,g,cov,retcode}=cmlprt(cml(data,0,&ll,start));
        bboot[i,.]=bhatb';
        i=i+1;
    endo;
    screen on;
    print "          Original b";
    print bhat;
    i=1; sortb=zeros(times,rows(bhat));
    do while i<=rows(bhat);
        bboot=sortc(bboot,i);
        sortb[.,i]=bboot[.,i];
        i=i+1;
    endo;
    low=round(times*pp/2);
    up=round(times*(1-pp/2));
    print "          E(beta)                95 % Confidence Interval";;
    print meanc(bboot)~sortb[low,.]'~sortb[up,.]';
    retp(bboot);
endp;

proc ll(b,data);
    local beta,s,y,x,e;
    y=data[.,1];
    x=ones(28,1)~data[.,2];
    beta=b[1:2];
    s=b[3];
    e=y-x*beta;
    retp(-1/2*ln(2*pi)-1/2*ln(s^2)-1/2*e^2/s^2);
endp;
```

画面表示

```
          Original b
```

77.795245
52.009819
50.427414

E(beta)	95 % Confidence Interval	
82.392275	40.703843	122.57860
51.308828	43.882788	57.999544
48.633581	36.363287	60.030404

上の計算では、CML を使っていますが、同様に Maxlik でも計算できます。ただし、残差を Bootstrap するので、かなりぶれてσがマイナスになるという非現実的なケースも考えられますので、CML でσに正値条件をつけて Log Likelihood を最大化する方がよいと思われます。一方、CML のコマンドを用いて Bootstrap するには次のようにします。

プログラム[要 CML]

```
new; cls;
library cml;
cmlset;
load data[29,2]=d:datafile1.txt;
data=data[2:29,.];
_cml_Bounds={-1e256 1e256,
                    -1e256 1e256,
                      0.001 1e256};
_cml_ParNames = {"CONST","X","SIGMA"};
_cml_Algorithm = 1;
start={0,0,1};
{x,f,g,cov,retcode}=cmlprt(cmlboot(data,0,&ll,start));

proc ll(b,data);
    local beta,s,y,x,e;
    y=data[.,1];
    x=ones(28,1)~data[.,2];
    beta=b[1:2];
    s=b[3];
    e=y-x*beta;
    retp(-1/2*ln(2*pi)-1/2*ln(s^2)-1/2*e^2/s^2);
endp;
```

画面表示

```
return code =       0
```

normal convergence

Mean log-likelihood -5.28306
Number of cases 50

Covariance of the parameters computed by the following method:
Bootstrapped covariance matrix

Parameters	Estimates	Std. err.	Gradient
CONST	78.1324	24.4584	0.0000
X	51.5974	5.1293	-0.0000
SIGMA	48.1281	6.4365	0.0000

Number of iterations 32
Minutes to convergence 0.00175

上のプログラムは cmlprt(cmlboot(data,0,&ll,start))というところが cmlboot を用いていて通常の cml だったところをこれに差し替えただけのものです。これにより Bootstrap した結果が表示されます。冒頭部分を library maxlik;として cml の接頭辞の付く名前をすべて max に変更すれば、maxlik でパラメータ制限なしに計算も可能です。

VAR Residual Bootstrap

時系列の場合でも Linear の VAR の場合には、基本的に OLS と計算は同じであって、残差を Bootstrap することによって、もともとの VAR のパラメータと独立変数、そして、このスクランブルした残差によって、

$$y^* = X\beta + e^*$$

を計算して、y*ともともとのXから VAR を times 回計算するものです。以下では、VAR の係数を求める部分だけプロトタイプとそれを times 回残差 Bootstrap させて、その数だけ係数をあらためて計算して、それを小さいものから大きいものに並び替えて Percentile 形式の９５%信頼区間を計算してみます。

プログラム

```
new; cls;
rndseed 911;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K; ddata=data[2:rows(data),.]-data[1:rows(data)-1,.];
```

```
{b,bootmeanb}=varboot(ddata,2,0.05,200);

proc(2)=varboot(data,lag,pp,times);
    local n,k,i,xlag,y,x,b,e,bboot,index,estar,ystar;
    local j,temp,bootmean,temp1,sorttemp,sortb,low,up;
    n=rows(data); k=cols(data);
    i=2; xlag=data[lag:n-1,.];
    do while i<=lag;
        xlag=xlag~data[lag+1-i:n-i,.];
        i=i+1;
    endo;
    y=data[lag+1:n,.];
    x=ones(n-lag,1)~xlag;
    b=inv(x'x)*x'y;
    print "Original b:" b;
    e=y-x*b;
/* Residual Bootstrap & Percentile of each estimates */
    i=1; bboot=zeros(rows(b)*cols(b),times);
    do while i<=times;
        index=ceil((n-lag)*rndu(n-lag,1));
        estar=e[index,.];
        ystar=x*b+estar;
        temp=inv(x'x)*x'ystar;
        j=2; temp1=temp[.,1];
        do while j<=cols(b);
            temp1=temp1|temp[.,j];
            j=j+1;
        endo;
        bboot[.,i]=temp1;
        i=i+1;
    endo;
    bootmean=meanc(bboot');
    sorttemp=zeros(times,rows(b)*cols(b));
    sortb=zeros(times,rows(b)*cols(b));
    i=1;
    do while i<=rows(b)*cols(b);
```

```
        sorttemp=sortc(bboot',i);
        sortb[.,i]=sorttemp[.,i];
        i=i+1;
    endo;
    low=round((pp/2)*times); up=round((1-pp/2)*times);
    sortb=sortb';
    print/rz "E(b) in a col" (1-pp)*100 "% ConfidenceInterval";;
    bootmean~sortb[.,low]~sortb[.,up];
    retp(b,bootmean);
endp;
```

画面表示

```
Original b:
        6.9607947         2.9751673         2.0938801
      0.059151587      -0.016030269       -0.20646028
      -0.50566020      -0.056169881         0.12189057
        1.6726696         0.59394204         0.95038666
        0.48186923         0.50952775         0.20496263
      -0.19541769       -0.38745010       -0.31405757
       -1.1082540        -1.3771054       -0.15801375
E(b) in a col                 95 % ConfidenceInterval
        7.0925680         1.7158755          12.180017
       0.054493490       -0.63169354         0.73597254
      -0.48822035        -1.5827520         0.62860965
        1.6430520        0.044643432          3.5791731
        0.47574894       -0.033921756          1.0543781
      -0.18734254        -1.3376302         0.72294041
       -1.0958460        -2.7189663         0.64734061
        3.0353982       -0.098225438          5.5212939
     -0.016323202        -0.31036513         0.30277952
     -0.055295341        -0.62360604         0.49117928
        0.57220015        -0.22806903          1.4349289
        0.51269902         0.19559290         0.80125925
      -0.39737162        -0.94529416        0.079506146
       -1.3742848        -2.2978644        -0.66302720
        2.1282864         0.41984808          3.7881122
      -0.20667350        -0.43892620     -0.0015765950
```

0.13090526	-0.22866524	0.50298903
0.93705718	0.39118406	1.4469174
0.20593174	-0.0062882107	0.38391892
-0.31394507	-0.61641341	-0.024511429
-0.15680761	-0.78061505	0.35417674

結果はもともとの係数の一列目の下に２列目を、その下に３列目をもっていった形にして、その脇にその係数に対する Percentile 信頼区間を示しています。プログラム的には、第３節後半のＶＡＲで取り扱った varmain の procedure を係数ｂだけを求めるように単純化して、そこから残差を求めて、上に述べた復元方法で y*に相当する部分を出してから、通常の定数項付きの線形ＶＡＲを times 回計算させています。時系列データですが、計算自体は残差の正規性を仮定した OLS を用いていますので、データの加工は切ったり貼ったり多少面倒ですが、このように通常の Residual Bootstrap が適用できます。

さらに、VAR や VARMA などには、Cholesky Factor Bootstrap といって、Cholesky 分解したものと Bootstrap した残差を用いて、Residual Bootstrap のように y*を復元する方法もあります。また、ソースが公開されている Bruce Hansen 教授の Grid　AR Bootstrap や、その他 Markov Chain Bootstrap、果ては State Space の Bootstrap などがあります。それらについては後ほど、適宜、取り上げていきたいと思います。