3.16 プログラミング 単位根とテスト(2) ver.0.1

ここでは、Structural Breakがある時の単位根テストについてのいくつかの方法についてプログラムを試みます。あわせて、最適なラグの数を求めるアルゴリズムを考えます。前章とは打って変わって、主にADFにもとづいた簡単なテストに各種ダミー変数を加えて、それを繰り返しループで回すことになります。

１）最適ラグ数の計算

ADFベースのテストにおける最適ラグ数の設定は試行錯誤が必要な箇所です。ある人は経験的に四半期ならその倍の８期まで、月次であれば24期までの、すべてのラグをチェックするかもしれません。最大の目的は単位根をテストすることにあるのですから、そうしたすべての場合をチェックするのは、けして悪いアプローチではないでしょう。それに対して、ある基準でラグ数を計算して決めてしまう方法もあります。モデルセレクションの応用としてAICやSCを用いることは悪くはありませんが、一般にそうして設定されたラグ数は小さすぎるということが知られていて、ラグ数計算にはAICの変形バージョンなどが考えられています。ここでは、パッケージソフトなどに用いられている代表的な３つの方法を考えます。

Schwertの経験則

この方法は、データに依存するのではなくて、データの数から経験的に最大考えられるラグ数を計算するというものです。データ数Nが与えられた時、ラグ数＝floor$(4(N/100)^{2/9})$を計算します。

プログラム(ADF Testの冒頭部分の変更)

```
    new; cls;
    load data[40,3]=d:datafile13.txt;
    Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
    print "Series of Y";
    call adf(Y);


proc(0)=adf(y);
  local lag,a;
  lag=floor(4*(rows(y)/100)^(2/9));        /* Schwert's rule of thumb */
  a=1;
  do while a<=lag;
     print;
     print "===========";
     print/lz "LAG=" a;
```

```
        print "===========";
        call adftest(y,a);
        a=a+1;
    endo;
endp;
．　．　．　以下略　　．　　．　　．
```

いままでADFタイプの単位根テストで３階建てにしていたprocの一番最上階のところにラグ数を計算するアルゴリズムを書き加えます。上の変更は、前章のADFのプログラムの冒頭の部分でlagの変数にラグを既知なものとして代入していたものを、そのインプットをなくしてｙだけの１変数インプットに直した上で、そのproc内部にラグ数を計算する式を加えたものです。インプットの変数lagがなくなる代わりに、ローカル変数にlagが加わり１式書き加えただけです。なお、式の中で2/9ではなくて1/4が採用されることもあります。上のプログラムは、ラグ数１から計算で定まったラグ数までのすべてを計算するものです。（なお定まったラグ数だけを計算させるには、a=1;からendo;までの代わりに、call adf(y,lag);だけにして、使わないaのローカル宣言をなくしてください。）

Perronの方法

こちらの方法は、データに依存したもので、最大考えられるラグ数（通常例えば12とか８）から、Augmentedタームの最大のラグ項が有意であるか調べるもので、ラグ数を最大ラグ数から１つずつ減らしていき、はじめて有意になったラグ数を最適ラグ数に採用するものです。ここで有意かどうか判断するのには、ｔ値やADFのtau値ではなくて、絶対値が1.6前後の独自のカットオフポイントを採用します。

プログラム(ADF Testの冒頭部分の変更)

```
    ．　．　．　．呼出し部省略　．　．　．　．
proc(0)=adf(Y);
    local a,lag,n,dy,i,j,x,rhs,temp,tau;
    lag=8;                              /* Begin with lag 12 usually. */
    do while lag>=1;
        n=rows(y);
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        rhs=x~ones(n-j+1,1)~seqa(1,1,n-j+1);
```

```
        temp=rhs[.,1]; rhs[.,1]=rhs[.,cols(x)]; rhs[.,cols(x)]=temp;
        print/lz lag;; tau=dftau(dy[j-1:n-1],rhs);
        if abs(tau)>1.6;                /* The last lag is significant. */
            break;
        endif;
        lag=lag-1;
    endo;
    print "-------------------------------------------------------";
    print/lz "           Choice of Lag # =     " lag;
    print "-------------------------------------------------------";
    a=0;
    do while a<=lag;
        print;
        print "============";
        print/lz "LAG=" a;
        print "============";
        call adftest(y,a);
        a=a+1;
    endo;
endp;
```

．　．　．　．　以下略　　．　　．　　．

上のプログラムでは、Schwertの経験則と同様に、ADFのタイプのテストの３階建てのprocの最上階を変更します。通常は12ラグから始めるのが普通ですが、データの特殊性と個数からここでは８を使うことにします。ループ構造は、do while lag>=1;からラグchoiceの画面表示のprint文の前のendo;までが、ラグを最大ラグ数から１つずつ減らしていくもので、その内部では、ADFの２階部分のドリフトとトレンドの両方がある場合のτ計算と同じアルゴリズムをもってきています。内側のループはADFのものと同じです。ただし、ほしいのはτではなくて、Augmentedタームの最終ラグのｔ値に相当するものですから、tempという一時変数を考えて、最終ラグとτの項の２列間の置き換えを行なっています。ADFの計算では直接右辺のタームを～のしるしでマージして代入していましたが、ここではわかりやすいように、rhsという変数を一度作成してから、それを右辺の独立変数側のタームとして一括代入しています。これによって、ADFのプログラムの１階部分がそのまま使えるようになります。そして、Augmentedタームの最終項のｔ値に相当するものがτに相当するものとなっていますから、この値の絶対値を1.6というカットオフポイントと比べて、それより大きい場合にはbreakで外側のループを抜けて、そこでラグ数を減らすことを中止します。最後まで1.6よりも大きくならなければ、ラグ数は１に設定するようになっています。

なお、a=0;からendo;までのループは、設定されたラグ数までのすべてのラグを計算するもので、もし必要がなければ、Schwertの経験則の場合と同じく、call adf(y,lag);だけにしてください。

ポイント　AとBの置き換え

temp=A;　A=B;　B=temp;

Aの中味を一度tempに置き、BをAに代入。最後に、Bにtempを代入する。

LM(F version)の方法

さらに考えられる方法としては、系列相関のところで扱ったLMテストのうちF versionのものを採用する方法があります。こちらは、ラグ数が１から順番に増やしていき、LM統計量が、例えば、初めて10%の有意水準のカットオフポイントよりも大きくなったとき、すなわち、その場合のP値が初めて10%よりも小さくなったときにループを止めてそのラグ数を採用します。プログラムは、一部変数の付け替えを除いて、ほとんど系列相関の章で扱ったLM Testの後半のpF'によるものをADFのprocの１階部分にもってきます。

プログラム

```
．　．　．　．呼出し部省略　．　．　．　．
proc(0)=adf(y);
    local lag,n,dy,i,j,x,rhs,lhs,nn,b,e,ematrix,e0,y1,x1,b1,n1,e1,pf,pv1;
    lag=1;                    /* Begin with lag 1. */
    do while lag<=12;      /* Max lag is considered as 12. */
        n=rows(y);
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        rhs=x~ones(n-j+1,1)~seqa(1,1,n-j+1);
        lhs=dy[j-1:n-1];
        nn=rows(rhs);
        b=inv(rhs'rhs)*rhs'lhs;
        e=lhs-rhs*b;
        ematrix=zeros(nn-lag,lag);
        j=1;
```

```
        do while j<=lag;
            ematrix[.,j]=e[lag+1-j:nn-j,.];
            j=j+1;
        endo;
        e0=e[lag+1:nn,.];
        y1=e0; x1=rhs[lag+1:nn,.]~ematrix;
        b1=inv(x1'x1)*x1'y1;
        n1=rows(x1);
        e1=y1-x1*b1;
        pf=(e'e-e1'e1)/e'e*nn;
        pv1=cdfchic(pf,lag);
        print/lz "LAG=" lag;print/lo "pF'=" pf;;print "Prob>X2" pv1;
        if pv1<0.1;
            break;
        endif;
        lag=lag+1;
    endo;
    print "-------------------------------------------------------------";
    print/lz "           Choice of Lag # =     " lag;
    print "-------------------------------------------------------------";
    call adftest(y,lag);
endp;
```

. . . . 以下略 . . .

画面表示

Series of Y

LAG= 1

pF'= 0.94418203 Prob>X2 0.33120465

LAG= 2

pF'= 2.5010342 Prob>X2 0.28635669

LAG= 3

pF'= 2.0487997 Prob>X2 0.56234016

LAG= 4

pF'= 2.7594120 Prob>X2 0.59885902

LAG= 5

pF'= 15.904726 Prob>X2 0.0071214150

Choice of Lag # = 5

．．．．以下略 ．．．

上のプログラムは、最後のラグ数「まで」ではなくて、そのラグ数だけのADFを求めるプログラムにしてあります。Choice of Lagの表示のところの後にa=0;からend;までを挿入してcall adftest(y,lag);に置きかえれば、ラグ１からそのラグ数まですべてを求めるものになります。上のいずれの場合にせよ、手動で、余裕をもって考えられるすべてのラグ数について単位根の有無をテストするのが何にも優ります。ただし、ラグ数とテストのパワーの間にはトレードオフの関係があることも忘れてはなりません。ラグ数を多くしすぎると検出力は一般に減少します。したがってSchwertの経験則も捨て難いものです。

２）構造変化のある時の単位根テスト

ここでは、ADF によるテストをベースに、データに構造変化がある場合についての単位根テストをプログラムします。データに構造変化が含まれる場合、通常の単位根テストのパワーが低まると同時に、構造変化をの前後のそれぞれは棄却するものであっても、前後をあわせて１つの期間にしてしまうと棄却しなくなっていまうとされています。そういうわけで、構造変化のある場合、ダミー変数を応用してデータを２つに分けてしまうことが重要です。２個や複数個の構造変化の場合のプログラムも存在しますが、ここでは１箇所だけの構造変化だけを考えることにしましょう。

BLS(Banerjee,Lumsdaine,Stock)Sequential Test

通常のADFのテストに対して、構造変化のダミー変数を１つだけ加えて、ADF同様にτを計算するものである。ここでは、簡単化のために、トレンド項の変化のダミーとドリフト項のダミーをそれぞれ別々に考える。すなわち、

トレント項のダミーの場合には、

D=t if t＝k+1,k+1, … N

D=0 if t＝1,2, … ,k

を考えて、構造変化があるとされる期ｋの次の期からは、もともとのトレンド項のｔの列がその２倍の2ｔになるようにダミー変数を設定するのに対して、それ以前はｔのままにするというものです。

ドリフト項ダミーの場合には、

D=1 if t＝k+1,k+1, … N

D=0 if t＝1,2, … ,k

を考えて、構造変化があるとされる期ｋの次の期からは、もともとの定数項の１の列がその２倍の２になるようにダミー変数を設定するのに対して、それ以前は１のままにするというものです。これらは別々なものとして、それぞれダミー変数設定をします。その際、

構造変化のある期ｋについてはあらかじめわかっていないものとして、計算上Singularになることを避けるために、全体のデータの系列の個数の約15％のところから約85%のところまでを繰り返し計算で逐次計算させるものです。

プログラム

```
    new; cls;
    load data[40,3]=d:datafile13.txt;
    Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
    print "BLS Sequential Test";
    call bls(Y,2);

proc(0)=bls(y,lag);
  local a;
  a=1;
  do while a<=lag;
     print; print/rz "LAG=" a; call blsseq(y,a);
     a=a+1;
  endo;
endp;

proc(0)=blsseq(y,lag);
  local n,dy,d2y,d3y,i,j,x,m,t,dt,dd;
  n=rows(y);
  dy=y[2:n]-y[1:n-1];
  i=1; j=2+lag; x=y[j-1:n-1];
  do while i<=lag;
     x=x~dy[j-1-i:n-1-i];
     i=i+1;
  endo;
  print "=============================================";
  m=round(0.15*n); print/rz "Period=" m "-" n-m;
  print "=============================================";
  print "-----------";
  print "Trend Break";
  print "-----------";
  t=0;
  do while t<=n-2*m;
```

```
        dt=zeros(m+t,1)|seqa(m+t+1,1,n);
        print/lz "break=" m+t;
        call dftau(dy[j-1:n-1], x~ones(n-j+1,1)~seqa(j-1,1,n-i)~dt[j-1:n-1]);
        t=t+1;
    endo;
    print;
    print "-----------";
    print "Drift Break";
    print "-----------";
    t=0;
    do while t<=n-2*m;
        dd=zeros(m+t,1)|ones(n,1);
        print/lz "break=" m+t;
        call dftau(dy[j-1:n-1], x~ones(n-j+1,1)~seqa(j-1,1,n-i)~dd[j-1:n-1]);
        t=t+1;
    endo;
endp;

proc dftau(dy,x);
    local b,e,n1,k,s2hat,varb,se,tau;
    n1=rows(x); k=cols(x);
    b=inv(x'x)*x'dy;
    e=dy-x*b;
    s2hat=e'e/(n1-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
    tau=b[1]./se[1];
    print "tau=" tau;;print/rz "( n=" n1 ")";
    retp(tau);
endp;
```

上のプログラムでは、ラグ数はマニュアルで２と設定して、１から２までのすべてのラグについて逐次計算させています。もちろん、ラグ数の設定で行なったように３階建ての最上階を変更することで、自動計算にすることも簡単に可能です。なお、３階建ての１階部分はADF計算とまったく同じです。最上階も名前は違いますが同じです。それらの中間の２階部分をTrend BreakとDrift Breakのそれぞれについて１つのダミー変数を加えて、その期ｋの位置を逐次計算で、m=round(0.15*n);とするとき、プログラムではもともとのダ

ミー変数をｋ＝m期に構造変化があるものとして、それをｔ＝０からn-2mまで１つずつ増していく形をとっています。ここでｔは実際の期ではなくて、最初のm期から数えた数です。したがって、実際の期の数を画面表示するには、print m+t;とするわけです。トレンドダミーの方は、dt=zeros(m+t,1)|seqa(m+t+1,1,n);をADFの右辺の最終項に加えます。ドリフトダミーの方も同様に、dd=zeros(m+t,1)|ones(n,1);を加えます。ただし、行の最後は適当に最大限加えられていますから、マージする際に、j-1からn-1までというふうに、従属変数の有効行数と同一にそろえます。

BLS(Banerjee,Lumsdaine,Stock) Rolling Test

今度はもう少し単純で全体のデータを３分の１に分割して最初のサブサンプルを用います。その分割したサブサンプルを、前から順に１つずつづらして、ローラー作戦で１つずつADFのテストをするものです。ただし、この方法は以下の例のように全体のデータサンプル数が十分にないときには避けた方が無難です。なぜなら各サブサンプルのデータ数からラグ数とさらに階差をとるための１を引いた実質のデータ数が10個前後まで落ち込んでしまうと、データが十分ではなくて、通常はマイナスのτの値もプラスにふれることもあるためです。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
call blsroll(L,2);

proc(0)=blsroll(y,lag);
    local a,n,s,t,ys;
    a=1;
    do while a<=lag;
        n=rows(y);
        s=round(1/3*n);
        print "=======";
        print/lz "s=" s;
        print/lz "lag=" a;
        print "=======";
        t=0;
        do while t<=n-s;
            ys=y[t+1:t+s];
            print/lz "Period from t=" t;;
```

```
            print; print/rz "LAG=" a; call adftest(ys,a);
            t=t+1;
        endo;
        a=a+1;
    endo;
endp;

proc(0)=adftest(y,lag);
    local n,dy,d2y,d3y,i,j,x;
    n=rows(y);
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    call dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(j-1,1,n-i));
endp;

proc dftau(dy,x);
    local b,e,n1,k,s2hat,varb,se,tau;
    n1=rows(x); k=cols(x);
    b=inv(x'x)*x'dy;
    e=dy-x*b;
    s2hat=e'e/(n1-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
    tau=b[1]./se[1];
    print "tau=" tau;;print/rz "( n=" n1 ")";
    retp(tau);
endp;
```

上の例はデータが短すぎてよくありませんが、データ数が十分に多いと仮定した上で、

```
    s=round(1/3*n);
```

というふうに、まずデータを３分割して四捨五入で正数化します。そしてt=0からn-sまで、

```
    ys=y[t+1:t+s];
```

のようにｙのシリーズのt+1からt+sまでのサブサンプルを作って、それをADFのprocに代入する作業をｔを１つずつ増加させて行ないます。サブサンプルを作っているだけなのでADFのprocの３階建ての構造のうち１、２階はそっくり利用します。（２階の部分は、必要のない部分は消して、通常の１階の系列のドリフトとトレンドの両方がある部分だけを使います。）最上階だけをサブサンプルをrollingして１つずつづらすものに作り変えています。ラグ数はマニュアルで設定するものとして、１から最大ラグ数までADFのτを計算するループの内側に、さらにｔを０からn-sまで１つずつ動かすループを入れています。

Perron Test

このテストは、構造変化があらかじめ定まった第ｋ期にあるとした上で、トレンド定常のalternativeに対してADFにもとづくテストを行なうものです。テストするモデルは、

ADF Model: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \varepsilon$

をベースにして、DU、DTB、DT*のダミー変数を加えた次のモデルをテストします。

ドリフトの構造変化

Model A: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \theta DU_t + \delta DTB_t + \varepsilon$

トレンドの構造変化

Model B: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \lambda DT_t^* + \varepsilon$

ドリフトおよびトレンドの両方の構造変化

Model C: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \theta DU_t + \delta DTB_t + \lambda DT_t^* + \varepsilon$

ここで、構造変化のある期をｋとすると、上のダミー変数は

DUt＝1　if t＝k+1,k+1, … N

DUt＝0　if t＝1,2, … ,k

DTBt＝1　if t＝k+1

DTBt＝0　otherwise

DTt*=t-k　if t＝k+1,k+1, … N

DTt*=0　if t＝1,2, … ,k

というようになります。プログラムでテストすべきモデルは上の３つなのですが、元になる帰無仮説とトレンド定常である対立仮説は、それぞれ、

Model A: (Null)　$y_t = \mu + d DTB_t + y_{t-1} + \varepsilon$

(Alternative)　$y_t = \mu_1 + \beta t + (\mu_2 - \mu_1) DU_t + \varepsilon$

Model B: (Null)　$y_t = \mu_1 + y_{t-1} + (\mu_2 - \mu_1) DU_t + \varepsilon$

(Alternative)　$y_t = \mu_1 + \beta t + (\beta_2 - \beta_1) DT_t^* + \varepsilon$

Model C: (Null)　$y_t = \mu_1 + y_{t-1} + d\,DTB_t + (\mu_2 - \mu_1)DU_t + \varepsilon$

(Altenative)　$y_t = \mu_1 + \beta t + (\mu_2 - \mu_1)DU_t + (\beta_2 - \beta_1)DT_t + \varepsilon$

ただし、ここで使われているDT*とは違って、

DTt=t　if t＝k+1,k+1, … N

DTt=0　if t＝1,2, … ,k

とします。対立仮説alternativeはどのモデルもすべてtrend stationaryとなります。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
print "Series of Y";
tb=25; /* at the time when a structural break occurs. */
call perrontest(y,tb);


proc(0)=perrontest(y,tb);
    local a,lag,n,dy,i,j,x,rhs,temp,tau;
    lag=8;                              /* Begin with lag 12 usually. */
    do while lag>=1;
        n=rows(y);
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        rhs=x~ones(n-j+1,1)~seqa(1,1,n-j+1);
        temp=rhs[.,1]; rhs[.,1]=rhs[.,cols(x)]; rhs[.,cols(x)]=temp;
        print/lz lag;; tau=dftau(dy[j-1:n-1],rhs);
        if abs(tau)>1.6;
            break;
        endif;
        lag=lag-1;
    endo;
    print "-------------------------------------------------------";
    print/lz "           Choice of Lag # =     " lag;
    print "-------------------------------------------------------";
```

```
    a=1;      /* ADF Type only */
    do while a<=lag;
        print;
        print "===========";
        print/lz "LAG=" a;
        print "===========";
        call perron(Y,a,tb);
        a=a+1;
    endo;
endp;

proc(0)=perron(y,lag,tb);
    local n,dy,d2y,d3y,i,j,x,d,dtb;
    print "Perron Test(MODEL A):";
    n=rows(y);
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    dtb=zeros(n,1); dtb[tb+1]=1;
    d=(zeros(tb,1)|ones(n-tb,1))~dtb;
    call dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
    print "Perron Test(MODEL B):";
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    d=zeros(tb,1)|seqa(1,1,n-tb);
    call dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
    print "Perron Test(MODEL C):";
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
```

```
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    dtb=zeros(n,1); dtb[tb+1]=1;
    d=(zeros(tb,1)|ones(n-tb,1))~(zeros(tb,1)|seqa(1,1,n-tb))~dtb;
    call dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
endp;

proc dftau(dy,x);
    local b,e,n1,k,s2hat,varb,se,tau;
    n1=rows(x); k=cols(x);
    b=inv(x'x)*x'dy;
    e=dy-x*b;
    s2hat=e'e/(n1-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
    tau=b[1]./se[1];
    print "tau=" tau;;print/rz "( n=" n1 ")";
    retp(tau);
endp;
```

画面表示

```
Series of Y
 8                tau=       -0.75418992 ( n=                30 )
 7                tau=        0.55518011 ( n=                31 )
 6                tau=       -0.16113311 ( n=                32 )
 5                tau=       -0.72944639 ( n=                33 )
 4                tau=       -0.92098122 ( n=                34 )
 3                tau=       -0.50235089 ( n=                35 )
 2                tau=        0.83144818 ( n=                36 )
 1                tau=         1.8370426 ( n=                37 )
------------------------------------------------------------
          Choice of Lag # =       1
------------------------------------------------------------
```

==========

LAG= 1

==========

Perron Test(MODEL A):

tau= -2.1994655 (n= 37)

Perron Test(MODEL B):

tau= -3.2871722 (n= 37)

Perron Test(MODEL C):

tau= -3.2000651 (n= 37)

ZA Test(Zivot,Andrews)

このテストは、Perron Testとは違って、データに依存して構造変化の期を繰り返しによって探し出すものです。Perron Testの３つのモデルをそのまま利用してもよいのですが、ここでは、Perron TestのそれぞれのモデルからDTBのダミー変数を取り除いたものをテストします（Model BにはもともとDTBのタームは含まれていないので同じになります)。

ドリフトの構造変化

Model A: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \theta DU_t + \varepsilon$

トレンドの構造変化

Model B: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \lambda DT_t^* + \varepsilon$

ドリフトおよびトレンドの両方の構造変化

Model C: $\Delta y_t = \mu + \alpha y_{t-1} + \beta t + \sum_{j=1}^{lag} \gamma_j \Delta y_{t-j} + \theta DU_t + \lambda DT_t^* + \varepsilon$

ここで、構造変化のある期を仮にｋとすると、上のダミー変数は

DUt＝1 if t＝k+1,k+1, … N

DUt＝0 if t＝1,2, … ,k

DTt*=t-k if t＝k+1,k+1, … N

DTt*=0 if t＝1,2, … ,k

というようになります。このｋが計算上Singularにならないように動かします。通常は10%やBLS Sequentialと同じ前後15%を除いた区間でｋを動かします。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
print "Series of Y";
```

```
    call zatest(y,4);

proc(0)=zatest(y,option);
    local lag,n,dy,i,j,x,rhs,temp,tau;
    lag=8;                              /* Begin with lag 12 usually. */
    do while lag>=1;
        n=rows(y);
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        rhs=x~ones(n-j+1,1)~seqa(1,1,n-j+1);
        temp=rhs[.,1]; rhs[.,1]=rhs[.,cols(x)]; rhs[.,cols(x)]=temp;
        print/lz lag;; tau=dftau(dy[j-1:n-1],rhs);
        if abs(tau)>1.6;
            break;
        endif;
        lag=lag-1;
    endo;
    print "-------------------------------------------------------";
    print/lz "          Choice of Lag # =     " lag;
    print "-------------------------------------------------------";
    call za(y,lag,option);
endp;

proc(0)=za(y,lag,option);
    local n,dy,d2y,d3y,i,j,x,d,dtb,tb,tau,vectau,mintau,mintb;
    n=rows(y);
    vectau=-9999*ones(n,1);
    tb=round(0.15*n);
    do while tb<n-round(0.15*n);
    print/lz "Break at=" tb;
    if option==1;
        print "Perron Version(Model A):";
```

```
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    dtb=zeros(n,1); dtb[tb+1]=1;
    d=(zeros(tb,1)|ones(n-tb,1))~dtb;
    tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
elseif option==2;
    print "Perron Version(Model B):";
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    d=zeros(tb,1)|seqa(1,1,n-tb);
    tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
elseif option==3;
    print "Perron Version(Model C):";
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
        i=i+1;
    endo;
    dtb=zeros(n,1); dtb[tb+1]=1;
    d=(zeros(tb,1)|ones(n-tb,1))~(zeros(tb,1)|seqa(1,1,n-tb))~dtb;
    tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
elseif option==4;
    print "ZA Version(Model A):";
    dy=y[2:n]-y[1:n-1];
    i=1; j=2+lag; x=y[j-1:n-1];
    do while i<=lag;
        x=x~dy[j-1-i:n-1-i];
```

```
            i=i+1;
        endo;
        d=(zeros(tb,1)|ones(n-tb,1));
        tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
    elseif option==5;
        print "ZA Version(Model B):";
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        d=zeros(tb,1)|seqa(1,1,n-tb);
        tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
    elseif option==6;
        print "ZA Version(Model C):";
        dy=y[2:n]-y[1:n-1];
        i=1; j=2+lag; x=y[j-1:n-1];
        do while i<=lag;
            x=x~dy[j-1-i:n-1-i];
            i=i+1;
        endo;
        d=(zeros(tb,1)|ones(n-tb,1))~(zeros(tb,1)|seqa(1,1,n-tb));
        tau=dftau(dy[j-1:n-1],x~ones(n-j+1,1)~seqa(1,1,n-j+1)~d[j:n,.]);
    else;
        errorlog "ERROR:Option number must be 1 through 6.";
    endif;
    vectau[tb]=tau;
    tb=tb+1;
    endo;
    mintau=maxc(vectau); mintb=maxindc(vectau);
    print "=====================";
    print/lz "Suspect at period:" mintb; print mintau;
    print "=====================";
endp;
```

```
proc dftau(dy,x);
    local b,e,n1,k,s2hat,varb,se,tau;
    n1=rows(x); k=cols(x);
    b=inv(x'x)*x'dy;
    e=dy-x*b;
    s2hat=e'e/(n1-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
    tau=b[1]./se[1];
    print "tau=" tau;;print/rz "( n=" n1 ")";
    retp(tau);
endp;
```

画面表示

```
Series of Y
 8                tau=      -0.75418992 ( n=              30 )
 7                tau=       0.55518011 ( n=              31 )
 6                tau=      -0.16113311 ( n=              32 )
 5                tau=      -0.72944639 ( n=              33 )
 4                tau=      -0.92098122 ( n=              34 )
 3                tau=      -0.50235089 ( n=              35 )
 2                tau=       0.83144818 ( n=              36 )
 1                tau=        1.8370426 ( n=              37 )
------------------------------------------------------------
          Choice of Lag # =      1
------------------------------------------------------------
Break at= 6
ZA Version(Model A):
tau=        -1.6545857 ( n=                37 )
Break at= 7
ZA Version(Model A):
tau=        -1.9763869 ( n=                37 )
. . . . . 中略 . . . . . .
Break at= 31
ZA Version(Model A):
tau=        -2.6012210 ( n=                37 )
```

```
Break at= 32
ZA Version(Model A):
tau=        -3.0646147 ( n=                  37 )
=====================
Suspect at period: 6
        -1.6545857
=====================
```

上のプログラムでは、ADFのプログラムとprocの１階と３階部分は同じです。基本的に、１階部分はラグ変数を自動的に求めるものに差し替えて、２階部分を呼び出す名前を変更してあります。メインのzaという２階部分は、Perron Testのsequential版３つに加えて、ZA testの３つのモデルの計６つのoptionを第３番目のインプットで指定するようにしています。なお、冒頭のzatest(y,option)のOptionの値の設定は、次のようになります。

1	Perron	Model A	
2		B	
3		C	
4	ZA’s	Model A	
5		B	（Option 2と同じ）
6		C	

プログラム的には、Perron Testと同じで、６つのモデルに対して、それを囲むようにdo whileループで構造変化のbreakの期である変数tbをデータ総数の15%から85%までの区間を動かしています。そのループの外側で、あらかじめ全データ数だけの長さをもつ列ベクトルを作成しておき、任意の小さな数を入れておきます（ここでは-9999）。これにtbのループ内で、ループが１回まわるごとに、第tb行目にτの値を放りこみます。そして、最終的にループをまわし終えてできた列（最初と最後のいくつかの部分には-9999が入っている）のなかでゼロの最も近い一番大きなものを探してきます。また、そのときのインデックスナンバーを得ます。なお、組込み関数maxc()とmaxindc()の使い方はグリッドサーチで行なった要領と同じです。