3.14 プログラミング 系列相関とテスト(2)　　ver.0.1

ここでは、系列相関がある場合に修正して推定する方法をいくつか見ていきます。現在そのほとんどがパッケージソフトに組み込まれていますが、ソフトごと推定結果に若干の違いを見ることを経験された方も多いことでしょう。ここでは、プログラムの方法とあわせて、なぜ同じ推定法でも違いが出てくるのかもあわせて探ります。

Cochrane-Orcutt 2 Step Procedure

この方法は、第１ステップとしてρをなにがしかの方法で求めてから、そのρをもとに、

$y_t^* = y_t - \rho y_{t-1}$　および　$x_t^* = x_t - \rho x_{t-1}$　　t=2,3, ... ,n

という変換をほどこしてから第２ステップとして OLS をさらにもう一度だけ行なうものです。最初の t=1 期のデータを使わないで t=2 期から推定を行なう方法です。

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];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
{bstar,rho,s2hat,se,serho}=corc2(y,x);
print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
print "rho:" rho;


proc (5) = corc2(y,x);
    local n,k,b,e,rho,ystar,xstar,bstar,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,.];
    ystar=y[2:n,.]-rho*y[1:n-1,.];
    xstar=x[2:n,.]-rho*x[1:n-1,.];
    bstar=inv(xstar'xstar)*xstar'ystar;
    estar=ystar-xstar*bstar;
    s2hat=estar'estar/(n-1-k);
    varb=s2hat*inv(xstar'xstar);
    se=sqrt(diag(varb));
    serho=sqrt((1-rho^2)/(n-1-k));
    retp(bstar,rho,s2hat,se,serho);
endp;
```

画面表示

b:	-3.3863782	1.2736474	0.46650862
se:	0.36524576	0.12540902	0.073559077
t:	-9.2715058	10.155947	6.3419586
rho:	0.57053151		

ここでは、ρの計算に通常の最小２乗法を用いて、

$$e_t = \rho\, e_{t-1} + v_t$$

にあてはめて、

$$\rho = \frac{\sum_{t=2}^{n} e_{t-1} e_t}{\sum_{t=2}^{n} e_{t-1}^2}$$

$$\rho = \mathrm{inv}(e_{t-1}\,' e_{t-1})\, e_{t-1}\,' e_t \quad \text{または} \quad \rho = (e_{t-1}\,' e_t)/(e_{t-1}\,' e_{t-1})$$

を OLS residuals から求めます。前章で１期ずらす方法をプログラムしましたが、それを

```
rho=e[1:n-1,.]'e[2:n,.]/e[1:n-1,.]'e[1:n-1,.];
```

というふうにあてはめます。この場合、２期目からの系列を用いています。

一方、これとは違って、ρの Yule-Walker による求め方として、

$$\rho = \frac{\sum_{t=2}^{n} e_t e_{t-1}}{\sum_{t=1}^{n} e_t^2}$$

を計算する方法があります。この分子に相当する部分は最小２乗法の分子とまったく同じですが、分母は t=1 から始まっていますから、１つ分の e^2 が余計に合計されていて、このρは上の最小２乗法のものよりもその分だけ小さくなります。プログラムでは、

```
rho=e[1:n-1,.]'e[2:n,.]/e[1:n,.]'e[1:n,.];
```

というふうに、分子の部分が１から n-1 ではなくて１からｎまでに変更になります。この部分だけを変更した際の結果は、

画面表示

b:	-3.4123789	1.2818990	0.46272031
se:	0.36066104	0.12398109	0.072620977
t:	-9.4614572	10.339472	6.3717170
rho:	0.55892044		

となり、実際にρは最小２乗法計算のρよりも若干小さくなり、それに対する２ステップの場合の係数の値が計算されています。

この他にｎの数が限られている場合の補正の方法として、Durbin-Watson 統計量からρに変換する Theil-Nagar の方法（ｎをデータ数、k を定数項を含む独立変数の数とすると）

$$\rho = \frac{n^2(1-\frac{DW}{2})+k^2}{n^2-k^2}$$

もあります。この場合の変更部分のプログラムは、変数 dw を新たに local 宣言して、

```
dw=(e[2:n,.]-e[1:n-1,.])'(e[2:n,.]-e[1:n-1,.]) / e'e;
rho=(n^2*(1-dw/2)+k^2)/(n^2-k^2);
```

というふうに変更すれば、

画面表示

```
   b:       -3.3636856        1.2664720       0.46978631
  se:        0.36917363       0.12662310      0.074364976
   t:       -9.1113918        10.001903       6.3173060
rho:        0.58031083
```

上の２つの場合よりも、このデータとモデルの場合、ρは若干大きくなり、それにもとづいて係数の値もかわってきます。いずれにせよ、これら２ステップの方法は、始めのρの推定値に１対 1 で依存するもので、ρの計算方法分だけ答えがあることになります。２ステップは、あくまでρの値から決まってくるβの係数を計算しているのであって、おおよその目安として利用できるだけにすぎません。通常、次にあげるρとβを交互に求めていく Iterative な推定方法と著しい乖離が見られます。

<u>Cochrane Orcutt Iterative Procedure</u>

この方法は、２ステップで終わるのではなくて、さらに求められた係数β*をもとにして residuals をｙ－ｘβ*によって求め、そこから再びρを計算し、それに対応するβ*を求めます。これを何度も繰り返して、ρが収束して動かなくなるまでこれらのステップを行なうものです。ここでは、一番最初にプログラムした最小２乗法のρを初期値に使った場合の iterative アルゴリズムを示します。

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];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
{bstar,rho,s2hat,se,serho}=corc(y,x);
print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
print "       rho:" rho;
print "se(rho):" serho;
print "          t:" rho/serho;
```

```
proc (5) = corc(y,x);
        local n,k,rho1,b,e,rho,ystar,xstar,bstar,estar,s2hat,varb,se,serho,i;
        n=rows(x); k=cols(x);
        rho1=0;
        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";
        i=1;
        do while abs(rho-rho1)>10e-8;
                ystar=y[2:n,.]-rho*y[1:n-1,.];
                xstar=x[2:n,.]-rho*x[1:n-1,.];
                bstar=inv(xstar'xstar)*xstar'ystar;
                e=y-x*bstar;
                print/rz i;; print rho;
                rho1=rho;
                rho=e[1:n-1,.]'e[2:n,.]/e[1:n-1,.]'e[1:n-1,.];
                i=i+1;
        endo;
        rho=rho1;
        bstar=inv(xstar'xstar)*xstar'ystar;
        estar=ystar-xstar*bstar;
        s2hat=estar'estar/(n-1-k);
        varb=s2hat*inv(xstar'xstar);
        se=sqrt(diag(varb));
        serho=sqrt((1-rho^2)/(n-1-k));
        retp(bstar,rho,s2hat,se,serho);
endp;
```

画面表示

```
Iteration:      #          rho
                  1          0.57053151
                  2          0.67485668
                  3          0.73519087
                                . . . . . .
```

```
                36          0.84625502
                37          0.84625515
   b:       -2.4733124       1.0312493       0.54825830
  se:       0.49912134       0.15519549      0.10417429
   t:       -4.9553330       6.6448411       5.2628944
      rho:        0.84625515
 se(rho):        0.090055891
        t:        9.3969994
```

ここでも、２ステップのところで説明したように、ρの求め方が違うと結果も若干違ったものになります。実際には、一番最初の最小２乗法によるρの推定だけを使ったアルゴリズムのほかに、ソフトウエアによっては、それとあわせて、一定以下のデータ数の場合には、自由度で補正した Theil-Nagar の方法が用いられることもあります。その都度パラメータは若干の変化をみせることになります。上の３通りのρの求め方のほかに、βの推定自体を、最小２乗法にもとづくのではなくて、残差平方和 RSS の最小値を計算して求めるものもあります。その場合には、上の３通りの方法分だけさらにすこしパラメータがずれることになります。その他、Cochrane-Orcutt では１行目のデータを使わないといっても、２行目の変数を作るときに必ず使うのであって、それを勘違いして最初から１行目をカットしたデータを使っていると、変更後の１行目の変数の中味がおかしい結果がおこり、その状態が全体の推定値に影響を与える場合もあります。この場合は、あきらかに仕様ではなくバグと言えます。またこれとは反対に、一番最初の最小２乗法によるρの求め方で、分子には２期目からかけ合わせた合計を求めるのですが、ミッシングバリューを含んだ計算でρを求めてしまっていることも考えられます。GAUSS のように行列計算で正確さを確かめる作業は重要です。

Prais-Winsten 2 Step Procedure

この方法は、t=1 期目をそこだけ特殊な加工をほどこして含めてしまうやり方です。つまり、t=1 期に対しては、その前に期がないために、最初の期だけをまるめて特殊な加工をして利用しようというものです。プログラム的には Cochrane-Orcutt のものに一部修正を加えるだけでできます。すなわち、

```
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,.];
```

というふうに、いままで欠落していた n=1 期に

$$y_1^* = \sqrt{1-\rho^2}\,y_1 \text{および} X_1^* = \sqrt{1-\rho^2}\,X_1$$

となるデータ一行目の加工したものをもともとの２行目からの Cochrane-Orcutt の変換に垂直方向に｜のマークでマージするだけです。冒頭の procedure を書き換えたものは、下

のようになります。ただし、この場合ｎ行すべてのデータ行列を使いますから、プログラム中 n-1-k というふうに１減らしていた自由度で割るところは、通常の n-k に戻します。

プログラム

```
proc (5) = pw2(y,x);
        local n,k,b,e,rho,ystar,xstar,bstar,estar,s2hat,varb,se,serho,dw;
        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,.];
        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;
```

画面結果

```
   b:       -3.4190752        1.2776664       0.46843295
  se:        0.35715243       0.12401110      0.072778613
   t:       -9.5731541        10.302838        6.4364094
 rho:        0.57053151
```

上の場合は、最小２乗法によるρを使ったものです。Cochrane-Orcutt 法と同様に、ほかの２種類のρの導出方法も使われることもあります。その場合には、係数が１対１で変わってくることになります。計算の方法を見てもわかるように、データの数ｎが小さいときには、１期を含めずに計算するというのはかなりの影響を係数計算に与えてしまいます。しかしながら、この Prais-Winsten の変換をするということは、最初の１期目の値を人工的にρを用いて操作作成することです。したがって、あったであろう０期目のデータを無視して作られているので、信頼性は落ちます。よく、Prais-Winsten が優れているという書き方をされる方がおられますが、いつも真であるとは限りません。ケースバイケースです。１期目のカットなのか、近似的に操作作成するのかという違いなのです。

Prais-Winsten Iterative Procedure

上の１期を操作してまるめた２ステップを Corchrane-Orcutt Iterative Procedure と同じように、Iterative にρが収束するまで計算を続けるのであれば、

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];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
{bstar,rho,s2hat,se,serho}=pw(y,x);
print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
print "       rho:" rho;
print "se(rho):" serho;
print "          t:" rho/serho;


proc (5) = pw(y,x);
    local n,k,rho1,b,e,rho,ystar,xstar,bstar,estar,s2hat,varb,se,serho,i;
    n=rows(x); k=cols(x);
    rho1=0;
    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";
    i=1;
    do while abs(rho-rho1)>10e-8;
        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=y-x*bstar;
        print/rz i;; print rho;
        rho1=rho;
        rho=e[1:n-1,.]'e[2:n,.]/e[1:n-1,.]'e[1:n-1,.];
        i=i+1;
    endo;
    rho=rho1;
    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

```
Iteration:       #          rho
                 1       0.57053151
                 2       0.66943943
                 3       0.72411361
                 . . . . . .
                24       0.80598173
                25       0.80598200
                26       0.80598215
   b:       -2.7264424        1.0671184        0.55837893
  se:       0.45593177        0.15132767       0.095713439
   t:       -5.9799350        7.0517069        5.8338613
      rho:       0.80598215
se(rho):       0.098656640
        t:       8.1695682
```

というふうになります。Iterative な Cochrane-Orcutt の推定値と少しずれてきます。この場合データ数が比較的少ないですから、１期目をどうするかの処理の違いが、このように違いになって出てきます。この場合も、ρの推定を３つのうちのどれにするかで三者三様の結果になります。

<u>OLS と First Difference Procedure</u>

ここで、２ステップの Prais-Winsten のプログラムを見てください。もしρの計算の部分を消去して、ρ＝０としてみます。これは、OLS そのものの計算になります。

```
      rho=0;
```

と２ステップのプログラムを書き換えてやれば、

画面結果

```
   b:       -3.9377145        1.4507860        0.38380813
```

se: 0.23699929 0.083228446 0.048017824
t: -16.614879 17.431372 7.9930345
rho: 0.00000000

というふうに、当然のことながら OLS の結果そのものになります。OLS そのものは、ρが有意でなり場合や、有意であっても比較的小さく０に近い場合には以前として有用な手段となります。他方、その反対の対極のρが１の時はどうでしょうか。この場合、説明変数データＸに定数項の１の列ベクトルが含まれていては singular になっていまいます。そこで、１の列ベクトルをはずしてから冒頭のプログラムのところでやった Cochrane-Orcutt の２ステップの方法でρを１と固定して計算させてみます。これは、データ行列Ｘから定数項の１の列ベクトルを取り除いてから

$y_t^* = y_t - y_{t-1}$ および $X_t^* = X_t - X_{t-1}$ t=2,3, .. , n

という First Difference の操作を変数に加えた推定に等しくなります。

画面表示

b: 0.98687939 0.50197512
se: 0.15758797 0.13399524
t: 6.2624029 3.7462161
rho: 1.0000000

注意すべきは、ここでは自由度は再び１減らした n-1-k になります。そして、もしDW値をあわせて計算しようとすれば、定数項をつけたものを計算しなくてはなりません。そうでなければ本来の Durbin-Watson 値ではなくなってしまいます。Hildreth-Lu におけるρと RSS との関係を調べてみてもわかるように、ρが１に近いところで決まる場合には、RSS の変化は微小なものとなります。したがって、ρが１に近い値では、この First Difference の方法も近似的ではありますが、簡便かつ有効な方法と言えます。

<u>Hildreth-Lu Grid Search Procedure</u>

この方法は、上のρが０の時の OLS とρが１の時の First Difference Procedure とのちょうどその間にρが来るときに、そのρをもとに変換をして推定して得られた残差平方和 RSS を最小にするものを選ぶものです。ちょうど、アルゴリズム的には、Box-Cox 変換のλの推定と同じようなプログラムをすればよいことになります。ここでは、最適値アルゴリズムのところで扱った２次元の Gird Search を簡略化した１次元の Grid Search を行ないます。直線上に並ぶ線の上の各点の高さを計測して、最小値または最大値を選んで、さらにその近傍の直線上のより狭い範囲より狭い範囲へと絞りこんで最小値または最大値を見つけ出すもので、四角のグリッドの中をサーチするものと同様に Grid Search の一種で、通常、１値の場合の Grid Search と呼ばれます。下のプログラムでは、Prais-Winsten 変換を採用したバージョンです。定義によっては、Cochrane-Orcutt の方法で１期目をカウントしない方法もあります。

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];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
{bstar,rho,s2hat,se,serho}=hildlu(y,x);
print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
print "       rho:" rho;
print "se(rho):" serho;
print "          t:" rho/serho;


proc (5) = hildlu(y,x);
    local n,k,b,e,rho,iter,length,start,finish,rhomax1,step,grid,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

```
Iteration:       #          rho
                 1      0.57053151
                 2      0.80000000
                 3      0.82222222
                 4      0.82592593
                 5      0.82551440
                 6      0.82554489
                 7      0.82554658
                 8      0.82554714
                 9      0.82554725
                10      0.82554723
  b:       -2.6579799       1.0502132       0.56290620
 se:       0.46495966       0.15287030      0.098184816
  t:       -5.7165817       6.8699622       5.7331289
```

rho: 0.82554723
se(rho): 0.094055505
t: 8.7772346

アルゴリズム自体は、２値の場合の Grid Search を１重にしたものです。そのループの間に、その都度 Prais-Winsten 変換で residuals を求め、その平方和を線分 Gird の上に並べて、その中で一番小さいものを選択します。もともとのステップの＋－ステップ分の範囲の線分グリッドの区間についてこの作業を繰り返します。ρの値が動かなくなったところでこの繰り返しの作業を打ち切って、ループから break で抜け出します。ここでは、1e-8 を採用しています。なお、grid の最初のスタート値は-0.9、フィニッシュ値は 0.9 に設定して、ステップ幅はその間を 18 で割った 0.1 に初期設定しています。これにより、－1 から＋１までの値をカバーしています。スタート値とフィニッシュ値、それにステップ幅が、さらに絞り込むときに小さくなっていきます。プログラムの呼出し部と、procedure 後半のβとそれに関する統計量の計算部分はこれまでのプログラムと同様です。なお、残差計算に Cochrane-Orcutt 法を用いると推定結果は若干違ったものになります。

ML Grid Search Procedure

Hildreth-Lu Grid Search の場合は、与えられたρに対して RSS（すなわち、プログラムでは e'e）を最小にするものを選択しましたが、この ML Gird では、与えられたρにたいして Log-Likelihood 関数を最大にする（下のプログラムではマイナス Log-Likelihood 関数を最小にする）ものを選択します。この場合、Log-Likelihhod を最大にするβとρを一括して求めるのではなくて、ρの関数であるβ(ρ)を間接的に求めることになります。プログラム的には、Hildreth-Lu の e'e の最小化のところを、これは、AR(1)にしたがう残差ｅが

$e_t = \rho e_{t-1} + \varepsilon_t, \quad |\rho| < 1, \quad \varepsilon t \sim NID(0, \sigma^2_\varepsilon)$ のときの Log-Likelihood の

$$LL = \frac{1}{2}\log(1-\rho^2) - \frac{n}{2}\log 2\pi\sigma_\varepsilon^2 - \frac{1}{2\sigma_\varepsilon^2}(y^* - \beta^* X^*)'(y^* - \beta^* X^*)$$

を最大化するように１行だけ変更するだけで足ります。ただし、アルゴリズムは最小値を求めるものですので、上の Log-Likelihood の全体にマイナスをつけたものを最小化します。

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];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
{bstar,rho,s2hat,se,serho}=mlgrid(y,x);
```

```
        print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
        print "       rho:" rho;
        print "se(rho):" serho;
        print "          t:" rho/serho;

proc (5) = mlgrid(y,x);
    local n,k,b,e,rho,iter,length,start,finish,rhomax1,step,grid,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]= -1/2*ln(1-rho^2)-n/2*(1+2*pi-ln(n))+n/2*ln(e'e); */
            grid[j]= -1/2*ln(1-rho^2)+n/2*ln(2*pi*stdc(e)^2)+e'e/(2*stdc(e)^2);
            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

```
Iteration:       #          rho
                 1       0.57053151
                 2       0.78000000
                 3       0.77822222
                 4       0.77840000
                 5       0.77839627
                 6       0.77839631
   b:       -2.8217134        1.0925255        0.54997898
 se:        0.44363399       0.14884070       0.092490261
   t:       -6.3604537        7.3402340        5.9463448
      rho:       0.77839631
se(rho):       0.10462887
        t:        7.4395942
```

Log-Likelihood 関数は、normalize したものでもしないものでも、どちらでもよいのですが、要するに Hildreth-Lu の場合には e'e の RSS を最小にしたのに対して、ここでは、ρに依存しない定数項とみなせるところと 1/2ln(1-ρ^2)の項が付け加わることになります。（なお、ここで－n/2ln(e'e)のところを最大にするのも e'e を最小にするのも基本的に同じことになります。）実際の推定結果は、1/2ln(1-ρ^2)の項の分だけ、この場合ρの値が下方に修正されています。ここで用いられる Log-Likelihood 関数は ARMA(1,1)のときと共通のものです。おそらく、もしデータが純粋に MA(1)の項を含まないのものであれば、結果的に、

この ML Gird と前の Hildreth-Lu のρの値は同じになるものと思われます。なお、1/2ln(1-ρ^2)を含まない Log-Likelihood の部分が MA(1)だけのときの Log-Likelihood に相当します。実際には、このデータは MA(1)の部分を若干含んでいて、ARMA(1,1)にふるまうので、ここではその分だけρが Hildreth-Lu よりも下方に振れているのだと思われます。

Beach-MacKinnon ML procedure

ここでは残差とρに関する３次式の係数との関係から、間接的に Log-Likelihhod 関数を最大にするρとそれに対する係数βｓを計算します。最後のところで示している Constained の Maximum Log-Likelihood を求める方法がソフトウエア内で技術的に不可能な場合には有効な方法です。これは、AR(1)にしたがう residuals のｅが

$e_t = \rho e_{t-1} + \varepsilon_t, \quad |\rho| < 1, \quad \varepsilon t \sim NID(0, \sigma^2_\varepsilon)$

のときの Log-Likelihood

$$LL = \frac{1}{2}\log(1-\rho^2) - \frac{n}{2}\log 2\pi\sigma_\varepsilon^2 - \frac{1}{2\sigma_\varepsilon^2}(y^* - \beta^* X^*)'(y^* - \beta^* X^*)$$

を最大化するρとβ*を求めるために、間接的に、2 Step Prais-Winsten のβの推定値をもとにして、さらに t=1,2, … ,n の全データに対して

$e_t = y_t - \beta X_t$

を求めて、

$$P = e_1^2 + e_n^2$$

$$Q = \sum_{t=2}^{2} e_t e_{t-1}$$

$$R = \sum_{t=2}^{n-1} e_t^2$$

を計算の上、ρ関する３次式の係数

$$f = -\frac{(n-2)Q}{(n-1)R}, g = -\frac{(n+1)R + P}{(n-1)R}, h = \frac{nQ}{(n-1)R}$$

に対して、$\rho^3 + f\rho^2 + g\rho + h = 0$ を解いて、解のうち$|\rho| < 1$を満たす１つの解をρとして収束するまで 2 step Prais-Winsten からの手順を繰り返します。下のプログラムでは、３次式のρの値を直接解かずに、３次式の左辺の絶対値が－１と１の間で最小値になるところを１値の Grid Search で探し出して、新旧のρ（1 期前のρは rho0、当期のρは zmax とします）が abs(zmax-rho0)/rho0<1e-5;の条件を満たすまで上の手順を繰り返し、ρが条件を満たせば break でρの計算(zmax の計算)を抜け出して、いままでどおりβとそれに対する統計量を求めています。この他に、ρを計算機的に正確に求める方法もあります。

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];
    y=ln(Y); x=ones(39,1)~ln(L)~ln(K);
    {bstar,rho,s2hat,se,serho}=beach(y,x);
    print "    b:" bstar'; print " se:" se'; print "    t:" bstar'./se';
    print "       rho:" rho;
    print "se(rho):" serho;
    print "          t:" rho/serho;

proc (5) = beach(y,x);
    local n,k,b,e,rho0,l,ystar,xstar,bstar,p,q,r,f,g,h,iter,length,start,finish,i,step,grid,
z,j,zmax,estar,s2hat,varb,se,serho;
    n=rows(x); k=cols(x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    rho0=e[1:n-1,.]'e[2:n,.]/e[1:n-1,.]'e[1:n-1,.];
    print "Initial rho:" rho0;
    l=1;
    do while l<=100;
        ystar=sqrt(1-rho0^2)*y[1,.]|y[2:n,.]-rho0*y[1:n-1,.];
        xstar=sqrt(1-rho0^2)*x[1,.]|x[2:n,.]-rho0*x[1:n-1,.];
        bstar=inv(xstar'xstar)*xstar'ystar;
        e=y-x*bstar;
        p=e[1]^2+e[n]^2;
        q=e[2:n]'e[1:n-1];
        r=e[2:n-1]'e[2:n-1];
        f=-(n-2)*q/((n-1)*r);
        g=-((n+1)*r+p)/((n-1)*r);
        h=n*q/((n-1)*r);

        iter=100; length=18;
        start=-0.9; finish=0.9;
        i=1;
        do while i<=iter;
            step=(finish-start)/length;
            grid=(-1e256).*ones(length+1,1);
```

```
            z=start;
            j=1;
            do while j<=length+1;
                grid[j]=abs(z^3+f*z^2+g*z+h);
                j=j+1;
                z=z+step;
            endo;
            zmax=start+(minindc(grid)-1)*step;
            if abs(zmax^3+f*zmax^2+g*zmax+h)<1e-10;
                break;
            endif;
            start=zmax-step; finish=zmax+step;
            print/rz i;;print zmax;
            i=i+1;
        endo;

        if abs(zmax-rho0)/rho0<1e-5;
            break;
        endif;
            rho0=zmax;
        l=l+1;
    endo;

    rho=zmax;
    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

```
Initial rho:        0.57053151
                    1         0.70000000
                    2         0.65555556
                    3         0.65679012
                    . . . . . .
                    6         0.77830615
                    7         0.77830559
                    8         0.77830567
                    1         0.80000000
                    2         0.77777778
                    3         0.77777778
                    4         0.77832647
                    5         0.77831123
                    6         0.77831123
                    7         0.77831067
                    8         0.77831065
                    9         0.77831065
rho=        0.77831066
   b:       -2.8220063          1.0926065         0.54994996
 se:        0.44359633          0.14883245        0.092480635
   t:       -6.3616539          7.3411849         5.9466499
      rho:        0.77831066
se(rho):        0.10464656
        t:         7.4375175
```

(ML Procedure（参考：CML ライブラリ＋正式版 GAUSS が必要）

通常のソフトウエアでは計算不可能ですが、Constrained Maximum Likelihood を簡単に求められる GAUSS の CML ライブラリでは、Direct ML を計算できます。ρ に絶対値１よりも小さいという制約があるために、通常のアルゴリズムでは Log-Likelihood 関数を最大化することはできません。そこで、Beach-MacKinnon の「婉曲的な」ML 法が考案されたのでしょう。CML を得意中の得意とする GAUSS の世界では、婉曲的にではなくて、直接正確に推定可能です。以下が Beach-MacKinnon のときと同じ Log-Likelihhood 関数を最大化して、それに対するρをβを求めるプログラムです。なお、このプログラムでは分散を制限なしでは収束しないので、そのための対処方法として、これまでの Likelihood の計算にもとづく計算方法と同じく 2 Step Prais-Winten の residuals の standard deviation の２乗を分散にして、連立式の一部に組みこんでいます。その他の詳しい Log-Likelihood

の作り方およびプログラムの組み方については、ライブラリの節の CML の章を参照してください。

プログラム

```
new; cls;
library cml; cmlset;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
y=ln(Y); x=ones(39,1)~ln(L)~ln(K); data=y~x;
start={1,1,1,0};
_cml_Bounds={-1e256 1e256,
                -1e256 1e256,
                -1e256 1e256,
                -0.999   0.999};
_cml_Algorithm = 1;
_cml_ParNames = {"CONST","X1","X2","rho"};
call cmlprt(cml(data,0,&ll,start));

proc ll(b,data);
    local n,k,x,y,ystar,xstar,e,llik;
    n=rows(data); k=cols(data);
    y=data[.,1]; x=data[.,2:k];
    ystar=sqrt(1-b[k]^2)*y[1,.]|y[2:n,.]-b[k]*y[1:n-1,.];
    xstar=sqrt(1-b[k]^2)*x[1,.]|x[2:n,.]-b[k]*x[1:n-1,.];
    e=ystar-xstar*b[1:k-1];
    llik=1/2*ln(1-b[k]^2)-n/2*ln(2*pi*stdc(e)^2)-e'e/(2*stdc(e)^2);
    retp(llik);
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood          2.21619
Number of cases      39

Covariance of the parameters computed by the following method:
```

Inverse of computed Hessian

Parameters	Estimates	Std. err.	Gradient
CONST	-2.8217	0.6246	-0.0000
X1	1.0925	0.1899	-0.0001
X2	0.5500	0.0990	-0.0000
rho	0.7784	0.1397	-0.0000

Number of iterations 162
Minutes to convergence 0.31867

簡単に説明すると、係数ｂの１番目から３番目を制限なしに、４番目（ここではρに相当）を-0.999 から 0.999 の範囲に制限した上で、後半の Log-Likelihood の procedure を CML ライブラリで最大化させています。Log-Likelihood にはスケラーの値そのものを用いています。ここでは定数項を含む３つの係数とρの計４つを同時に動かして Log-Likelihood を最大化させています。

このように、ρの求め方や残差自体の処理の仕方、どういった Log-Likelihood を実際に使うのかによって、その組み合わせの数だけρと係数の推定値は若干の違いを見せるわけです。

```
/*
** These programs above were independently written by Yosuke Amijima
** (C) Copyright 2002 Yosuke Amijima. All Rights Reserved.
**
** The results of other software may slightly differ in some points.
*/
```