3.18 プログラミング 共和分とテスト　　ver.0.1

まず、OLS回帰の残差のADFの考え方を用いた、１個だけの共和分を対象にしたテストからはじめて、さらに複数個の共分和を対象にしたよく知られているテストのプログラムをいくつか試みます。

Engle-Granger Cointegration Test

ここでは、あらかじめ従属変数と独立変数がわかっているときに、

$Y=X\beta+\varepsilon$

のregressionをまず行ない、そのN×１のディメンションの残差εの系列についてADFまたはDFのテストをするものです。なお、通常は最初のregressionのＸの中に定数項を表す１の列ベクトルを含みます。トレンドを考える場合には、さらにＸの中に1,2,3,...Tのトレンド項を表す列ベクトルを含むことになります。

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];
y=Y; x=ones(rows(L),1)~L~K;
call egcoint(y,x);
print;
print "================================";
print "With trend";
call egcoint(y,seqa(1,1,rows(x))~x);
```

```
proc(0)=egcoint(y,x);
    local b,e;
    print "================================";
    print "Engle-Granger Cointegration Test";
    print "================================";
    b=inv(x'x)*x'y;
    e=y-x*b;
    call adf(e);
endp;
```

```
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;
        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;


proc(0)=adftest(y,lag);
    local n,dy,d2y,d3y,i,j,x;
```

```
    print "ADF Test(No Drift):";
    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);
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

```
================================
Engle-Granger Cointegration Test
================================
 8          tau=     -0.37400957 ( n=          30 )
 7          tau=      0.61824300 ( n=          31 )
 6          tau=     0.011153917 ( n=          32 )
 5          tau=     -0.11329542 ( n=          33 )
 4          tau=      0.97974402 ( n=          34 )
 3          tau=     0.010719605 ( n=          35 )
 2          tau=      -2.1223865 ( n=          36 )
```

Choice of Lag # = 2

==========

LAG= 0

==========

ADF Test(No Drift):

tau= -2.6557750 (n= 38)

==========

LAG= 1

==========

ADF Test(No Drift):

tau= -4.6301542 (n= 37)

==========

LAG= 2

==========

ADF Test(No Drift):

tau= -2.4105943 (n= 36)

==========

With trend

==========

Eagle-Granger Cointegration Test

==========

8	tau=	-0.67197756 (n=	30)
7	tau=	0.76642428 (n=	31)
6	tau=	-0.26932338 (n=	32)
5	tau=	-0.0017793676 (n=	33)
4	tau=	1.1127035 (n=	34)
3	tau=	0.028355379 (n=	35)
2	tau=	-1.8022076 (n=	36)

Choice of Lag # = 2

```
===========
LAG= 0
===========
ADF Test(No Drift):
tau=        -2.4794821 ( n=                  38 )

===========
LAG= 1
===========
ADF Test(No Drift):
tau=        -4.2036277 ( n=                  37 )

===========
LAG= 2
===========
ADF Test(No Drift):
tau=        -2.3346762 ( n=                  36 )
```

プログラム上は、ADFの３階建てのプログラムのうち、２階部分を通常の系列のDriftなし（定数項なし）の部分だけを使って書き改めて、さらにその上に４階部分としてegcointという１従属変数と複数の独立変数をインプットとして代入した時に残差を計算してADFのプログラムを呼び出すprocを加えています。冒頭の呼出し部分は、そのegcointを呼び出していますが、通常の定数項つきの独立変数の呼び出しと、さらにトレンド項を含んだものの呼び出しの２つをコメント表示をはさんで置いています。繰り返しますが、このバージョンでは、残差を計算する前に定数項（トレンド項のあるものはトレンド項）を加えて、ADFの計算の中では定数項なしのもの（必要ならばADFの内部に定数項を加えてください）を常に用いています。ただし、通常は外で定数項をつけたりトレンド項をつけたりすると内のADF計算ではそれらの項はつけません。またその逆というケースも多々見られます。なお、ラグを考えないDFの計算をするには、２階部分のADFの計算のラグのところに、強制的に０を代入すればDFのラグ数０のDFの計算になります。当然のことながら、ADFと同じような単位根テストのほとんどすべてのテストがここでは利用できます。KPSSや構造変化をともなったテストなどをそのままあてはめることができるでしょう。ただし、このOLS回帰残差を用いる方法はどれも、すべての変数がI(1)であり、かつ共和分が１個だけ存在する時にのみ有効です。また、独立変数から従属変数に対しての因果性が認められるときにのみ有効です。複数個の共和分の存在や変数がI(2)のものを含んだ場合についてはテストできませんし、変数間に因果性がはっきりとしない場合にはOLS残差を求めること自体

が不可能です。

Error Correction Model

ここでは最も簡単なECMの例をprocedureで示しておきます。いろいろなパターンがありますが、すべての変数がI(1)であって、それらの間に共和分の関係が１個だけ存在するケースを考えます。すなわち、１個の共和分が存在するとして、

$$\varepsilon_t = Y_t - \alpha - \beta_1 L_t - \beta_2 K_t$$

をいう誤差項をEngle-Grangerの方法と同じようにして計算してから、そのラグ項を含む

$$\Delta Y_t = \delta_1 \Delta L_t + \delta_2 \Delta K_t - \gamma \varepsilon_{t-1} + u_t \quad \text{ここで} \gamma > 0$$

を１階の階差の関係を推定するものです。まずは、プロトタイプとして有用でしょう。

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=Y; x=L~K;
call ecm(y,x);

proc ecm(y,x);
local n,k,b,e,dy,dx,s2hat,varb,se,t,g,corr,r2;
x=ones(rows(x),1)~x;
n=rows(x); k=cols(x);
b=inv(x'x)*x'y;
print "Suspected Cointegrating vector:";
print 1~(-b[2:rows(b)]');
e=y-x*b;
dy=y[2:n]-y[1:n-1]; dx=(x[2:n,2:k]-x[1:n-1,2:k]);
x=dx~(-e[1:n-1]);
b=inv(x'x)*x'dy;
e=dy-x*b;
s2hat=e'e/((n-1)-k);
varb=s2hat*inv(x'x);
se=sqrt(diag(varb));
t=b./se;g=b[k];
print "Error Correction Model:";
print "                    b                    se                    t";;
print b~se~t;
```

```
    print/lz " n=" n-1;
    print /lz "df=" n-1-k;
    corr=corrx(dy~(x*b));
    r2=corr[1,2]^2;
    print/lz "R2=" r2;
    print/lz "gamma=" g;; print/lz "1/gamma=" 1/g;
    retp(g);
endp;
```

画面表示

```
Suspected Cointegrating vector:
        1.0000000        -1.3069296        -1.6223203
Error Correction Model:
                 b                se                 t
        1.1090867        0.25793248         4.2999108
        1.5603529        0.29394694         5.3082810
       0.25161730        0.14536048         1.7309884
 n= 38
df= 35
R2= 0.79594174
gamma= 0.2516173          1/gamma= 3.9742895
```

上の計算で、この場合のR2はＹ(この場合dＹ)とその推定値の相関係数の２乗です。調整項の係数γの逆数は、調整完了の期間を表します。このECMが成立しているとすれば、この場合、ほぼ４期で調整がなされることを示しています。

CRDW（Cointegration Regression Durbin Watson）

この考え方は、はっきりとした統計量としては疑問が残るものの、相対的に小さなR2と大きなDW統計量の組み合わせのときに共和分が疑われるという経験上の直感を数値関係に直したものであり、EGテストを補完する上でも有用です。ただし、ここで使われるDW統計量は、その分母のところが平均値からの距離を取っているので、厳密にはDWとは違います。すなわち、

$$CRDW = \frac{\sum_{t=2}^{n}(u_t - u_{t-1})^2}{\sum_{t=1}^{n}(u_t - \bar{u})^2} < R^2$$

という関係になるかをテストします。これとは別に、CRDWの値そのものを統計量にしてデータ数100に対して、0.511(1%)、0.386(5%)または0.322（10%）のカットオフポイントと比較する方法もあります。

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=Y; x=ones(rows(L),1)~L~K;
    call crdw(y,x);

proc crdw(y,x);
  local n,b,e,dw,r2;
  n=rows(x);
  b=inv(x'x)*x'y;
  e=y-x*b;
  dw=sumc((e[2:n,.]-e[1:n-1,.])^2)/sumc((e-meanc(e))^2);
  r2=1-e'e/(y'(eye(n)-1/n*ones(n,1)*ones(n,1)')*y);
  if dw<r2;
     print "CRDW < R2: No cointegration.";
  else;
     print "CRDW >= R2: Cannot determine no cointegration";
  endif;
  print/lo "CRDW=" dw "(R2=" r2 ")";;print/lz "n=" n;
  print "[Cutoff points: 0.511(1%),0.386(5%),0.322(10%) for n=100]";
  retp(dw);
endp;
```

画面表示

```
CRDW < R2: No cointegration.
CRDW= 0.63482362        (R2= 0.99529249        )n= 39
[Cutoff points: 0.511(1%),0.386(5%),0.322(10%) for n=100]
```

上のプログラムでは、EG テストと同じように、定数項つきの因果性をもった OLS 回帰で残差を求め、その残差にもとづいて CRDW と R2 の２つの値を求め、その大小関係によって条件分岐で結果を示しています。きわめて手軽にでき、「一応の目安」となります。

Johansen Cointegration Test

さて、ここからはOLS残差から離れて、従属変数と独立変数に区別をつけずに分析する方法に進みます。ここでは、変数がm個ある時、m－1個まで共和分は存在し得ることを考えて、複数個の共和分を前提にした、因果性を考えないテストを行ないます。今仮にすべての系列を水平方向にマージした系列の行列をZtとします。そこで、

$$\Delta Zt = P1\Delta Zt\text{-}1 + \ldots + Pk\text{-}1\Delta Zt\text{-}(k\text{-}1) + R_{0t}$$

$$Zt\text{-}k = Q1\Delta Zt\text{-}1 + \ldots + Qk\text{-}1\Delta Zt\text{-}(k\text{-}1) + R_{kt}$$

という差分項とラグ項を従属変数（行列）にしたそれぞれの補助回帰を行なって、それぞれの残差をR0tおよびRktとします。それら２つから、残差のプロダクトモーメント行列

$$Sij = T^{-1}R_{it}'R_{jt} \qquad i,j=0\text{または}k$$

を求めて、

$$| \lambda S_{kk} - S_{0k}S_{00}^{-1}S_{0k} | = 0$$

すなわち、

$chol(S_{11})^{-1}S_{0k}S_{00}^{-1}S_{0k}\ chol(S_{11})^{-1}$ または $S_{11}^{-1}S_{0k}S_{00}^{-1}S_{0k}$のeigen valueを変数の個数だけ求めます。これを大きいものから降順に並べなおして、Likelihood Ratio Testの最終項に２をかけた数であるTrace Statisticの

$$\lambda_{trace} = -T\Sigma\ln(1-\lambda_i)$$

および

$$\lambda_{max} = -T\ln(1-\lambda_{r+1})$$

を求めてテスト統計量とします。通常はどちらも変数の個数だけあって、それぞれの値は大きい方からCointegrating Rankの0,1,..,k-1に相当します。

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];
z=Y~L~K;
call johansen(z,3);

proc(2)=johansen(z,lag);
    local n,k,t,dz,x,i,j,dy,lagy,v,u,s00,s11,s01,s10,invc,lambda,lmax,tracestat;
    n=rows(z); k=cols(z); t=n-lag;
    dz=z[2:n,.]-z[1:n-1,.];
    x=dz;
    i=1; j=1+lag; x=x[j-1:n-1,.];
```

```
    do while i<=lag-1;
        x=x~dz[j-i-1:n-1-i,.];
        i=i+1;
    endo;
    dy=x[.,1:k];
    lagy=z[1:n-lag,.];
    if lag==1;
        x=ones(t,1);
    else;
        x=ones(t,1)~x[.,k+1:lag*k]; /* You could change here to x= x[.,k+1:lag*k]; */
    endif;
    u=dy-x*inv(x'x)*x'dy;
    v=lagy-x*inv(x'x)*x'lagy;
    s00=u'u/t;
    s11=v'v/t;
    s01=u'v/t;
    s10=v'u/t;
    invc=inv(chol(s11)');
    lambda=eig(invc*s10*invpd(s00)*s01*invc');
    /* Or     lambda=eig(inv(s11)*s10*inv(s00)*s01); */
    lmax=rev(sortc(-t*ln(1-lambda),1));
    tracestat=rev(cumsumc(rev(lmax)));
    print "=======================================================";
    print "   Coint Rank              Lmax                  Trace Stat";
    print "=======================================================";
    __fmtnv={"*.*lg " 8 8};
    call printfmt(seqa(0,1,rows(lmax))~lmax~tracestat,1);
    print "=======================================================";
    retp(lmax,tracestat);
endp;
```

画面表示

```
=====================================================
  Coint Rank              Lmax                  Trace Stat
=====================================================
         0              13.516105              21.657961
```

1	6.6088271	8.1418558
2	1.5330288	1.5330288

==========

上のプログラムでは、補助回帰のところに定数項をつけたバージョンです。なお、ラグが１の場合には定数項だけになるように場合分けされています。（定数項なしのプログラムが必要の場合には、適宜、定数項を削除して計算してください。）

Stock-Watson Common Trends Test

Johansen と同様に、従属変数独立変数の区別のない複数の系列について、ラグを考慮に入れた $\overline{\Phi} = (\xi_{t-1}'\xi_{t-1})^{-1}\xi_{t-1}\xi_t$

を計算した上で、Φの降順での eigenvalue の列ベクトルをλとすると、ｋ個の変数に対するｍ個の Common Trends の存在のテストには、次のような T と[real(λ_{m+1})-1]をかけ合わせた統計量

$q_f(k, m) = T[real(\lambda_{m+1})-1]$

を考えます。ここで、λ_{m+1}はλの(m＋1)番目の要素です。計算するにあたって、そのままの系列、定数項を入れて Demaen したもの、そして定数項とトレンドを考えて除去したものの３つについての推定量Φを計算します。Stock-Watson(1988)の Table1 から 3 のλ_*および$\lambda_*{}^{\mu}$および$\lambda_*{}^{\tau}$のそれぞれの場合に結果は対応しています。

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];
data=Y~L~K;
m=1;        /* # of common trends under Ha */
p=1;        /* # of lag considered    */
call sw(data,p,m);


proc(0)=sw (data,p,m);
    local d,n,xt,dbeta;
    print "Cointegration Test for Stock-Watson Common Trends";
    print "**lambda**";
    d=data;
    call swmain(d,p,m);
```

```
    print "**lambda myu**";
    n=rows(data);
    xt=ones(n,1);
    dbeta=inv(xt'xt)*xt'data;
    d=data-xt*dbeta;
    call swmain(d,p,m);
    print "**lambda tau**";
    xt=ones(n,1)~seqa(1,1,n);
    dbeta=inv(xt'xt)*xt'data;
    d=data-xt*dbeta;
    call swmain(d,p,m);
endp;

proc swmain(d,p,m);
    local n,k,r,v,mat,alphat,delta,wt,dwt,x,i,phi,dy,xi,xi_1,phibar,lambda,qf;
    n=rows(d); k=cols(d);
    {r,v}=eigrs2(d'd);
    mat=r~v'; mat=rev(sortc(mat,1));
    r=mat[.,1]; v=mat[.,2:cols(mat)]';
    delta=sqrt(k)*v;
    alphat=delta[.,1:k];
    wt=d*alphat/k;
    dwt=wt[2:n,.]-wt[1:n-1,.];
    x=ones(n-1-p,1);
    i=1;
    do while i<=p;
        x=x~dwt[p+1-i:n-1-i,.];
        i=i+1;
    endo;
    dy=dwt[p+1:n-1,.];
    phi=inv(x'x)*x'dy;
    xi=wt*phi';
    xi_1=xi[1:rows(xi)-1,.];
    xi=xi[2:rows(xi),.];
    phibar=inv(xi_1'xi_1)*xi_1'xi;
    lambda=real(eig(phibar));
```

```
    lambda=rev(sortc(lambda,1));
    qf=rows(xi)*(lambda[m+1]-1);
    print "qf( k=";;print/rz k;;print ", m=";;print/rz m;;print ") = ";;print qf;
    retp(qf);
endp;
```

画面表示

```
Cointegration Test for Stock-Watson Common Trends
**lambda**
qf( k=          3 , m=          1 ) =        1.8237457
**lambda myu**
qf( k=          3 , m=          1 ) =       -2.9410820
**lambda tau**
qf( k=          3 , m=          1 ) =       -19.169483
```

Phillips-Ouliaris Test

Demean したデータ系列または、トレンドと Demean を考えた系列を作成してｚとして

$z_t = \Pi z_{t-1} + e_t$について

統計量Pz = Ttr($\hat{\Omega} M_{zz}^{-1}$)を求める

ここで$\hat{\Omega} = \frac{1}{T}\sum_{t=1}^{T}\hat{e}_t^2 + 2\frac{1}{T}\sum_{i=1}^{q}\sum_{t=i+1}^{T} w_i \hat{e}_t \hat{e}_{t-i}$　　$M_{zz} = \frac{1}{T}\sum_{t=1}^{T} z_t^2$

Ωのところは、Phillips-Perron の Unit Root のテストのコアとなる部分と同じである。ラグ数ｑまでの計算を考えている。Mzz の計算では、e の系列がすでに１個減っているので、もともとのｚの系列の先頭の列を取り除いたものを用いることにする。ここでは、この e の列の個数 T に、Ωと Mzz の逆行列の積のトレース（対角成分の和）結果をかけ合わせたものを判断すべき統計量としています。

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];
z=Y~L~K;
nlag=3;
call po(z,nlag);
```

```
proc(0)=po(z,nlag);
    local n,xt,dbeta;
    print "Phillips-Ouliaris Cointegration Test";
    print "Raw:";
    z=z;
    call pomain(z,nlag);
    print "Demean";
    n=rows(z);
    xt=ones(n,1);
    dbeta=inv(xt'xt)*xt'z;
    z=z-xt*dbeta;
    call pomain(z,nlag);
    print "Demean+Detrend";
    xt=ones(n,1)~seqa(1,1,n);
    dbeta=inv(xt'xt)*xt'z;
    z=z-xt*dbeta;
    call pomain(z,nlag);
endp;

proc pomain(z,nlag);
    local n,dy,x,n1,k,b,e,s,i,w,pz,omegahat,t,mzz;
    n=rows(z);
    dy=z[2:n,.]-z[1:n-1,.]; x=z[1:n-1,.];
    n1=rows(x);
    b=inv(x'x)*x'dy;
    e=dy-x*b;
    i=1; s=0;
    do while i<=nlag;
        w=1-i/(nlag+1);
        s=s+2*w*e[1:n1-i,.]'e[i+1:n1,.]/n1;
        i=i+1;
    endo;
    omegahat=e'e/n1+s;
    t=n1; z=z[2:n,.];
    mzz=z'z/t;
    pz=t*sumc(diag(omegahat*inv(mzz)));
```

```
    print "Pz Stat=" pz;
    retp(pz);
endp;
```

画面結果

```
Phillips-Ouliaris Cointegration Test
Raw:
Pz Stat=          8.3251747
Demean
Pz Stat=          22.836721
Demean+Detrend
Pz Stat=          40.898654
```

プログラム的には、Stock-Watson の時と同じように、そのままの系列、定数項を加えて Demean したもの、さらに Detrend を考えたものをｚとして、その複数個の系列を一括して、Phillips-Perron の Unit Root のテストと同じアルゴリズムを使ってラグを考えたΩの推定量を求めます。これと Mzz の逆行列の計算結果をかけ合わせたもののトレースを

```
sumc(diag(omegahat*inv(mzz)))
```

というふうに、対角要素を求め列で返す diag の結果を、さらに sumc でその合計をとってトレースにしています。この値に、T をかけて統計量になります。ラグ数はここでも自動計算にしていませんが、ラグ数の計算は後の章で VAR の枠組みの中で扱うことにします。

ここで、系列相関から始まって、単位根のテストおよび共和分のテストなどで重点的に用いられた、変数のラグをとったり差分をとったりする事項をまとめておきます。本書ではコマンドを極力用いずに普遍的なプログラムを書くという立場から、変数のディメンションを操作する手法を用いています。以下に示すように、通常よく見られるものとしては、簡易なコマンド、lag1 や lagn、それに trimr といったものを使って、１期のラグをとる、複数期のラグをとる、そして上から何行目までを trim する。下から何行目までを trim するといったコマンドを用います。また、ラグで生じた欠損値のドットを消すために trimr ではなくて、packr というドットを含んだ行を消去するコマンドも用いることもよく見かけます。

lag1(x)	Ｘを１期ずらしたラグをとる
lagn(x,i)	Ｘをｉ期ずらしたラグをとる
trimr(x,i,j)	Ｘの上かｉ行分、下からｊ行分を行単位で削除する
packr(x)	Ｘのうちドットを含んだ行を行単位で削除する

ポイント

１期異なる２つの系列

```
=====================================================================
        コマンドに依存しない方法                  コマンドに依存した方法
=====================================================================
             x=seqa(1,1,10);                          x=seqa(1,1,10);
             n=rows(x);
             xt=x[2:n,.];                             xt=trimr(x,1,0);
             xt_1=x[1:n-1,.];                         xt_1=trimr(x,0,1);
             print xt~xt_1;                           print xt~xt_1;
=====================================================================
   xt   xt_1
1        1
2   2    2      2.0000000         1.0000000
3   3    3      3.0000000         2.0000000
4   4    4      4.0000000         3.0000000
5   5    5      5.0000000         4.0000000
6   6    6      6.0000000         5.0000000
7   7    7      7.0000000         6.0000000
8   8    8      8.0000000         7.0000000
9   9    9      9.0000000         8.0000000
10 10             10.000000         9.0000000
```

複数期異なる２つの系列

```
=====================================================================
        コマンドに依存しない方法                  コマンドに依存した方法
=====================================================================
             x=seqa(1,1,10);                          x=seqa(1,1,10);
             n=rows(x); i=3;                                 i=3;
             xt=x[1+i:n,.];                           xt=trimr(x,i,0);
             xt_i=x[1:n-i,.];                         xt_i=trimr(x,0,i);
             print xt~xt_i;                            print xt~xt_i;
=====================================================================
   xt   xt_3
1        1
```

```
2        2
3        3
4   4    4       4.0000000          1.0000000
5   5    5       5.0000000          2.0000000
6   6    6       6.0000000          3.0000000
7   7    7       7.0000000          4.0000000
8   8            8.0000000          5.0000000
9   9            9.0000000          6.0000000
10  10           10.000000          7.0000000
```

1 階の差分

コマンドに依存しない方法	コマンドに依存した方法

x=seqa(1,1,10);
n=rows(x);
dx=x[2:n,.]-x[1:n-1,.];
 	
print dx;
	x=seqa(1,1,10);
	
	dx=packr(x-lag1(x));
	(または dy=trimr(x-lag1(x),1,0);)
	print x-lag1(x); print dx;

```
                ドット消去前                消去後
1   1-.=                .
2   2-1=        1.0000000           1.0000000
3   3-2=        1.0000000           1.0000000
4   4-3=        1.0000000           1.0000000
5   5-4=        1.0000000           1.0000000
6   6-5=        1.0000000           1.0000000
7   7-6=        1.0000000           1.0000000
8   8-7=        1.0000000           1.0000000
9   9-8=        1.0000000           1.0000000
10  10-9=       1.0000000           1.0000000
```

複数階の差分

コマンドに依存しない方法	コマンドに依存した方法

```
x=seqa(1,1,10);                          x=seqa(1,1,10);
n=rows(x); i=3;                                  i=3;
dx=x[1+i:n,.]-x[1:n-i,.];                dx=packr(x-lagn(x,i));
                                        ( または dy=trimr(x-lagn(x,i),i,0); )
print dx;                                print x-lagn(x,i); print dx;
```

==========

		ドット消去前	消去後
1	1-.=	.	
2	2-.=	.	
3	3-.=	.	
4	4-1=	3.0000000	3.0000000
5	5-2=	3.0000000	3.0000000
6	6-3=	3.0000000	3.0000000
7	7-4=	3.0000000	3.0000000
8	8-5=	3.0000000	3.0000000
9	9-6=	3.0000000	3.0000000
10	10-7=	3.0000000	3.0000000