5.3　モンテカルロシミュレーション 古典的仮定の変更　ver.0.1

前章では、下の５つの OLS 古典的仮定について１つ１つテストしてチェックしました。

【古典的仮定】

Ａ１）$E(e_i)=0$　　誤差の平均は０

Ａ２）$Var(e_i)=\sigma^2$　　均一分散

Ａ３）$Cov(e_i,e_j)=0$　　系列無相関

Ａ４）Ｘは non-stochastic　　説明変数は非確率変数

Ａ５）$e_i \sim N(0,\sigma^2)$　　誤差は正規分布にしたがう

ここでは、これらのうち３つの仮定を緩めて、 (A5)の正規性のない、(A2)の不均一分散である、(A3)の系列相関がある３つのケースについて、それぞれ極めて基本的でありますが、どのように設定するのかを説明します。まず最初に、上の仮定を満たしたままで、構造変化の設定から始めることにします。

構造変化の設定

【方法】データをグループ分けして、１つ以上の係数を若干変化させる。

ここでは、例えば、1000 個のデータを前半部分と後半部分の２つに分けて、それぞれ上の OLS の古典的仮定にもとづいて proc olssim で 500 個ずつのデータを生成します。（もちろん、タイムシリーズのデータであること仮定します。）その際に、係数βのうちの１つ以上の係数を２つのグループの間で少し変えてやることになります。ここでは、最終の係数を前半のグループは９、後半のグループは 9.1 にしています。その係数の貢献度にもよりますが、通常は、ほんの少しの係数の変化で十分です。

プログラム

```
new; cls;
b1={0.5,2,5,9};
{y1,x1}=olssim(b1,0,4,500);
b2={0.5,2,5,9.1};
{y2,x2}=olssim(b2,0,4,500);
y=y1|y2; x=x1|x2;

w=wstat(y,x);
n=rows(x); k=cols(x);
theta=0.948; @ At 5% @
call cusumtest(w,n,k,theta);
```

```
    c=0.25379;    @ At 5% @
    call cusumsqtest(w,n,k,c);

proc wstat(y,x);
    local n,k,w,ylag,xlag,bhatlag,e,t,v;
    n=rows(x); k=cols(x);
    w=zeros(n-k,1);
    t=k+1;
    do while t<=n;
        xlag=x[1:t-1,.]; ylag=y[1:t-1,.];
        bhatlag=inv(xlag'xlag)*xlag'*ylag;
        e=y[t,.]-x[t,.]*bhatlag;
        v=1+x[t,.]*inv(xlag'xlag)*x[t,.]';
        w[t-k,.]=e/sqrt(v);
        t=t+1;
    endo;
    retp(w);
endp;

proc (4) = cusumtest(w,n,k,theta);
    local j,wj,wmin,wmax,theta,wstar;
    j=seqa(k+1,1,n-k);
    wj=cumsumc(w)./stdc(w);
    wmax=theta*sqrt(n-k)+2*theta*(j-k)/sqrt(n-k);
    wmin=-theta*sqrt(n-k)-2*theta*(j-k)/sqrt(n-k);
    wstar=maxc(abs(wj*sqrt(n-k)/((n-k)+2*(j-k))));
    if wstar>theta;
        print "CUSUM test(RESULT:Reject H0)";
    else;
        print "CUSUM test(RESULT:Not reject H0)";
    endif;
        library pgraph;
        graphset;
        _pltype={1,1,6};
        title ("CUSUM TEST"); xlabel ("time j"); ylabel ("CUSUM");
        xy(j,wmin~wmax~wj);
```

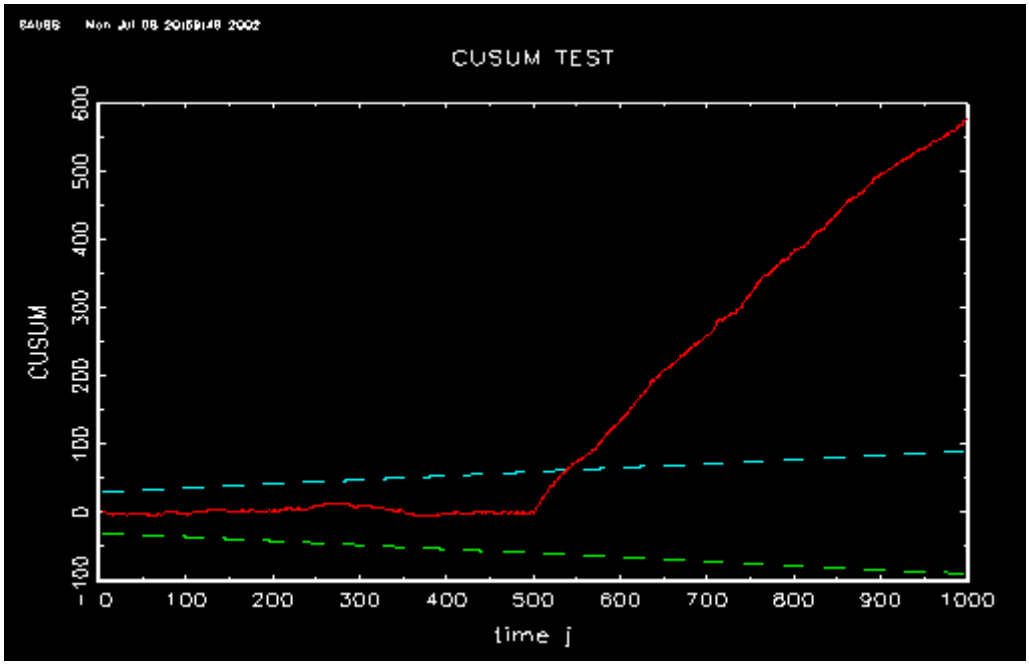
```
    retp(j,wmin,wj,wmax);
endp;

proc (4) = cusumsqtest(w,n,k,c);
    local qj,qmin,qmax,qstar,j,qstar;
    j=seqa(k+1,1,n-k);
    qj=cumsumc(w^2)./sumc(w^2);
    qmax=c+(j-k)/(n-k);
    qmin=-c+(j-k)/(n-k);
    qstar=maxc(abs(qj-(j-k)/(n-k)));
    if qstar>c;
        print "CUSUMSQ test(RESULT:Reject H0)";
    else;
        print "CUSUMSQ test(RESULT:Not reject H0)";
    endif;
        library pgraph;
        graphset;
        _pltype={1,1,6};
        title ("CUSUMSQ TEST"); xlabel ("time j"); ylabel ("CUSUMSQ");
        xy(j,qmin~qmax~qj);
    retp(j,qmin,qj,qmax);
endp;

proc(2)=olssim(b,myu,var,n);
    local k,x,u,y;
    k=rows(b);
    x=100*rndu(n,k-1);    @ Suppose this x is given. @
    x=ones(n,1)~x;
    u=myu+sqrt(var)*rndn(n,1);
    y=x*b+u;
    retp(y,x);
endp;
```

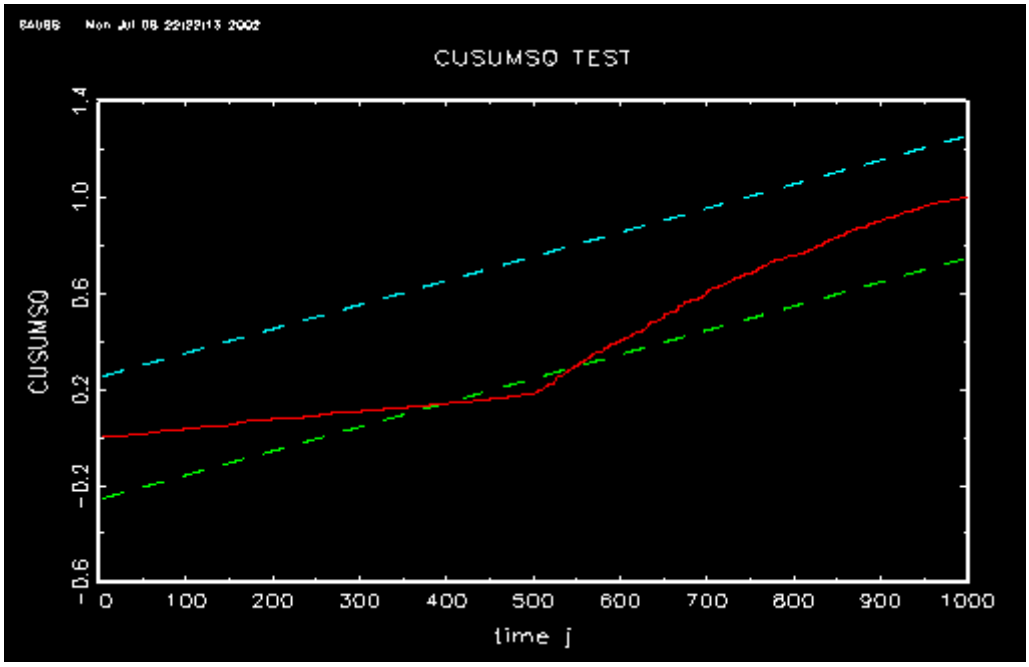
画面表示

CUSUM test(RESULT:Reject H0)

CUSUMSQ test(RESULT:Reject H0)

グラフ表示





CUSUMSQ のグラフでは、実際には、上限と下限の線は 0 と 1 のところでキンクするべきです。グラフから、3.10 で扱った CUSUM テストでも CUSUMSQ テストでも（くわしくはその章をご覧ください）、500 のあたりで明らかに構造変化があることがわかります。実際、上の画面表示でも両方のテストともに H0 が棄却されていて、構造変化があると判断できます。そのほかに、単位根テストなどのシミュレーションでは、トレンド項がある場合には、そのトレンド項の係数自体を単独で変更するやり方、それにトレンド項と定数項をあわせて変更するやり方などがあります。

Ａ５）非正規性の設定

【方法】（標準）正規乱数部分をほかの乱数関数に置きかえる。

今まで (標準)正規乱数を用いて誤差項をあらかじめ設定していたものを、ここでは、それ以外の分布の乱数に変更してやります。ただし、その平均は０となり、（Ａ１）の誤差項の平均０は保持されるものとします。下のプログラムでは、前章の正規性のテストと同じものを使って、最後の proc olssim の中の rndn(n,1)のところを何かほかの分布にしたがう乱数にかえてみます。あわせて、分散の var のところは４ではなくて、それぞれの分布の乱数の分散をそのまま使うために、var=1 に設定しておきます。下では、GAUSS 標準のガンマ分布乱数と、自作の Cauchy 分布およびχ2 分布の乱数を用いて、あらためて最初から OLS 残差を計算して、その正規性をテストしてみます。

プログラム

```
    new; cls;
    b={0.5,2,5,9};
    {y,x}=olssim(b,0,1,1000);       @ Set var=1 to use the distribution's variance itself. @

    b=inv(x'x)*x'y;
    e=y-x*b;
    print "          mean                    var";
    print/rd meanc(e) vcx(e);
        library pgraph;
        graphset;
        hist(e,19);
    call jbtest(y,x,0.05);

proc(4)=jbtest(y,x,pp);
    local n,k,b,e,skew,kurtosis,jb,p;
    n=rows(x); k=cols(x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    skew=meanc(e^3)/(meanc(e^2)^1.5);
    kurtosis=meanc(e^4)/(meanc(e^2)^2);
    print "Skewness of Residuals=" skew;
    print "Kurtosis of Residuals=" kurtosis;
    jb=n/6*(skew^2+(kurtosis-3)^2/4);
    p=cdfchic(jb,2);
    if p>=pp;
```

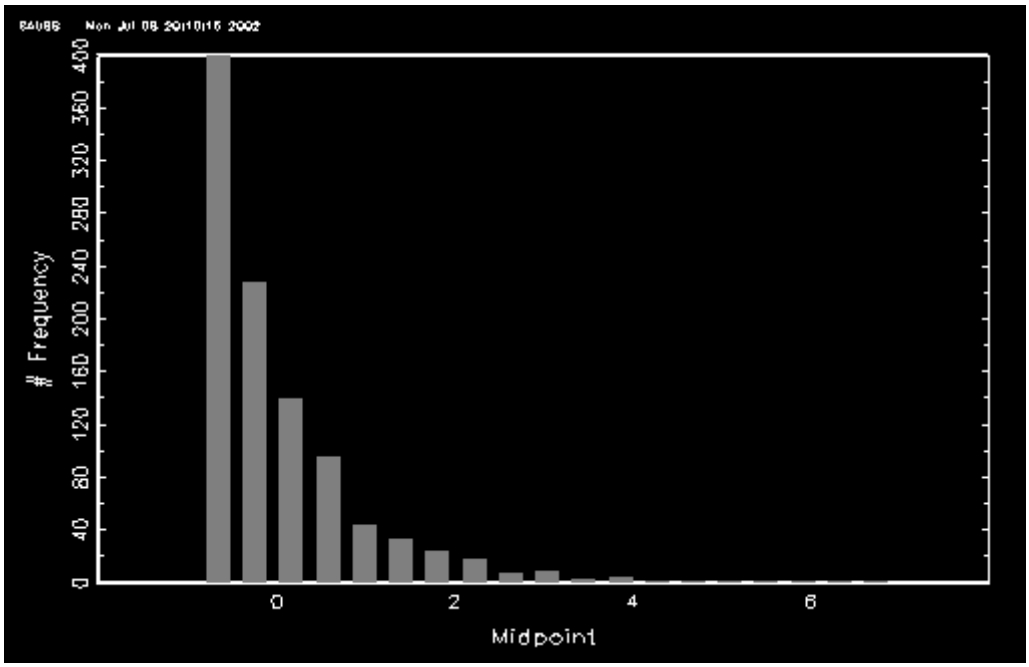
```
        print "Jarque-Bera Normality Test (RESULT:Not reject H0)";
    else;
        print "Jarque-Bera Normality Test (RESULT:Reject H0)";
    endif;
    print "X2( 2 )=" jb ; print "P-value=" p;
    retp(jb,p,skew,kurtosis);
endp;

proc(2)=olssim(b,myu,var,n);
    local k,x,u,y;
    k=rows(b);
    x=100*rndu(n,k-1);
    x=ones(n,1)~x;
    u=myu+sqrt(var)*rndgam(n,1,0.8); @ Gamma Distribution @
    /* Others:
    u=myu+sqrt(var)*rndcauchy(n,1); @ Cauchy Distribution @
    u= myu+sqrt(var)*rndx2(n,1); @ X2 Distribution @
    */
    y=x*b+u;
    retp(y,x);
endp;

proc rndcauchy(r,c);
    local x,u,v;
    u=rndn(r,c);
    v=rndn(r,c);
    x=u./v;
    retp(x);
endp;

proc rndx2(r,c,df);
    local m,x,i;
    m=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
```

```
        x=x.*x;
        m[.,i]=sumc(x');
        i=i+1;
    endo;
    retp(m);
endp;
```

画面表示（ガンマ分布のケース）

```
        mean                 var
        0.00000000          0.77446542
Skewness of Residuals=          2.2177436
Kurtosis of Residuals=          10.663272
Jarque-Bera Normality Test (RESULT:Reject H0)
X2( 2 )=          3253.5700
P-value=          0.00000000
```

グラフ表示



上のように、きれいな平均が０で減少していく分布になっています。Jarque-Bera テストの結果は、明らかに正規性が棄却されています。

画面表示（Cauchy 分布のケース）

```
        mean                 var
```

0.00000000 1027.39724182

Skewness of Residuals= 15.491079

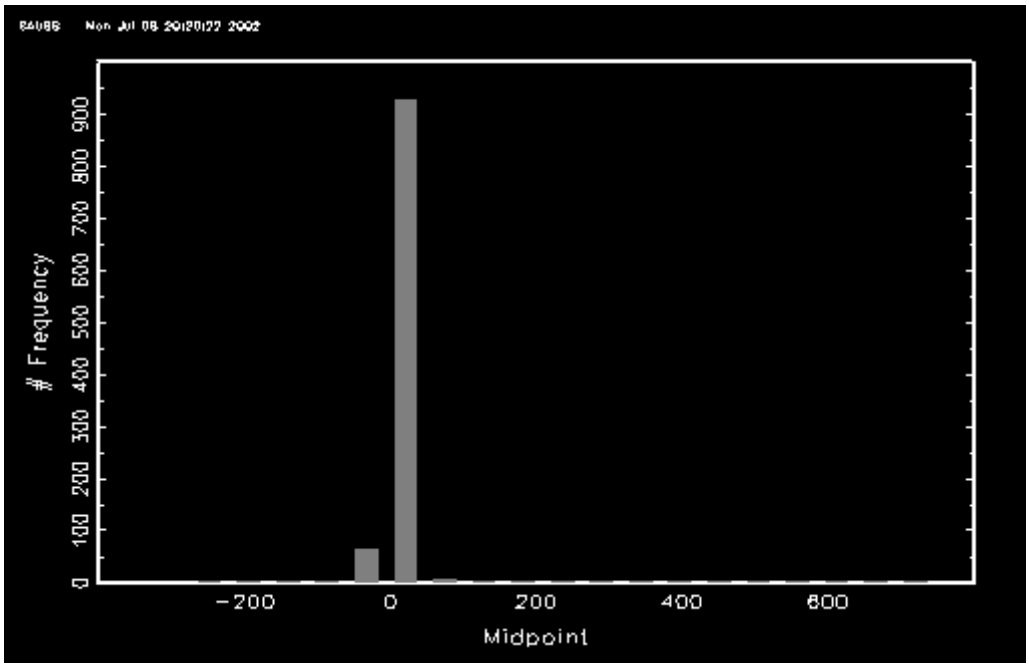
Kurtosis of Residuals= 343.73339

Jarque-Bera Normality Test (RESULT:Reject H0)

X2(2)= 4857954.1

P-value= 0.00000000

グラフ表示



上の場合、Cauchy 分布にしたがう乱数を生成するのに、簡易的に、２つの独立な標準正規乱数の商を用いています。これにより、分散が非常に大きい（無限に近い）テイルが分厚い分布ができます。OLS の残差では、０付近で頻度が集中しているように見えますが、分散の値やグラフのメモリの広がりからわかるように、非常にテイルが広がったものになっています。Jarque-Bara テストも明らかに正規性が棄却される結果になっています。なお、この Cauchy 分布にしたがう乱数では、この節の最初に扱った中心極限定理が成り立たないことが知られています。

画面表示（χ^2分布のケース）

mean var

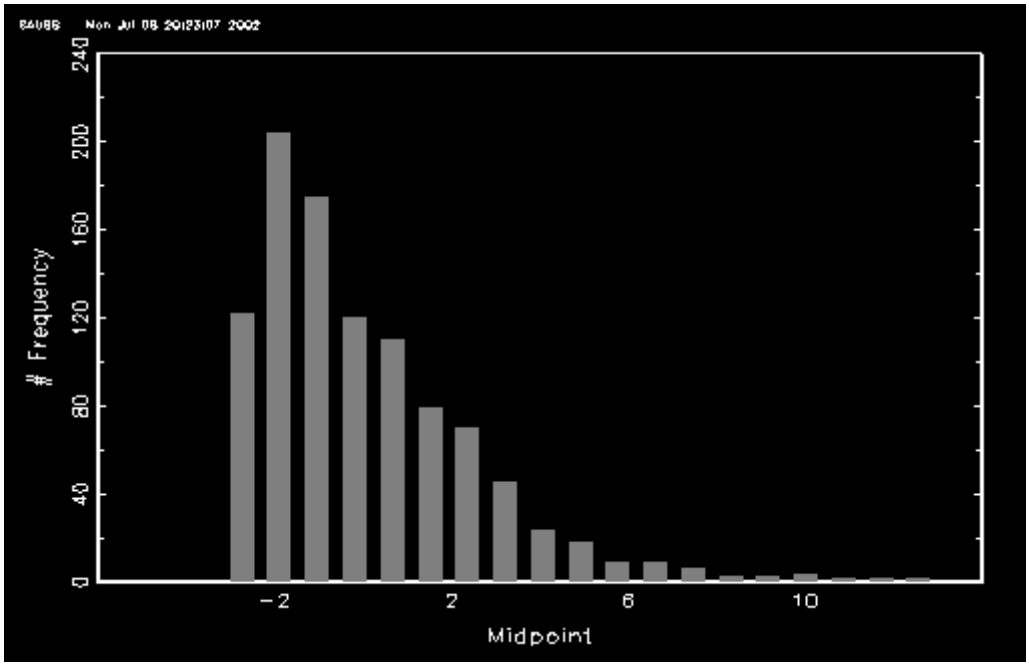
0.00000000 5.72345941

Skewness of Residuals= 1.4225567

Kurtosis of Residuals= 5.7639886

Jarque-Bera Normality Test (RESULT:Reject H0)
X2(2)=　　652.97359
P-value=　1.6165420e-142

次にここでは、χ^2にしたがう乱数を設定してみます。通常、乱数を作成する場合には、正式には、inverse CDF の proc を作成してから、それを用いて乱数を発生させるのが、より正確でフォーマルなのですが、ここでは、χ^2分布の定義「N(0,1)にしたがう互いに独立な df 個の確率変数の２乗和の分布」にしたがって、標準正規分布から生成しています。
グラフ表示



したがって、上の３つの例のように OLS 誤差項に正規性が保たれていない場合には、通常の OLS による推定では問題が生じることになります。（A５）の仮定を満たすためには、残差の設定には、正規分布が必要になることがわかると思います。

<u>不均一分散の設定</u>
【方法】ある変数に応じて比例または反比例するように正規乱数にかけてやる。
ここでは、説明変数の中のどれか、例えば所得の大小などに関するものなどに残差項が比例するように設定してみます。もちろん、複数の説明変数の組み合わせに比例することも考えられますし、そのトータルとしての従属変数またはその期待値に比例することもあります。ここでは、１説明変数に比例、反比例する場合にしぼって説明しましょう。

プログラム

```
new; cls;
b={0.5,2,5,9};                  /*   b={0.5,2,5,-9};   /*
{y,x}=olssim(b,0,4,100);
b=inv(x'x)*x'y;
e=y-x*b;
yhat=x*b;
    library pgraph;
    graphset;
    _plctrl=-1;
    xlabel("Yhat"); ylabel("e")
    xy(yhat,e);
call whitetest(y,x,0.05);


proc(2)=whitetest(y,x,pp);
local n,k,z,i,j,h,p,wh,b,y1,b1,e1,r2;
n=rows(x); k=cols(x);
z=zeros(n,k*(k+1)/2);
h=1; i=1;
do while i<=k;
    j=i;
    do while j<=k;
        z[.,h]=x[.,i].*x[.,j];
        j=j+1; h=h+1;
    endo;
    i=i+1;
endo;


b=inv(x'x)*x'y;
y1=(y-x*b)^2;
b1=inv(z'z)*z'y1;
e1=y1-z*b1;
r2=1-e1'e1/(y1'(eye(n)-1/n*ones(n,1)*ones(n,1)')*y1);
wh=n*r2;
```

```
    p=cdfchic(wh,k*(k+1)/2-1);



    if p>=pp;
        print "White Test (RESULT:Not reject H0)";
    else;
        print "White Test (RESULT:Reject H0)";
    endif;
    print/rz "X2(" k*(k+1)/2-1 ")=" wh ; print "P-value=" p;
    retp(wh,p);
endp;



proc(2)=olssim(b,myu,var,n);
    local k,x,u,y;
    k=rows(b);
    x=100*rndu(n,k-1);
    x=ones(n,1)~x;
    u=myu+sqrt(var)*rndn(n,1).*(x[.,4]/100);     /*    ./(x[.,4]/100)    */
    y=x*b+u;
    retp(y,x);
endp;
```

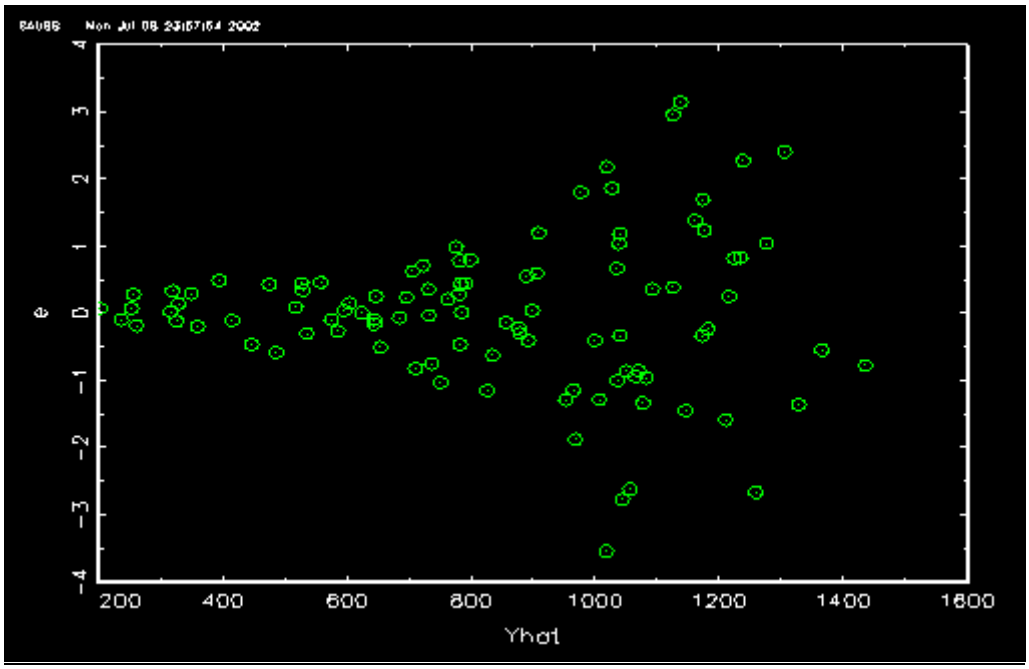
画面結果

```
White Test (RESULT:Reject H0)
X2(                    9 )=          32.015354
P-value=     0.00019791535
```

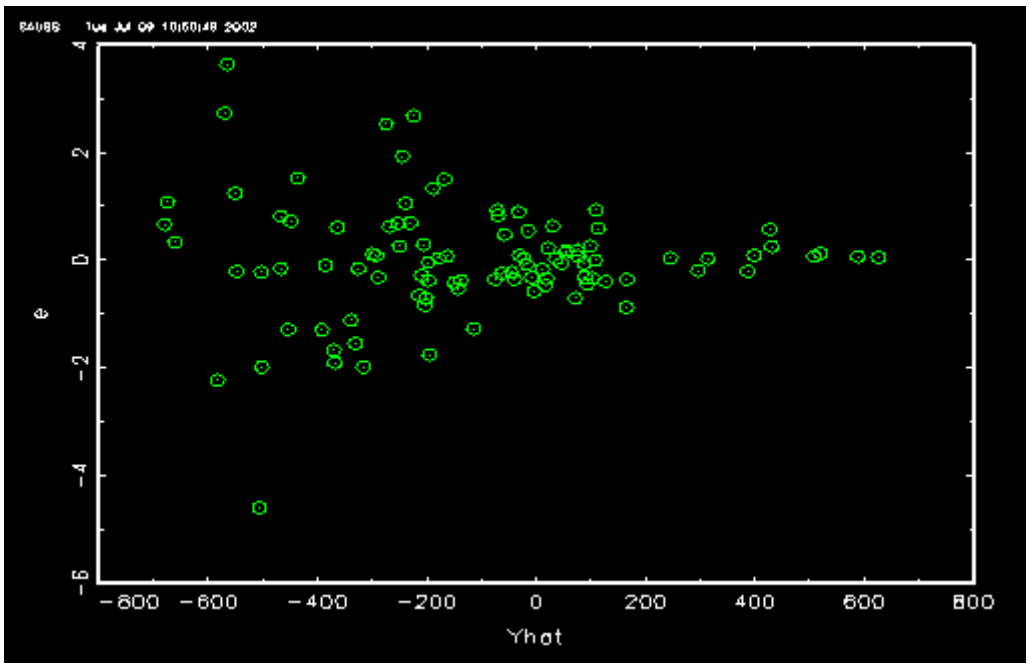
具体的には、乱数発生の場所に直接ある変数の何倍か（あるいは何分の１か）をかけあわせます。その変数の係数がプラスの貢献度を持っている場合には、残差の散らばりは発散していきます。他方、比例するように設定しても、その変数の係数がマイナスの貢献度をもっている場合には、残差の散らばりは縮小の方向になります。反比例させる場合は、乱数項にその変数の何倍か（あるいは何分の１か）の逆数をかけてやることになります。ただし、反比例ですから、急激に散らばりが縮小したり（その反対に発散したり）することがありますから、適宜パラメータの大きさを変化する必要があります。上のプログラムでは通常は u=myu+sqrt(var)*rndn(n,1)というふうに乱数で残差を設定するのに対して、u=myu+sqrt(var)*rndn(n,1).*(x[.,4]/100); という具合に、定数項を第 1 変数とする第４変数によって、残差が異なるようにします。この場合、第４変数の 100 分の１の値に応じて

残差の散らばり、すなわち標準偏差が比例するようになっています。なお、残差の分散と変数の値を比例させるには、変数の何分の１かを２乗して.*(x[.,4]/100)^2 としてやると設定できます。上の結果は、グラフ表示でも明らかなようにｅ-Yhat プロットでも明らかなように不均一分散が認められ、White テストでも均一分散が棄却される結果になっています。

グラフ表示



グラフ表示



他方、これを同じ変数の大きさにもとづいて、残差のちらばりを縮小にもっていくには、

いままで乱数項に何かをかけていたものを u=myu+sqrt(var)*rndn(n,1)./(x[.,4]/100);というぐあいに割ってやります。ここで注意が必要です。比例の反対のグラフは必ずしも反比例ではなくて、もし同じスケールで割ってやると、比例とは逆の残差の縮小よりもさらに急激に縮小してしまいます。別な方法に、その係数を b={0.5,2,5,-9};などというぐあいに９から－９に反転させてやっても残差縮小は実現できます。ただし、その場合には係数自体が変化するので、推定の意味自体が別のものになります。

系列相関の設定

【方法】u_0が０で所与のもと、$u_t=\rho u_{t-1}+e_t$ ここで $e_t \sim N(0,\sigma^2)$なる u_t再帰的に求める。具体的には、今まで proc olssim でもって u=myu+sqrt(var)*rndn(n,1);として設定してきた残差項を、今度はループでもって再帰的に足しあわせていくことになります。以下のプログラムでは、olssimar1 の最終項に係数 rho をともなって作成し、その中で、

```
u=zeros(n,1);
u[1]=myu+sqrt(var)*rndn(1,1);
i=2;
do while i<=n;
    u[i]=rho*u[i-1]+(myu+sqrt(var)*rndn(1,1));
    i=i+1;
endo;
```

というふうに、最初に u の領域をｎ×１の零ベクトルで確保しておいてから、まず、u[0]＝０ですから、u[1]=myu+sqrt(var)*rndn(1,1);になります。これを所与として、u[2]から順に、ｉ＝２からｎまで、u[i]=rho*u[i-1]+(myu+sqrt(var)*rndn(1,1));でもって再帰的にρにもとづいて u を生成していきます。こうしてできた u の列ベクトルを用いて、olssim と同様に、ｘからｙを生成します。これを{y,x}=olssim2(b,0,1,100,rho);として冒頭で呼び出した後、dwtest の proc を呼び出してテストします。ここでは、dwtest のリターンが１値なので、直接 print 文で画面表示しています。また、そのあと、OLS 残差ｅを順に１からのｎまでの数列に対して、ゼロ基準線つきでプロットしていっています。なお、下のプログラムの場合はρ＝0.95 の場合ですが、この他に、ρ＝－0.95 やρ＝０の場合も、その後で分析することにします。３行目の rho=のところの数値を変更してください。

プログラム

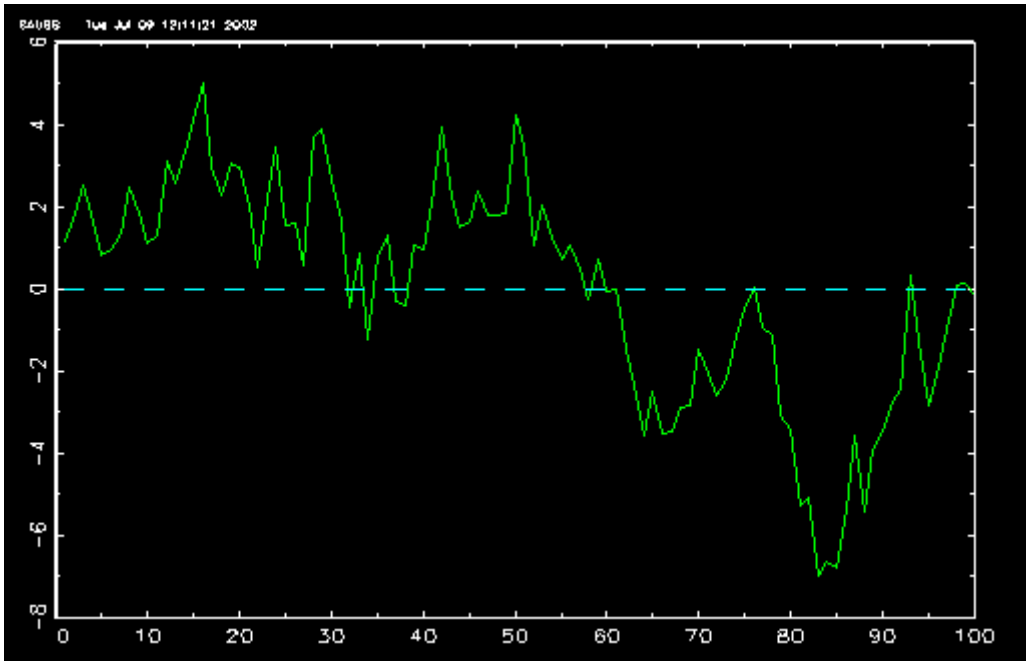
```
new; cls;
b={0.5,2,5,9};
rho=0.95;                          /* rho=-0.95 or rho=0 */
{y,x}=olssimar1(b,0,1,100,rho);

print/rz " rho=" rho;
```

```
    print "D.W.=" dwtest(y,x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    library pgraph;
    graphset;
    xy(seqa(1,1,rows(e)),e~zeros(rows(e),1));

proc dwtest(y,x);
    local n,b,e,dw;
    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^2);
    retp(dw);
endp;

proc(2)=olssimar1(b,myu,var,n,rho);
    local k,x,u,y,i;
    k=rows(b);
    x=100*rndu(n,k-1);
    x=ones(n,1)~x;
    u=zeros(n,1);
    u[1]=myu+sqrt(var)*rndn(1,1);
    i=2;
    do while i<=n;
        u[i]=rho*u[i-1]+(myu+sqrt(var)*rndn(1,1));
        i=i+1;
    endo;
    y=x*b+u;
    retp(y,x);
endp;
```

画面表示（ρ=0.95 のケース）

```
rho=                0.95
D.W.=         0.18934240
```

グラフ表示

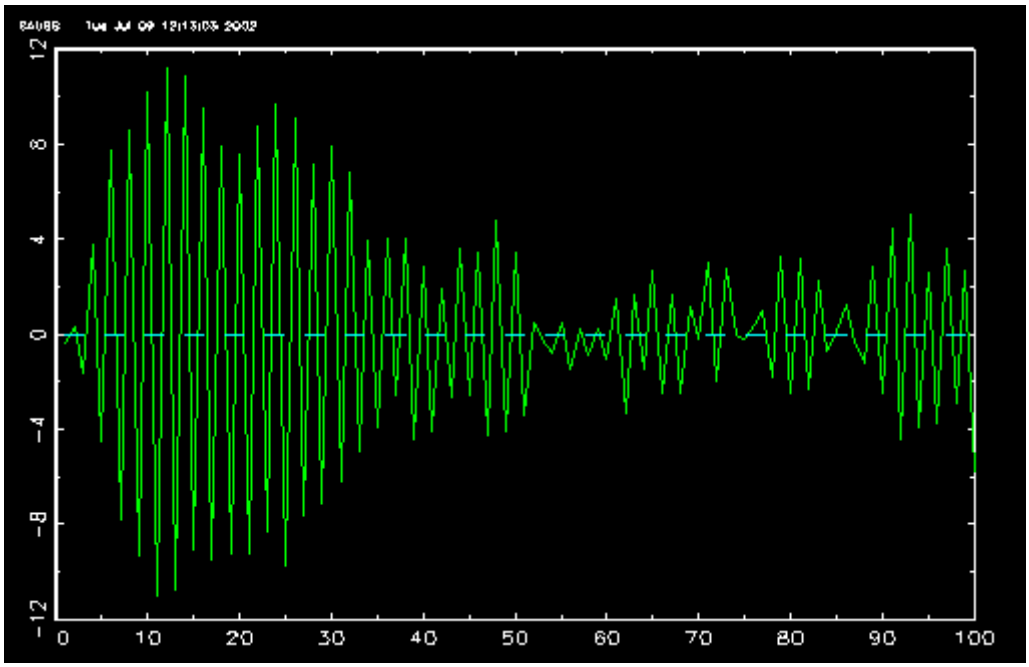


画面表示（ρ＝－0.95 のケース）

rho=　　-0.95

D.W.=　　3.9220094

グラフ表示



上のようなρ＝0.95 という正の（１次の）系列相関がある場合には、ダービンワトソン統計量は、０と dL の間に位置します。この場合、明らかに０に極めて近くなっていることが

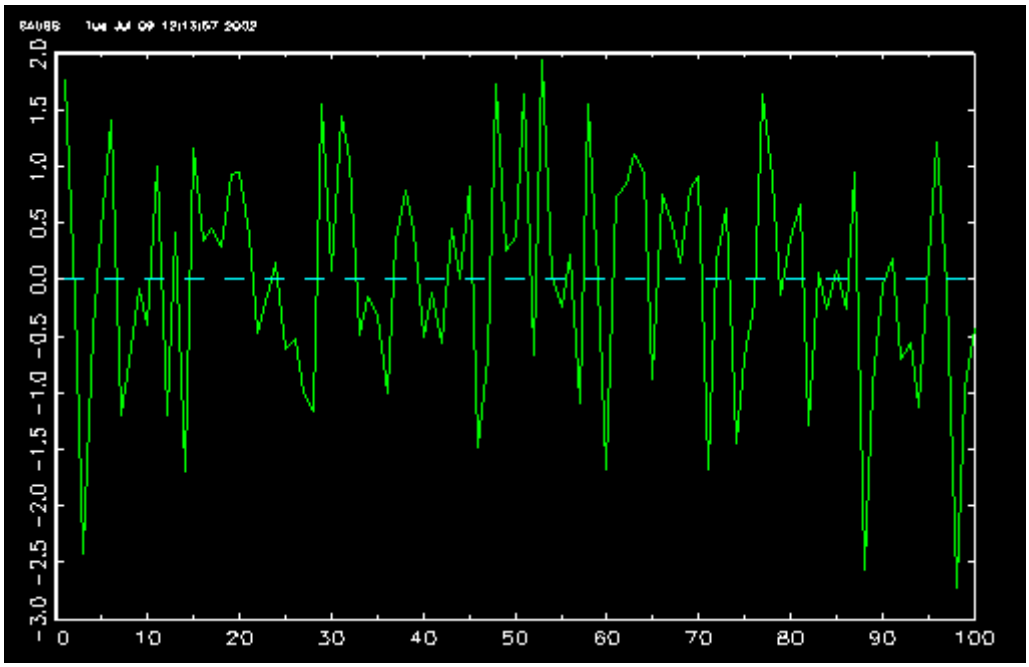
わかります。グラフ的には、残差がプラスに転じるとしばらくプラスのまま推移し、マイナスになるとしばらくマイナスのまま推移する傾向があります。他方、ρ＝－0.95 といった負の系列相関がある場合には、ダービンワトソン統計量は、４－dL と４の間に位置し、この場合極めて４に近い値になっています。グラフでは、ほとんどのパターンで、負正負正の交互の繰り返しになります。

画面表示（ρ＝０のケース）

rho=　　0

D.W.=　　1.9838728

グラフ表示



上のρ＝０の場合には、誤差項は、当然のことながら、系列相関のないものになります。ダービンワトソン統計量は、dU と４－dU の間に位置し、この場合、２に極めて近い値になります。

　このほかに、｜ρ｜＞１の場合には、OLS 残差は発散してしまいます。係数 rho=のところに適当な数、例えば２とか－2 とかを代入して確かめてみてください。

<u>高次の系列相関の設定</u> AR(q)

【方法】AR(2)の場合 u_0 および u_{-1} が０のもとで、$u_t=\rho_1 u_{t-1}+\rho_2 u_{t-2}+e_t$ ここで e_t～N(0, σ^2)なる u_t 再帰的に求める。３次以上の AR(q)の場合も同様に再帰的に求める。

今まで、AR(1)の系列相関では、ρは１つだけで最初の u_0 だけを０としていましたが、AR(2)ではさらに、u_1 を求めるために u_{-1} も０として計算します。以下、次のようになります。

$u_1 = \rho_1 0 + \rho_2 0 + e_1$

$u_2 = \rho_1 u_1 + \rho_2 0 + e_2$

$u_3 = \rho_1 u_2 + \rho_2 u_1 + e_3$

.

.

$u_n = \rho_1 u_{n-1} + \rho_2 u_{n-2} + e_n$

というふうに、第３期の u 以降は同じように１期前と２期前の u を用いて撹乱項と合わせて再帰的に求めることになります。すなわち、AR(q)の場合、最初のｑ期は一部または全部の過去の u を０と置いて求めることになります。ここで $u_t = \rho_1 u_{t-1} + \rho_2 u_{t-2} + e_t$ がなぜ AR(2)

なのかは、$X_t = \rho_1 X_{t-1} + \rho_2 X_{t-2} + e_t$ を $X_t = u_t$ で評価してやることによって容易にわかります。
プログラム上では、

```
u=zeros(n,1);
u[1]=myu+sqrt(var)*rndn(1,1);
u[2]=rho1*u[1]+(myu+sqrt(var)*rndn(1,1));
i=3;
do while i<=n;
    u[i]=rho1*u[i-1]+rho2*u[i-2]+(myu+sqrt(var)*rndn(1,1));
    i=i+1;
endo;
```

というふうになります。すなわち、u[1]と u[2]をループの外で定義してやっておいてから i=3 から u[i]=rho1*u[i-1]+rho2*u[i-2]+e;を n までループでまわしてやって再帰的に u[i]を求めます。これを proc の中でｘとβをかけ合わせたもの足して、ｙの系列を生成します。
プログラム

```
new; cls;
b={0.5,2,5,9};
rho1=0.1;   rho2=0.2;
{y,x}=olssimar2(b,0,1,100,rho1,rho2);

print/rz " rho1=" rho1;
print/rz " rho2=" rho2;
print "D.W.=" dwtest(y,x);
b=inv(x'x)*x'y;
e=y-x*b;
    library pgraph;
    graphset;
```

```
        xy(seqa(1,1,rows(e)),e~zeros(rows(e),1));
    call lbqtest(y,x,12,0.05);
    print;
    call bpqtest(y,x,12,0.05);

proc (0) = lbqtest(y,x,p,pp);
    local n,b,e,lbq,lbqj,p,j;
    n=rows(x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    j=1; lbq=0;
    do while j<=p;
        print/rz j "th order";;
        lbqj=n*(n+2)*(sumc(e[j+1:n,.].*e[1:n-j,.])/sumc(e^2))^2/(n-j);
        lbq=lbq+lbqj;
        if cdfchic(lbq,j)>=pp;
            print " Ljung-Box Q' = " lbq "(RESULT: Not reject H0)";
        else;
            print " Ljung-Box Q' = " lbq "(RESULT: Reject H0)";
        endif;
        j=j+1;
    endo;
endp;

proc(0)=bpqtest(y,x,p,pp);
    local n,b,e,bpq,bpqj,p,j;
    n=rows(x);
    b=inv(x'x)*x'y;
    e=y-x*b;
    j=1; bpq=0;
    do while j<=p;
        print/rz j "th order";;
        bpqj=n*(sumc(e[j+1:n,.].*e[1:n-j,.])/sumc(e^2))^2;
        bpq=bpq+bpqj;
        if cdfchic(bpq,j)>=pp;
            print " Box-Pierce Q = " bpq "(RESULT: Not reject H0)";
```

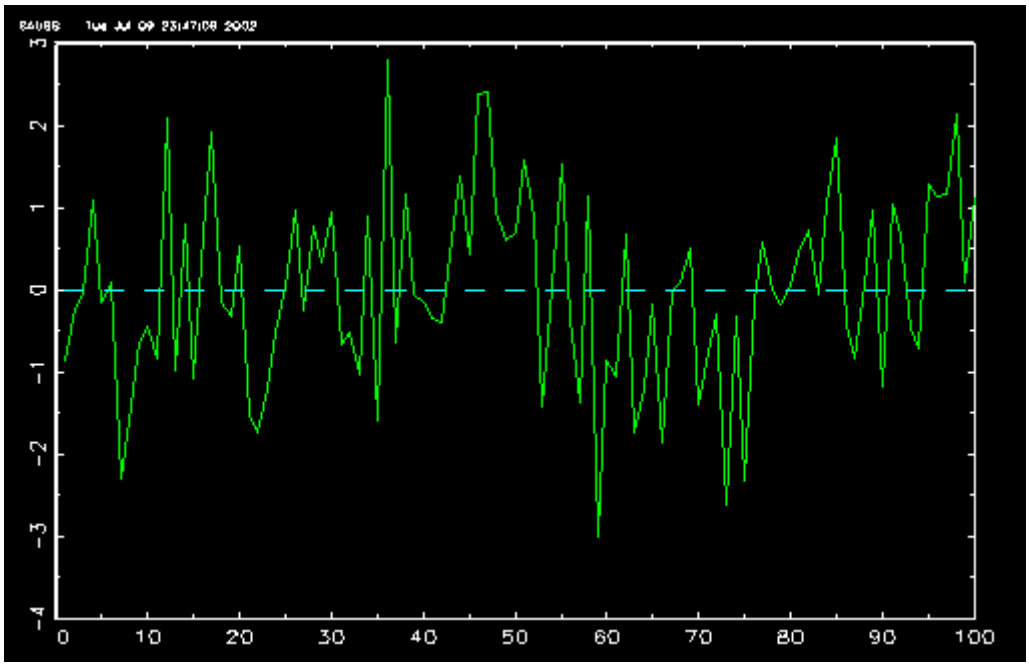
```
        else;
            print " Box-Pierce Q = " bpq "(RESULT: Reject H0)";
        endif;
        j=j+1;
    endo;
endp;

proc dwtest(y,x);
    local n,b,e,dw;
    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^2);
    retp(dw);
endp;

proc(2)=olssimar2(b,myu,var,n,rho1,rho2);
    local k,x,u,y,i;
    k=rows(b);
    x=100*rndu(n,k-1);
    x=ones(n,1)~x;
    u=zeros(n,1);
    u[1]=myu+sqrt(var)*rndn(1,1);
    u[2]=rho1*u[1]+(myu+sqrt(var)*rndn(1,1));
    i=3;
    do while i<=n;
        u[i]=rho1*u[i-1]+rho2*u[i-2]+(myu+sqrt(var)*rndn(1,1));
        i=i+1;
    endo;
    y=x*b+u;
    retp(y,x);
endp;
```

結果は、下のように２期目で初めて Q および Q'統計量が H0 を棄却していて（これらのテストについては３節の系列相関とテストの章をご覧下さい）、AR(2)が認められます。ただし、正規乱数の大きさと係数の関係およびその時々の乱数によって、必ずしも AR(2)にはっきりとなるわけではありません。AR(1)と同様に、rho1 と rho2 の関係が定常性の条件、

$\rho_2 < 1 - \rho_1$,　$\rho_2 < 1 + \rho_1$,　$-1 < \rho_2 < 1$

の$\rho_1 - \rho_2$プレーン上の三角形範囲からはずれる場合には発散していきますので、何か大きな数を係数に代入してみて確かめてください。

グラフ表示



画面表示

rho1=　　0.1

rho2=　　0.2

D.W.=　　1.7482690

1 th order Ljung-Box Q' =	1.4369305 (RESULT: Not reject H0)
2 th order Ljung-Box Q' =	8.8751408 (RESULT: Reject H0)
3 th order Ljung-Box Q' =	10.601177 (RESULT: Reject H0)
4 th order Ljung-Box Q' =	12.400731 (RESULT: Reject H0)
5 th order Ljung-Box Q' =	12.707603 (RESULT: Reject H0)
6 th order Ljung-Box Q' =	13.038168 (RESULT: Reject H0)
7 th order Ljung-Box Q' =	13.991489 (RESULT: Not reject H0)
8 th order Ljung-Box Q' =	14.713815 (RESULT: Not reject H0)
9 th order Ljung-Box Q' =	15.016494 (RESULT: Not reject H0)
10 th order Ljung-Box Q' =	15.127357 (RESULT: Not reject H0)
11 th order Ljung-Box Q' =	15.144614 (RESULT: Not reject H0)
12 th order Ljung-Box Q' =	15.162561 (RESULT: Not reject H0)

1 th order Box-Pierce Q =	1.3946679 (RESULT: Not reject H0)
2 th order Box-Pierce Q =	8.5411836 (RESULT: Reject H0)
3 th order Box-Pierce Q =	10.182610 (RESULT: Reject H0)
4 th order Box-Pierce Q =	11.876308 (RESULT: Reject H0)
5 th order Box-Pierce Q =	12.162120 (RESULT: Reject H0)
6 th order Box-Pierce Q =	12.466758 (RESULT: Not reject H0)
7 th order Box-Pierce Q =	13.335963 (RESULT: Not reject H0)
8 th order Box-Pierce Q =	13.987472 (RESULT: Not reject H0)
9 th order Box-Pierce Q =	14.257510 (RESULT: Not reject H0)
10 th order Box-Pierce Q =	14.355331 (RESULT: Not reject H0)
11 th order Box-Pierce Q =	14.370387 (RESULT: Not reject H0)
12 th order Box-Pierce Q =	14.385871 (RESULT: Not reject H0)

ダービンワトソン統計量の概算 critical value の計算

ここでは、ダービンワトソン統計量の H0: ρ=0 に対する pp=0.05（片側５％）の critical value をシミュレーションで繰り返すことによって概算を計算してみます。やり方は、まず olssimar1 で任意のρ（この場合 0.9）に対してｙと（定数項の１の列つき no）ｘの系列を生成させます。それを proc dwsim で times=10000 回、H0 の下で人工的に作り出したｙともともとのｘについての OLS から得られるダービンワトソン統計量を計算し、その分布の pp×times 番目の値を得ます。これをさらに timesmean=100 回繰り返して、それらの値の平均値を求めています。なお、timesmean=1 に設定すればすぐに結果は得られます。

プログラム（所要数十分）

```
new; cls;
b={0.5,2,5,9};
rho=0.9;
{y,x}=olssimar1(b,0,1,100,rho);
call dwsim(y,x,10000,100,0.05);


proc(0)=dwsim(y,x,times,timesmean,pp);
local n,k,b,u,s2hat,j,i,m,e,y1,d;
n=rows(x); k=cols(x);
b=inv(x'x)*x'y;
u=y-x*b;
s2hat=u'u/(n-k);
j=1; m=zeros(timesmean,1);
do while j<=timesmean;
```

```
        i=1; d=zeros(times,1);
        do while i<=times;
            e=sqrt(s2hat)*rndn(n,1);
            y1=x*b+e;
            d[i]=dwtest(y1,x);
            i=i+1;
        endo;
        d=sortc(d,1);
        m[j]=d[pp*times];
        j=j+1;
    endo;
    print/lz "    n=" n;
    print/lz "k-1=" k-1;
    print/rz pp "% level";
    print/lo "H0:rho=0       d=" meanc(m);
endp;

proc dwtest(y,x);
    local n,b,e,dw;
    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^2);
    retp(dw);
endp;

proc(2)=olssimar1(b,myu,var,n,rho);
    local k,x,u,y,i;
    k=rows(b);
    x=100*rndu(n,k-1);
    x=ones(n,1)~x;
    u=zeros(n,1);
    u[1]=myu+sqrt(var)*rndn(1,1);
    i=2;
    do while i<=n;
        u[i]=rho*u[i-1]+(myu+sqrt(var)*rndn(1,1));
```

```
        i=i+1;
    endo;
    y=x*b+u;
    retp(y,x);
endp;
```

画面表示

```
  n= 100
k-1= 3
                0.05 % level
H0:rho=0      d= 1.6745158
```

なお、上の d の critical value の値はおおむね dL と dU の中間に位置します。また、ρの値によって大きく変化はしません。より正確をきすやり方としては、このように繰り返しの平均をとるのではなくて、後の章で取り扱うブートストラップなどでの残差の繰り返しリサンプリングによって、分布に依存しない critical value を得ることができます。上のプログラムは元の OLS 回帰の分散だけをもとに乱数を残差項に発生させることを繰り返した簡易バージョンです。

Additional Notes[Nyblom-Hansen の構造変化のテストの無効性]

ここで、構造変化のテストとして Nyblom-Hansen テストがモンテカルロスシミュレーションによって、実は信頼性のまったくないテストであることを実証してみよう。上の CUCUM テストでは、var=4 の場合についてテストしてみた。定数項を除く、X1 から X3 は[0,100]の範囲にあるので、残差が４というのは比較的小さい値である。同じ設定で最終 X3 の係数を少しだけ変更して同じく 9 から 9.1 に変えると Lc 値は非常に大きな値となり他の係数の「パラメータ安定」の帰無仮説も棄却してしまう。ここで、自作の olssim の残差分散 var を１から 10000 まで変化させてみよう。おもしろい結果が出る。なお、各パラメータの安定性のテスト Lj の臨界点は 0.748(1%), 0.470(5%)で、ジョイントテスト Lc の臨界点は 1.600(1%), 1.240(5%)とされている。

プログラム

```
rndseed 1;
v={1,4,100,400,3600,10000};
i=1;
do while i<=rows(v);
    var=v[i];
    b1={0.5,2,5,9};
    {y1,x1}=olssim(b1,0,var,500);
```

```
    b2={0.5,2,5,9.1};
    {y2,x2}=olssim(b2,0,var,500);
    y=y1|y2; x=x1|x2;
    dat=y~x;
    dataname={"Y","CONST","X1","X2","X3"};
    print/lz "#####    var=" v[i] "#####";
    call lc(dat,dataname);
    i=i+1;
endo;

proc(2)=olssim(b,myu,var,n);
    local k,x,u,y;
    k=rows(b);
    x=100*rndu(n,k-1);    @ Suppose this x is given. @
    x=ones(n,1)~x;
    u=myu+sqrt(var)*rndn(n,1);
    y=x*b+u;
    retp(y,x);
endp;

/*
LC.PRC
This is a Gauss procedure which calculates the LC test
for parameter instability in linear models as reported
in Hansen (1992) Journal of Policy Modeling.

Bruce E. Hansen
Department of Economics
Social Science Building
University of Wisconsin
Madison, WI 53706-1393
bhansen@ssc.wisc.edu
http://www.ssc.wisc.edu/~bhansen/

The format for the command is
lc(dat,datname)
```

```
The matrix "dat" is a nx(k+1) matrix, with the dependent
(y) variable in the first colume, and the independent (x)
variables in the remaining columes.  If a constant is desired
in the regression, it should be included in the dat matrix.
The vector "datname" is a (k+1)x1 list of variable names,
for the columes of "dat".
The output is printed to the screen.
*/
proc (0) = lc(dat,datname);
local n,k,y,x,m,b,e,xe,e2,sig2,e2m,se,datnames,
f,s,v,lj,li,bb,ff,t1,pr,r2;

n=rows(dat);
k=cols(dat)-1;
y=dat[.,1];
x=dat[.,2:k+1];
m=invpd(moment(x,0));
b=m*(x'y);
e=y-x*b;
xe=x.*e;
e2=e.^2;
sig2=meanc(e2);
e2m=e2-sig2;
se=sqrt(diag(m*moment(xe,0)*m))|sqrt(meanc(e2m.^2)/n);
r2=1-sumc(e2)/sumc((y-meanc(y)).^2);
f=xe~e2m;
s=cumsumc(f);
v=f'f;
lj=sumc(sumc(s.*(s*invpd(v))))/n;
li=sumc(s.^2)./diag(v)/n;
datnames=datname[2:k+1]|"Variance";
bb=datnames~(b|sig2)~se~li;
ff = "#*.*lG";
t1="" $+ "Dependent Variable = " $+ datname[1];
$t1;
"";
```

```
pr = printfm("Variable"~"Estimate"~"St Error"~"Lc",0~0~0~0,ff~16~8);"";
"          ____________________________________________________";
pr = printfm(bb,0~1~1~1,ff~16~8);"";
"          ____________________________________________________";
"";"";
"Joint LC         " lj;
"R-squared     " r2;
"";"";
"*************************************************************";
"";"";
endp;
```

画面表示

```
#####  var= 1                #####
Dependent Variable = Y
        Variable        Estimate        St Error              Lc
       ____________________________________________________
           CONST      0.67391815      0.25667925       54.899239
              X1       2.0009005    0.0034044519       42.100334
              X2       4.9971977    0.0033798029       41.791491
              X3       9.0475927    0.0036184823       42.515441
        Variance       9.7760983      0.31722652      0.28293371
       ____________________________________________________
Joint LC              59.623847
R-squared          0.99990112
************************************************************

#####  var= 4                #####
Dependent Variable = Y
        Variable        Estimate        St Error              Lc
       ____________________________________________________
           CONST     -0.39827311      0.34612575       42.174501
              X1       2.0086979    0.0038564758       31.343654
              X2       5.0054120    0.0039741972       31.900614
              X3       9.0523872    0.0040731822       36.149008
        Variance       12.073116      0.47598281      0.22202352
```

Joint LC	42.508110		
R-squared	0.99986457		

****

var= 100

Dependent Variable = Y

Variable	Estimate	St Error	Lc
CONST	2.4563709	1.0782212	7.6469629
X1	1.9818105	0.011714179	5.7773447
X2	4.9909590	0.011321791	6.3293366
X3	9.0434762	0.011673401	8.5163923
Variance	115.39796	5.2947563	0.057419208

Joint LC	9.1322242		
R-squared	0.99874164		

****

var= 400

Dependent Variable = Y

Variable	Estimate	St Error	Lc
CONST	-1.4950028	1.9685177	1.0647527
X1	2.0049853	0.021874149	0.58891549
X2	5.0179576	0.021834163	0.96938032
X3	9.0577157	0.021042624	2.6905455
Variance	391.00755	16.612094	0.30778716

Joint LC	3.8883221		
R-squared	0.99589132		

****

var= 3600

Dependent Variable = Y

Variable	Estimate	St Error	Lc

CONST	3.1989887	5.3889479	0.79684564
X1	1.9585391	0.064458982	1.1150503
X2	5.0209817	0.063456521	0.76586667
X3	9.0331635	0.063614156	0.74639215
Variance	3523.1468	161.47645	0.13366169

Joint LC 1.6230222
R-squared 0.96611213
****

var= 10000
Dependent Variable = Y

Variable	Estimate	St Error	Lc
CONST	0.57089104	9.0315287	0.35175396
X1	1.9359710	0.10648591	0.27112246
X2	5.1764647	0.098889136	0.18176960
X3	8.9328116	0.10342660	0.22495172
Variance	8961.1750	415.07626	0.16107142

Joint LC 0.94951126
R-squared 0.91119522
****

Hansen 教授の procedure 中、m=invpd(moment(x,0));とあるのは、inv(x'x)または invpd(x'x)のことである。ｆｆでフォーマットのある print 文を implicit に行なっている。正式には 2.6 章文字と変数ラベルの章で行なったように printfmt で扱うのが正統的なプログラム方法である。理論的な日本語での紹介説明は、蓑谷千凰彦『計量経済学の理論と応用』pp.292-294 を参照のこと。

明らかに、モンテカルロスタディにより残差分散を非常に小さい値から大きい値に変化させてやると、この統計値は容易に棄却と受容が逆転するばかりか、他のパラメータの安定性のテストの統計値にも影響を与えていて、信頼性の全くないものであると結論できる。