5.2　モンテカルロシミュレーション OLS と古典的仮定　ver.0.1

ここでは、初歩の初歩ですが、OLS の Classical Assumptions とモンテカルロの乱数設定について取り扱います。極めて基本ですが、すべての設定はここから始まり、どの仮定をどう変更するのか、あるいはそのまま保持するのかということを念頭において、どんな複雑なシミュレーションも取り扱われるべきものです。

【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

これを単純な１説明変数、１切片、１従属変数からなる $y_i=\alpha+\beta x_i+u$ にどう反映させるのでしょうか。それは、通常の５つの仮定を調べたのとは逆の手順で x と u のところから設定していくことになります。

手順１）データが存在しない場合には、次善の策として説明変数Ｘを rndu によって定める。例えば０から 100 までの間の一様乱数を考えるのなら、一様乱数をスケーリングして、100 倍したものをｘとする。（慣習的に一様乱数を使う。）

```
n=2000;
x=100*rndu(n,1);
```

手順２）誤差項 u の設定には、標準正規乱数 rndn を用いて、例えば $N(0,\sigma^2 I)$の正規分布にしたがうとしたいのであれば、分散のルートを rndu にかけスケーリングする。（さらに誤差の平均が０でない場合には、これに誤差の平均μを加える。通常はμ＝0 を設定する。）

```
myu=0; var=4;
u=myu+sqrt(var)*rndn(n,1);
```

手順３）係数αとβを任意に与えて、$y_i=\alpha+\beta x_i+u$ の式から従属変数ｙを求める。

```
a=1; b=2;
y=a+x*b+u;
```

手順４）できあがったｘとｙの系列を用いて分析をする。必要であればさらにこれをループにより times 回繰り返してやって分析をする。ただし、残差 u の乱数生成の文は、ループの内側に入れて、その都度異なる乱数が使われるようにする。

プログラム

```
new; cls;
n=2000;
x=100*rndu(n,1);     @ Suppose this is given. @
```

```
myu=0; var=4;
u=myu+sqrt(var)*rndn(n,1);
a=1; b=2;
y=a+x*b+u;
print y~x;
```

上のプログラムのように、まずｘを決めます。（OLS の枠組みのなかでは、少なくとも、ｘとｙとを同時に乱数で設定するのではありません。必ずｘを与えられたものとします。）データがあればデータを使います。データがなければ、慣例として、一様乱数を用いてこの説明変数も作成されます。厳密に言えば、（Ａ４）の non-stochastic ではなくなりますが、これは少なくともｙやその他の変数に依存して決まってくる確率変数ではありません。そういう意味で（Ａ４）を考えて設定しています。次に、残差項 u にはN(0,1)を生成する rndn を用いて、分散のルートをとった値にこれをかけることによって、スケーリングすることによって $N(0,\sigma^2)$を作り出しこれは設定します。残差の平均は通常は０で（Ａ１）を満たすように設定します。x と u に対して、係数αとβを外部から与えます。これらをもとに、ｙを計算します。OLS 自体が x と u のリニアコンビネーションですので、できあがったｙともともとのｘとを回帰分析すれば、その残差ももともと設定した残差 u は $N(0,\sigma^2)$と同じようなふるまいをするでしょう。（Ａ１）の残差の平均は０のはずですし、（Ａ２）分散も均一であるはずです。（Ａ３）の系列間の共分散は、$N(0,\sigma^2)$がもとですので、これもテストをすれば０であることがわかるでしょう。（Ａ５）の残差項の正規性も当然成り立つはずです。このように、OLS ベースのシミュレーションは、与えられた説明変数をもとに、従属変数を正規乱数を用いて「逆から仕組んでいく」ことになります。

なお、実際のモンテカルロシミュレーションでは、これに times 回の繰り返しを加えることによって、分析することがよく見られます。つまり、下のプログラムのように、

プログラム

```
new; cls;
n=2000;
x=100*rndu(n,1);
times=1000;
m=zeros(times,2);
i=1;
do while i<=times;
    myu=0; var=4;
    u=myu+sqrt(var)*rndn(n,1);
    a=1; b=2;
    y=a+x*b+u;
    x1=ones(n,1)~x;
```

```
    b1=inv(x1'x1)*x1'y;
    e=y-x1*b1;
    m[i,.]=meanc(e)~vcx(e);
    i=i+1;
endo;
print "        mean                    var";
print/rd meanc(m)';
```

画面表示

```
        mean                 var
        0.00000000           3.99466200
```

上の例では n=2000 個の系列の OLS 分析をしてその残差ｅを調べた時、これを times=1000 回繰り返すと、もともと１回でも平均は E(ｅ)＝0 ですが、その分散はもともとの u の分散の σ^2 の値４に、大数の法則により、近づいていきます。すなわち、Var(ｅ)→４です。ちなみに、u のμを０以外の数に設定しても、OLS 回帰で定数項に吸収されるので、１回の繰り返しであっても最初から残差の平均は常に E(ｅ)＝0 です。

それでは、繰り返しのない場合について、実際に残差のふるまいについて、不均一分散および系列相関そして正規性の諸テストをしておきましょう。これまでは、１切片、１説明変数のケースでしたが、２説明変数以上の一般的なケースで考えましょう。

具体的な変更点は、

（手順１）x=100*rndu(n,1);のところを x=100*rndu(n,k-1);とｋ-1 列分考える。

（手順２）残差 u の設定は説明変数が何個になろうと１列分だけで変更なし。

（手順３）a と b の設定を複数にする。ここでは定数項も含んだ係数を一括して、列で例えば b={0.5,2,5,9}としてやりましょう。ここでは定数項も含んで４つの係数がありますから k-1=3 になります。なお、最初の 0.5 が定数とします。

まずは、上の設定例にしたがって、定数項の１の列を含んだｘの行列とｙの系列を作成する procedure を作ってみましょう。

プログラム

```
new; cls;
/* rndseed 9999; */      @ If you need the same result, set rndseed here. @
b={0.5,2,5,9};
{y,x}=olssim(b,0,4,1000);
print y~x;

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;
```

上のプログラムのようにすれば、ｘの最初の列に１ばかりの入ったｘのデータとそれをもとにシミュレートされた１列のｙのデータが生成されます。なお、いつでも同じデータが作成されるようにするためには、「rndseed 任意の番号;」で設定できます。プログラムではｂに最初に定数項の値を含む４つの係数が１列に入っています。この行数を数えて、ｋを割り出します。定数項を除いた k-1 個のデータ系列をあらかじめ一様乱数によって生成しておきます。100 はスケーリングで何でもかまいません。列ごとに変えて、あとでマージしてももちろんかまいません。ｎ×(k-1)になるようにします。これに通常の行列での OLS の計算と同じようにｘの最初の列に１ばかりの列を定数項に対応するものとして入れて、これをあらためてｘとおき直します。残差項は、以前と同じようにμと var に基づいて標準正規乱数から正規乱数への変換で計算して u とおきます。ｘと u をもとにｂを行列形式でかけ合わせて、x*b+u とすることによって、ｙを計算します。これにより、ｙともともとのｘの系列が生成されます。

Ａ１）$E(e_i)=0$　(誤差の平均は０)

誤差項の平均が０になっていることを OLS 誤差ｅのプロットとともに確かめてみましょう。プロットの基準線はゼロの線であるとします。上で作成した proc olssim で 1000 個ずつのｙとｘの系列を生成させ、それをもとにもう一度 OLS をしなおして、その OLS 残差ｅを求めて、meanc(e)と vcx(e)で、平均と分散を求めます。（なお、stdc(e)とすると標準偏差をもとめることができます。）

プログラム

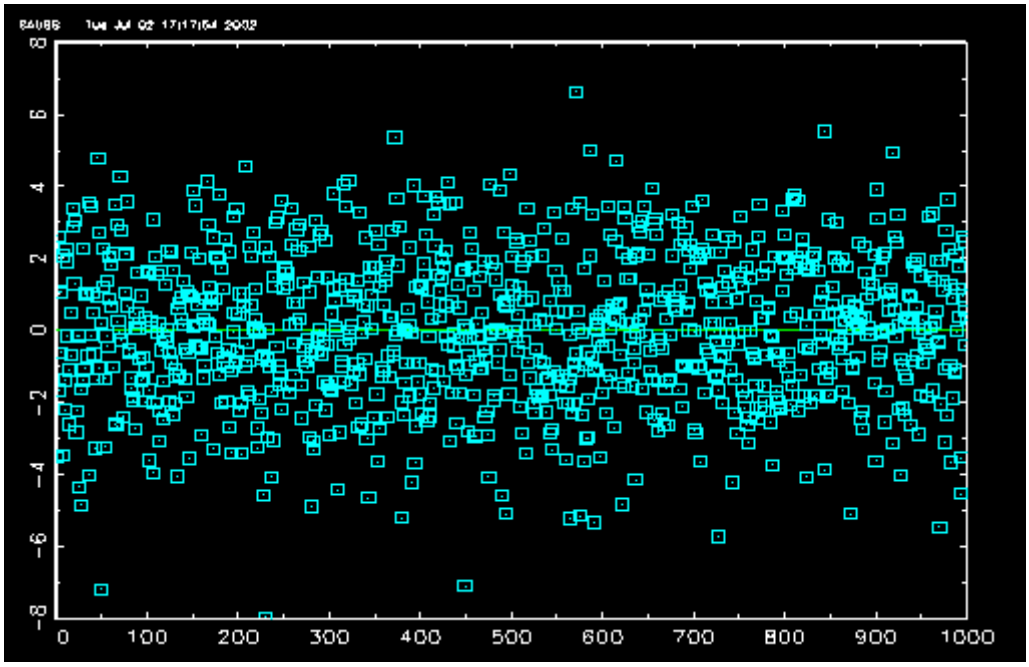
```
    new; cls;
    b={0.5,2,5,9};
    {y,x}=olssim(b,0,4,1000);

b=inv(x'x)*x'y;
e=y-x*b;
print/rd meanc(e) vcx(e);
    library pgraph;
    graphset;
    _plctrl={0,-1};
```

```
    xy(seqa(1,1,1000),zeros(1000,1)~e);

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

0.0　　4.16100131

グラフ表示



上のように、常に残差の平均はゼロに限りなく近くなります。分散は繰り返して平均をとれば、以前のプログラムのように u の分散の４に近くなりますが、１回かぎりのシミュレーションでは、おおよそ４のまわりの数になります。グラフの作成で、_plctrl={0,-1};とあるのは、pgraph の line control のグローバル設定をしている部分です。ｙの最初の列の系列つまりゼロの点にはデフォルトの０どおり線で点を結びます。次の列つまり残差の系列の点には-1 と設定して線で結ばないようにしています。なお、色は、特に設定しないかぎり緑、ブルー. . .の順になります。

Ａ５）$e_i \sim N(0, \sigma^2)$　(正規性)

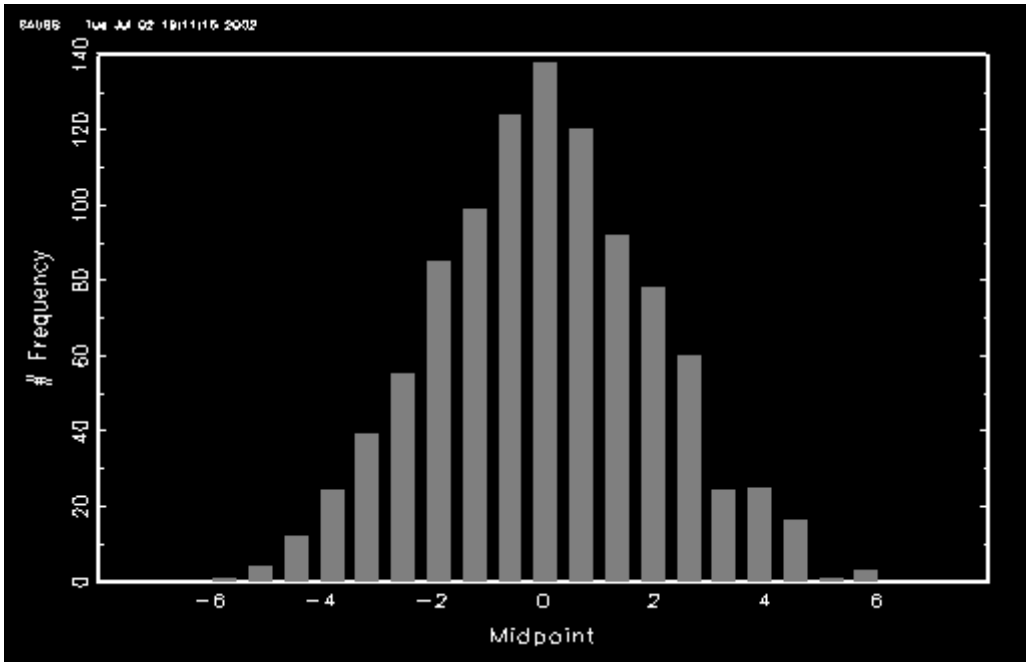
次に正規性を見るために、OLS 残差ｅのヒストグラムを hist で作成して視覚的に散らばりの分布を見ます。さらに、第３節の不均一分散のテストのところでプログラムしました Jarque-Bera Normality Test でテストしてみます。

プログラム

```
new; cls;
b={0.5,2,5,9};
{y,x}=olssim(b,0,4,1000);

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);   @ 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;
```

グラフ表示



画面表示

```
        mean                 var
        0.00000000           3.99403619
Skewness of Residuals=       0.032739854
Kurtosis of Residuals=        2.9675831
Jarque-Bera Normality Test (RESULT:Not reject H0)
X2( 2 )=         0.22154559
P-value=         0.89514210
```

OLS 残差の分布は、ヒストグラムにより、ほぼ正規性が保たれていることがわかります。
厳密に、Jarque-Bera Test を行なうと、H0 が棄却されないので正規性が確認できます。

Ａ２）Var(e_i)＝σ^2　(均一分散)

今度は、不均一分散になっていないか White Test でチェックしてみます。このテストはメモリを大量に消費しますので、このところだけ n=1000 から n=100 に変更しています。あわせて、ｅ－Ｙhat プロットで視覚的に不均一分散になっていないかを見てみます。

プログラム

```
new; cls;
b={0.5,2,5,9};
{y,x}=olssim(b,0,4,100);   @ Because of workspace limitation, n=100 here. @

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;
```

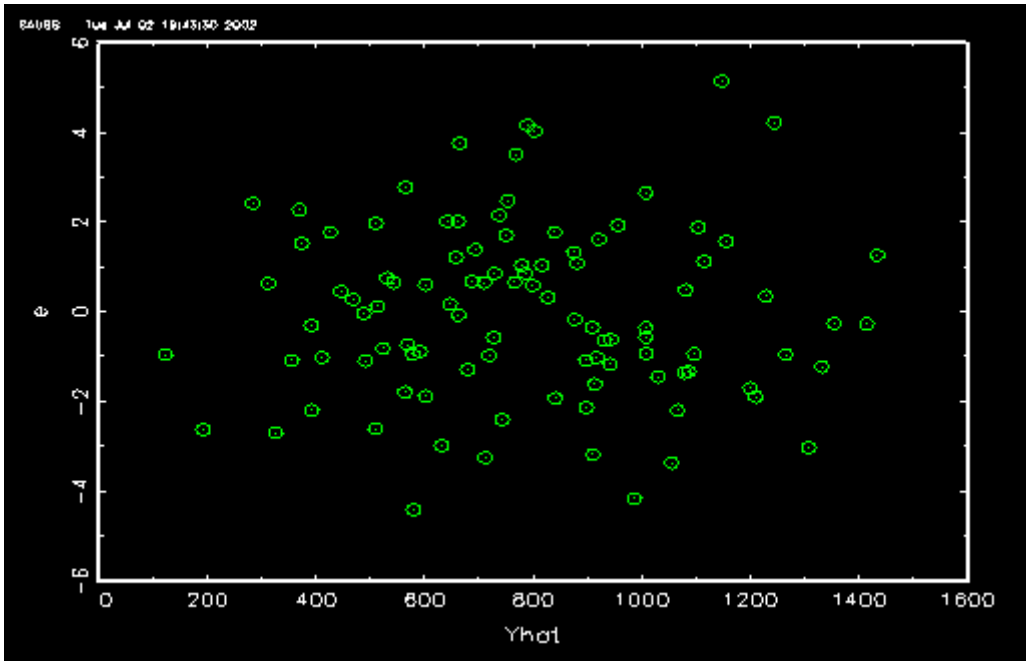
```
    b1=inv(z'z)*z'y1;
    e1=y1-z*b1;
    r2=1-e1'e1/(y1'(eye(n)-1/n*ones(n,1)*ones(n,1)')*y1);
    wh=n*r2;
    p=cdfchic(wh,k*(k+1)/2-1);

    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);   @ 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

```
White Test (RESULT:Not reject H0)
X2(                    9 )=          7.8103732
P-value=         0.55336338
```

上のように、White テストの H0 は棄却されないので、不均一分散はないと言えます。なお、White Test のプログラムがどうなっているかは、もう一度、第３節の不均一分散とテストのところをご覧ください。ここでは、その proc を呼び出して 0.05（5%）で評価をしています。次のグラフも 100 個のデータ系列で十分とは言えませんが、不均一ではないと視覚的に判断できます。

グラフ表示（ｅ－Ｙhat プロット 100 個のデータ系列の場合）



Ａ３）Cov(e_i,e_j)＝0　(系列無相関)

　最後のテストとして、誤差項間の系列相関の有無についてテストしてみます。ここでは、古典的に Durbin-Watson のテストをしてみます。

プログラム

```
    new; cls;
    b={0.5,2,5,9};
    {y,x}=olssim(b,0,4,1000);
    dw=dwtest(y,x);
    print "D.W.=" dw;

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)=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

D.W.=　　2.0041844

これは、ほとんど２に近い値をとるので、テーブルを見てチェックするまでもなく、系列相関はないといってよいでしょう。したがって、Ａ４）のＸは non-stochastic（説明変数は非確率変数）というところを拡大解釈するならば、古典的な仮定の要件はすべてみたすことになります。

今度は、OLS の推定量についての諸性質をあらためて確かめておきましょう。

【OLS 推定量の諸性質】

Ⅰ）係数 $\hat{\beta}$ は不偏推定量 Unbiasedness

II）係数 $\hat{\beta}$ は一致推定量 Consistency

III） s^2 は σ^2 の不偏推定量

IV）係数 $\hat{\beta}$ は正規分布にしたがう

Ⅰ）不偏推定量(Unbiasedness)

係数の推定量は、平均すれば、真の値そのものに一致する。すなわち、

$$E(\hat{\beta}) = \beta$$

の関係を、以下のプログラムでは、繰り返しを用いて係数の推定量を何度も計算し、それを平均した値を割り出しています。上の関係が成り立つならば、繰り返しの平均値はもともと設定した b={0.5,2,5,9};にほぼ等しくなるはずです。

プログラム

```
    new; cls;
    b={0.5,2,5,9};
```

```
    call unbiasedb(b,0,4,2000,2000);

proc unbiasedb(b,myu,var,n,times);
    local m,i,y,x;
    m=zeros(times,rows(b));
    i=1;
    do while i<=times;
        {y,x}=olssim(b,0,4,1000);
        m[i,.]=( inv(x'x)*x'y )';
        i=i+1;
    endo;
    print meanc(m);
    retp(m);
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

```
        0.49887744
         2.0000343
         4.9999703
         9.0000164
```

プログラムは、それぞれの繰り返しで出てくる列ベクトル inv(x'x)*x'y を転置することで行ベクトルにしてから、あらかじめ領域確保された m 行列の i 行目に繰り返し格納していっています。最終的に、できあがったm行列の１列目には定数項の係数が、２から４列目にはその他の係数が入っています。これをまとめて列単位で meanc でもって、平均値を出します。最終的にできあがった係数の times=2000 回の平均値が上の結果です。ほぼ、あらかじめ設定したｂの列ベクトルに近い結果になっています。

II）一致推定量(Consistency)

サンプルが十分に大きくなれば、確率極限の意味において、係数の推定量は真の値に近づいていきます。すなわち、

$$p\lim(\hat{\beta}) = \beta$$

という関係がサンプル数が増加していくにしたがって近づいていくというものです。以下のプログラムでは、サンプル数を singular にならない範囲での小さなサンプル数（ここでは start=20）から N=2000 に至るにしたがって、OLS 係数が徐々に設定した b の値に近づいていく様子をグラフ化したものです。

プログラム

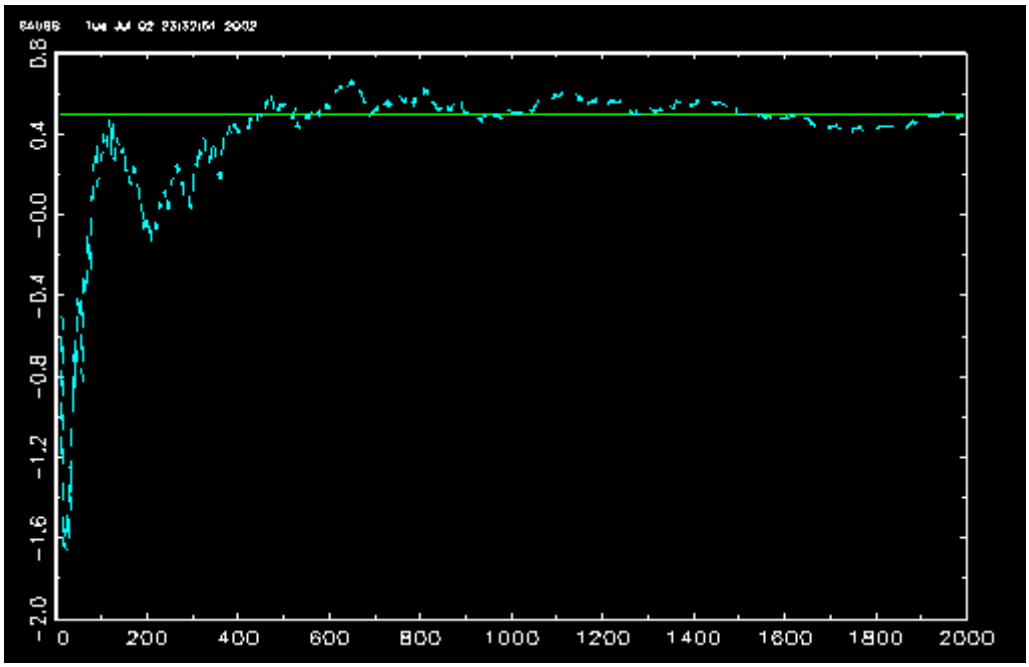
```
new; cls;
b={0.5,2,5,9};
call consistency(b,0,4,2000,1);
call consistency(b,0,4,2000,2);
call consistency(b,0,4,2000,3);
call consistency(b,0,4,2000,4);

proc consistency(b,myu,var,n,j);
    local y,x,start,m,i,y1,x1;
    {y,x}=olssim(b,myu,var,n);
    start=20;
    m=zeros(n-start+1,rows(b));
    i=start;
    do while i<=n;
        y1=y[1:i]; x1=x[1:i,.];
        m[i-start+1,.]=( inv(x1'x1)*x1'y1 )';
        i=i+1;
    endo;
        /* Graph of jth coefficient. */
        library pgraph;
        graphset;
        xy(seqa(10,1,n-start+1),b[j]*ones(n-start+1,1)~m[.,j]);
    retp(m);
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;
```

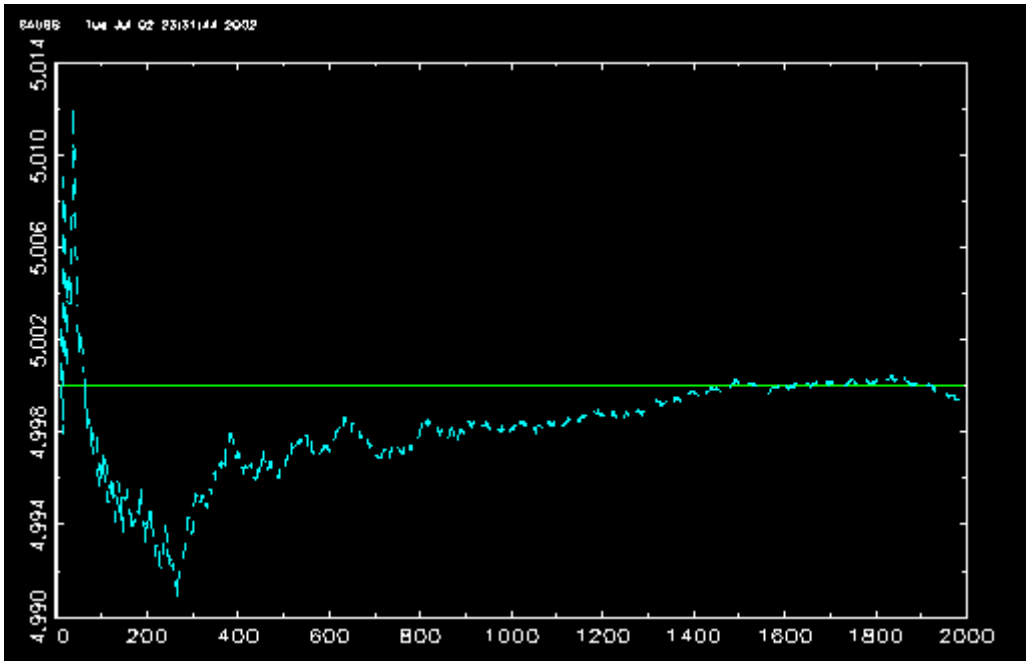
プログラムでは、olssim の乱数発生は１回だけの呼び出しで、その代わりに、サンプルの数を徐々にｉループで、start 値からｎまで増やしてやることで、その都度の係数ベクトルを得て、それをm行列に格納していっています。（なお、m 行列の最初の start－１の行は０が入ったままで空です。）このm行列のｊ番目の列を基準線のｂのｊ番目の数 b[j]とともにｘｙプロットしたものが、以下のグラフになります。

ここでは、最初の係数（定数項の係数）と第３番目の係数のみを示しておきます。どちらのグラフも初期には大幅にｂの値から乖離しているけれども、徐々にサンプルの大きさが増すにつれてｂの値に近づいていく様子がわかると思います。なお、乱数の発生状況によって、初期のぶれは基準線の下から始まることもありますし、上からはじまることもあります。近づいていくのも上からと下からとそれぞれのケースがあります。

グラフ表示（第１係数[定数項]）



グラフ表示（第３係数）



III）s^2 は σ^2 の不偏推定量

推定量 $s^2 = \dfrac{\sum \hat{e}^2}{n-k}$ は、分散 σ^2 の不偏推定量である。すなわち、推定量 s^2 を平均してやると、σ^2 になることを意味する。いま、ここで σ^2 は var=4 とわかっているので、（Ｉ）で行なったように、繰り返しによって、s^2 をたくさん求めてやって、それを平均する。プログラムでは、その値が var=4 にほぼ同じになるかを確かめる。

プログラム

```
new; cls;
b={0.5,2,5,9};
call unbiaseds2(b,0,4,2000,2000);


proc unbiaseds2(b,myu,var,n,times);
    local k,m,i,y,x,b1,e;
    k=rows(b);
    m=zeros(times,1);
    i=1;
    do while i<=times;
        {y,x}=olssim(b,myu,var,n);
        b1=inv(x'x)*x'y;
        e=y-x*b1;
```

```
        m[i]=(e'e/(n-k));
        i=i+1;
    endo;
    print "E(s2)=" meanc(m);
    retp(m);
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

```
E(s2)=          4.0005805
```

プログラムでは、ｉループで times 回 s2 を求める作業をし、m 列ベクトルのｉ番目にそれぞれの値が格納されるようにし、最後にm列ベクトルの平均をとってやる。なお、mはここでは行列ではない（列が２以上ない）ので、m[i,.]ではなく、m[i]としてやっても差し支えはない。結果は、ほぼ４となり、var=4 にきわめて近い値になっていることがわかる。

IV) 係数推定量は正規分布にしたがう

最後に、係数の推定量は、正規分布にしたがうことを、グラフで視覚的に確かめるとともに、(残差ベースではなくて) 変数ベースの Jarque-Bera Test によって見てみよう。プログラムでは、（Ｉ）で行なったのと同様に times 回繰り返しにより係数のm行列に格納し、今度はこれを平均してしまうのではなくて、これをもとにヒストグラムを描き、正規性のテストをする。

プログラム

```
    new; cls;
    b={0.5,2,5,9};
    call normality(b,0,4,2000,2000,0.05,1);
    call normality(b,0,4,2000,2000,0.05,2);
```

```
    call normality(b,0,4,2000,2000,0.05,3);
    call normality(b,0,4,2000,2000,0.05,4);

proc(0)=normality(b,myu,var,n,times,pp,j);
    local m,i,y,x,e,skew,kurtosis,jb,p;
    local mx,m1,m2,m3,m4,m5;
    m=zeros(times,rows(b));
    i=1;
    do while i<=times;
        {y,x}=olssim(b,myu,var,n);
        m[i,.]=( inv(x'x)*x'y )';
        i=i+1;
    endo;
        library pgraph;
        graphset;
        hist(m[.,j],19);
    mx=m[.,j];
    m1=mx-meanc(mx);
    m2=sumc(m1^2)/times;
    m3=sumc(m1^3)/times;
    m4=sumc(m1^4)/times;
    m5=sqrt(m2^3);
    skew=m3/m5;
    kurtosis=m4/(m2^2);
    print/lz "Coefficient" j;
    print "Skewness=" skew;
    print "Kurtosis=" 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;
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;
```

なお、上のプログラムではm行列を求めるところまでは以前と同じですが、そのあと hist でヒストグラムを作成し、変数ベースの skew(nees)と kurtosis を求め、それをもとに正規性のテストを行なっています。なお、skewness はデータが左右対称な分布であれば、ほぼ０となり、右に偏っていれば正の値、左に偏っていれば負の値になります。また、kurtosis は正規分布であれば、ほぼ３になり、上方に偏っていれば３よりも大きく、下方に偏っていれば３よりも小さくなります。以下の結果では、どの係数も、Skewness はほぼ０に近い値で、kurtosis も３に近い値を示しています。このことから、ほぼ左右対称な分布かつ、正規分布であることがわかります。また、Jarque-Bera Test もすべての係数について H0 が棄却されないので、正規性が保たれていると言えます。

画面表示

```
Coefficient 1
Skewness=      -0.058937833
Kurtosis=         2.9961596
Jarque-Bera Normality Test (RESULT:Not reject H0)
X2( 2 )=         1.1591185
P-value=         0.56014520
Coefficient 2
Skewness=      -0.090859910
Kurtosis=         3.0069174
Jarque-Bera Normality Test (RESULT:Not reject H0)
X2( 2 )=         2.7558285
P-value=         0.25210383
Coefficient 3
Skewness=      -0.071787610
Kurtosis=         2.9105974
Jarque-Bera Normality Test (RESULT:Not reject H0)
```

X2(2)=　　2.3838897

P-value=　　0.30363017

Coefficient 4

Skewness=　　0.016047369

Kurtosis=　　2.9985931

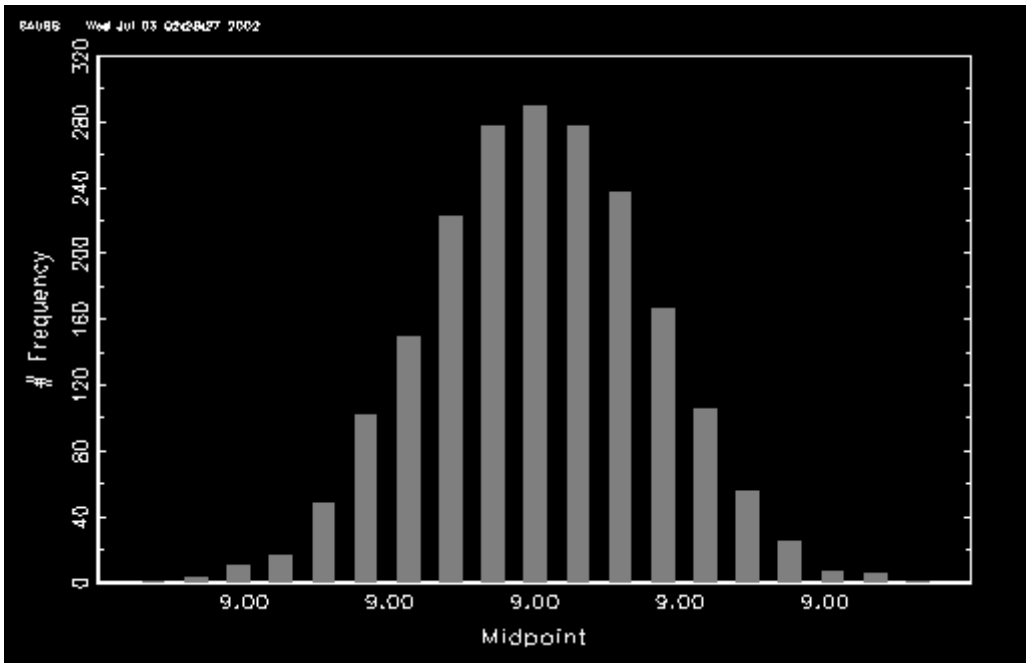
Jarque-Bera Normality Test (RESULT:Not reject H0)

X2(2)=　　0.086004293

P-value=　　0.95790933

以下のグラフも、b=９を中心に、きれいな正規分布になっていることがわかります。

グラフ表示（第４係数）



Confidence Interval(信頼区間)

ここでは、平均などの Point Estimate(点推定)ではなくて、Interval Estimate(区間推定)として、OLS 係数β's に対しての信頼区間を得るプログラムを考えます。すなわち、

$$\hat{\beta} - t_c se(\hat{\beta}) < \beta < \hat{\beta} + t_c se(\hat{\beta})$$

を計算します。SE は OLS によって各係数について求まります。t_cは自由度 n－k で pp までの compliment のインバースマッピングの値を用います。90%の信頼区間を得たい場合には、1－0.9 の２分の１の 0.05 までのｔ分布の compliment について考えます。プログラムをするには、OLS 乱数を生成する proc を一回だけ呼び出して、それをもとに通常通り OLS を行ない SE を列のまま求め、t_cに一括してかけ合わせて interval の増減分を列ベクトルとして計算します。これをβの OLS 推定量と足し引きするわけです。

プログラム

```
    new; cls;
    b={0.5,2,5,9};
    call interval(b,0,4,2000,0.05);

proc(0)=interval(b,myu,var,n,pp);
    local y,x,k,b1,e,s2hat,varb,se,interval;
    if pp>=0.5;
        errorlog "ERROR: Parameter pp must be less than 0.5.";
        retp;
    endif;
    {y,x}=olssim(b,myu,var,n);
    k=cols(x);
    b1=inv(x'x)*x'y;
    e=y-x*b1;
    s2hat=e'e/(n-k);
    varb=s2hat*inv(x'x);
    se=sqrt(diag(varb));
    interval=cdftci(pp,n-k).*se;
    print/rz (1-2*pp)*100 "% Confidence Interval";
    print "          beta                    +-";;
    print/rd b1~interval~(b-interval)~(b+interval);
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

```
                90 % Confidence Interval
      beta              +-
      0.38929733        0.22202046        0.27797954        0.72202046
      2.00104766        0.00245941        1.99754059        2.00245941
      5.00053664        0.00245080        4.99754920        5.00245080
      8.99930777        0.00248227        8.99751773        9.00248227
```

上のプログラムは(1－2×pp)×100%の OLS 係数βの信頼区間を求めるプログラムです。パラメータ pp に 0.05 を入れると、90%の信頼区間が得られます。乱数シミュレーション olssim でｙとｘの値をもとめて（ｘには１の列も含まれます)、それを呼び出して、通常の OLS を行列で解いて、係数の分散を求め、それに t 値をかけることによって、係数βからのプラスマイナスの幅を計算して、それらによって信頼区間を求めています。

上の結果は、１度限りの場合の s2 と係数の SE から求めた計算上の信頼区間でした。今度は、繰り返しを用いて、上下位 pp×100%の値を並べ替えを用いて勘定してみましょう。

プログラム

```
new; cls;
b={0.5,2,5,9};
call cinterval(b,0,4,2000,2000,0.05);

proc(0)=cinterval(b,myu,var,n,times,pp);
    local m,i,y,x,j,k;
    if pp>=0.5;
        errorlog "ERROR: Parameter pp must be less than 0.5.";
        retp;
    endif;
    m=zeros(times,rows(b));
    i=1;
    do while i<=times;
        {y,x}=olssim(b,myu,var,n);
        m[i,.]=( inv(x'x)*x'y )';
        i=i+1;
    endo;
    k=cols(x);
    j=1;
    do while j<=k;
```

```
        m[.,j]=sortc(m[.,j],1);
        j=j+1;
    endo;
    print/rz (1-2*pp)*100 "% Confidence Interval";
    print "       Lower Bound                Mean        Upper Bound";;
    print/rd m[round(pp*times),.]'~meanc(m)~m[round((1-pp)*times).,]';
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

```
                90 % Confidence Interval
     Lower Bound              Mean       Upper Bound
      0.27197562        0.50294993        0.73379577
      1.99736954        1.99998149        2.00262762
      4.99751501        4.99995101        5.00253395
      8.99740922        8.99999488        9.00249587
```

上の結果は、2000 回繰り返して、それぞれの係数の下限と上限を平均値（中位値ではありません）をはさんで表示させたものです。ｊループでそれぞれの列について昇順にソートしてやって、その下限の pp×times 番目の数値と上限の(pp－1)×times 番目の数値を平均値をはさんで水平方向にマージしたものを表示しています。ここでは下限と上限はそれぞれ行ベクトルですので、列ベクトルに変換するため転置をさせてからマージしています。結果は、１行目の定数項については多少の違いが見られますが、この並び替えによる順位値での信頼区間は、前の１回限りの OLS によるｔ値と SE による計算で求めた信頼区間にほぼ一致した結果になっています。

単位根τ分布の百分位点の計算(ドリフトなしトレンドなしのケース)

以下では、上で行なったようなサンプルの下位 pp×times 番目の値を得る方法で、単位根テストで用いられる DF テストのτの Critical Value の概算をしてみます。ここでは、前

章で扱った Wiener Process のプログラムを用いてランダムウオークにしたがう変数Xを生成しておいて、これをもとにドリフトなしトレンドなしのシンプルな DF 検定のτを求めます。これを繰り返して、そのτのサンプル分布における下位 pp×100%の値を得る方法で正式な方法ではありませんが、τの critical value の概算を計算します。

プログラム

```
new; cls;
n={25,50,100,250,500,1000};
pp={0.01,0.025,0.05,0.10,0.90,0.95,0.975,0.99};
i=1;
do while i<=rows(n);
    print/lz "n=" n[i];
    j=1;
    do while j<=rows(pp);
        /* Replace timesmean=1 to some number to be accurate. */
        /* You could set 1000, but it would take some hours each. */
        print/rz pp[j]~dftau(n[i],10000,1,pp[j]);
        j=j+1;
    endo;
    i=i+1;
endo;

proc dftau(n,times,timesmean,pp);
    local m,j,tau,i,x,dx,x1,k,b,e,s2hat,varb,se;
    m=zeros(timesmean,1);
    j=1;
    do while j<=timesmean;
        tau=zeros(times,1);
        i=1;
        do while i<=times;
            x=wiener(n,n); @ Setting dt=n/n=1. @
            dx=x[2:n]-x[1:n-1];
            x1=x[1:n-1]; /* ~ones(n-1,1) */
            k=cols(x1);
            b=inv(x1'x1)*x1'dx;
            e=dx-x1*b;
```

```
            s2hat=e'e/(n-k);
            varb=s2hat*inv(x1'x1);
            se=sqrt(diag(varb));
            tau[i]=b[1]/se[1];
            i=i+1;
        endo;
        tau=sortc(tau,1);
        m[j]=tau[pp*times];
        j=j+1;
    endo;
    /* print meanc(m); */
    retp(meanc(m));
endp;

proc wiener(T,n);
    local dt,x,dw,i;
    dt=T/n;
    x=zeros(n,1);
    dw=sqrt(dt)*rndn(n,1);
    x[1]=dw[1];
    i=2;
    do while i<=n;
        x[i]=x[i-1]+dw[i];
        i=i+1;
    endo;
    retp(x);
endp;
```

画面表示

n= 25

0.01	-2.7024621
0.025	-2.2842307
0.05	-1.9740058
0.1	-1.6337536
0.9	0.93393406
0.95	1.3540192

0.975	1.7114508
0.99	2.1792244

n= 50

0.01	-2.6438181
0.025	-2.2166492
0.05	-1.9635877
0.1	-1.6512659
0.9	0.94424331
0.95	1.302312
0.975	1.6816693
0.99	2.1519462

n= 100

0.01	-2.5728659
0.025	-2.2855906
0.05	-1.957277
0.1	-1.6378256
0.9	0.91552379
0.95	1.2789765
0.975	1.6592704
0.99	2.0334148

（　　以下、n=250 から n=1000 までの表示については省略　　）

プログラムでは、一番下の階には、wiener の proc を置いてランダムウオークにしたがうＸを生成しています。２階部分の dftau では、dX を従属変数側に、X_{t-1} を独立変数側にとって、DF テストのτを OLS のｔ値を計算する要領で計算します。これをｉループで times 回繰り返して、τのサンプル分布を列ベクトル tau に格納します。これを昇順にソートし、その下位 pp×times 番目の値を得ます。ｊループでは、これを timesmean 回繰り返してｍベクトルに格納して、より正確な critical value を得るために critical value の平均を m ベクトルを平均することによって計算します。上の計算では１に設定してあって、この平均は得ていません。1000 程度にすればより正確になりますが、各行の計算に数十分から数時間かかります。結果は、テイルの外側に行けばいくほど若干の違いはありますが、ほぼ Dickey のテーブルに近い値を得ます。（注）GAUSS 4.0 以降では、途中計算なしで計算終了後一括して表示がなされるようです。なお、プログラムを stop させると、その時までに出された結果が 3.6 以前と同様に出されます。3.6 以前では１行ずつ計算結果が出ます。