5.9　モンテカルロシミュレーション　Power 検出力　　ver.0.1

通常、統計学の検定でも計量経済学の回帰を基本に置いた検定でも、帰無仮説が真と仮定して、そのテストの統計量をもとにしてある有意水準にもとづいて仮説を棄却する、または、棄却しない判定を行なう。この帰無仮説が真であるもとで、仮説を棄却するケースを Type I Error と呼ぶ。通常のテイルの外側の棄却領域に入る確率にこれは一致する。さらにこれに加えて帰無仮説が偽である仮説を別に考える（この仮説は前の真の帰無仮説に一致する場合もあるし離れてずれている場合もある)。この偽の帰無仮説を棄却しないケースを Type II Error と呼ぶ。ここで、帰無仮説が真であるところの分布を止めておいて、すなわち、Type I Error の確率を一定として、帰無仮説が偽であるところの分布をもとの帰無仮説が真であるところの分布を平行移動したものとして、その偽の帰無仮説を棄却しないような Type II Error を小さくするようにすることになります。ここで、検出力 Power を

１‐(Type II Error の確率)＝Power

と定義します。グラフ上は、帰無仮説が真であるところの分布の右側にずれて帰無仮説が偽であるところの分布があるとするならば、この Power の入る領域は、帰無仮説が真であるところの分布の右側の棄却領域の critical value よりも右側かつ、帰無仮説が偽の領域になります。帰無仮説が偽であるところの分布がちょうど帰無仮説が真であるところの分布に一致するとき、POWER は帰無仮説が真であるところの分布の棄却領域に入る確率に一致します。両側テストであれば、棄却域は両側にあって、有意水準にこの確率は一致します。帰無仮説が偽であるところの分布を真のものから右にずらして離していくにしたがって、この Power の領域は大きくなります。十分に離れてしまうと、その確率は１になります。両側テストであれば、このずらしていく方向を左側にしても、同様に、Power はだんだん大きくなり、十分に離れると１になります。

理論上の Power 検出力

まずは、理論上の分布をもとにした Power を計算しグラフにしてみます。

標準正規分布のケース

ここでは、分散が１の正規分布、すなわち、標準正規分布にしたがうテストを考えて、帰無仮説を次のように

H0:　μ=0

H1:　μ=μ1

として、H0 の仮説を真としておいて、偽の H1 の仮説のμ1 の値を左右にずらしていくことによって、その有意水準αが 0.05 の場合の Power を求めてみます。

プログラム

```
new; cls;
```

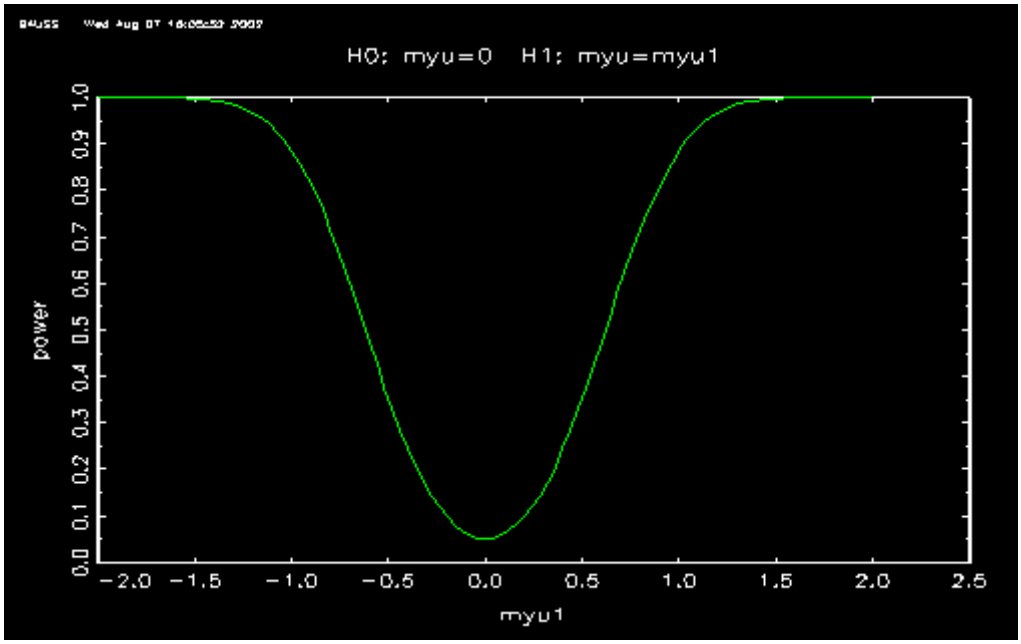
```
call powertestn(10,0,0.05);

proc powertestn(n,myu,alpha);
    local se,zval,low,up,power,i,mean,lowerz,upperz,beta;
    se=1/sqrt(n);
    zval=cdfni(1-alpha/2);
    low=myu-zval*se;
    up=myu+zval*se;
    power=zeros(101,1);
    i=0;
    do while i<=100;
        mean=myu-2+0.04*i;
        lowerz=(low-mean)/se;
        upperz=(up-mean)/se;
        beta=cdfn(upperz)-cdfn(lowerz);
        power[i+1]=1-beta;
        i=i+1;
    endo;
        library pgraph;
        graphset;
        title("H0: myu=0    H1: myu=myu1");
        ylabel("power"); xlabel("myu1");
        xy(seqa(myu-2,0.04,101),power);
    retp(power);
endp;
```

上のプログラムでは、まずse=1/sqrt(n)で分散が１とするときの上限下限を求める際に使うSEをsqrt(1)を（すなわち１を）sqrt(n)で割ることで計算しておきます。これを、標準正規分布のCDFのインバースマッピングを求めるcdfniに確率値を入れることによって、Z値をもとめ、これに上のSEをかけたものを真のμの値（ここでは０）から引くことによって、low=myu-zval*se;およびup=myu+zval*se;とすることで下限と上限を計算します。なお、この場合、両側テストを想定しますので、αを２で割ったものを１から引いたcomplimentの確率値をcdfniに代入してZ値を求めます。今、－２から２までの区間を100等分にしてやって、最初も含めて101の目盛での値を偽の時のμ1としてやってずらしていきます。おのために、まずゼロで領域を確保しています。ここでは真のμは０ですから、そこからのμ1までの相対距離をmeanとしてやって、mean=myu-2+0.04*i;によって、－２から0.04ずつｉが０から100まで動かしてやります。上限下限をlowerz=(low-mean)/se;および

upperz=(up-mean)/se;としてSEでstandardizedした相対位置で計算してやってから、beta=cdfn(upperz)-cdfn(lowerz);によって、Type II Error の確率を理論上の標準正規分布の CDF によって求めます。１からこれを引いた値が相対距離 mean 時の Power になります。これをｉを 1 つずつずらすごとに格納していきます。ただし、GAUSS には[0]というインデックスは仕様上存在しませんから、１つずらして[i+1]をインデックスにして、ｉが０から 100 までに対して、インテックスは１から 101 まで動くことになります。こうしてそれぞれの相対距離 mean 値に対して計算された Power の 101×１の列ベクトルを、横軸を偽の方のμ1 の値に、縦軸を Power にして描かせます。

グラフ表示(ｎ＝10, μ＝0)



上では、μ＝０の真の方の帰無仮説を止めておいて、μ＝μ1 の偽の方をずらせてやって、その 101 の目盛に対応する値をそれぞれ計算した Power をプロットしています。よく見かけるようなおわん型の形をしています。真と偽の分布がちょうど一致するときに、底の部分は有意水準に一致します。ここでは 0.05 に一致しています。偽の方のμ1 が十分に真の方のμ＝０から離れていると、Power は１になります。このことは冒頭の説明のとおりになっています。

次に、データの個数ｎを変えてやると、Power が変化することをグラフで見てみます。

プログラム

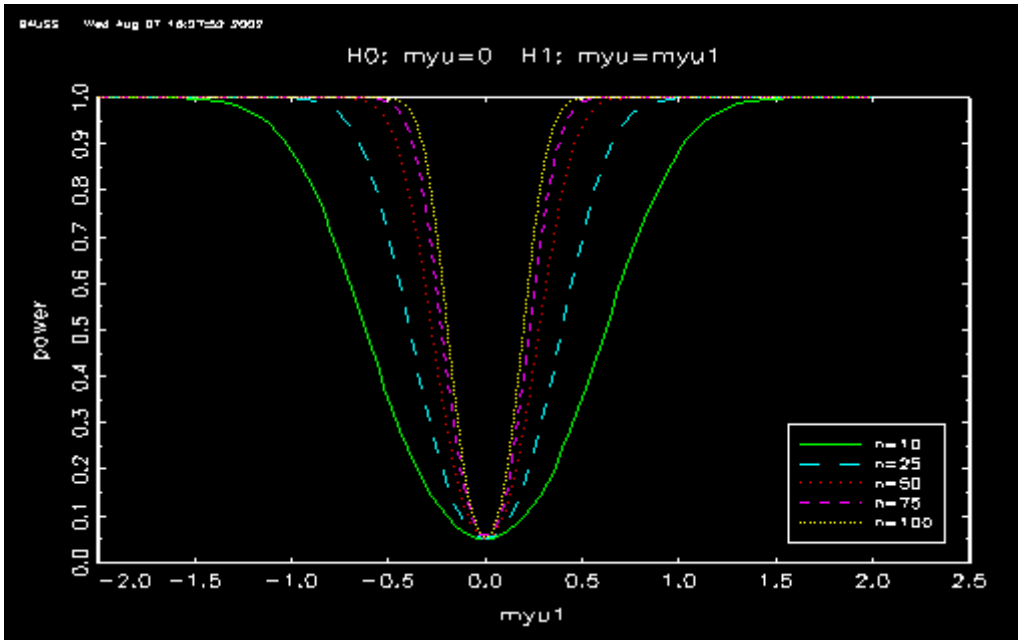
```
new; cls;
call powern(0,0.05);

proc powern(myu,alpha);
    local ndata,se,zval,low,up,power,i,j,n,mean,lowerz,upperz,beta;
```

```
    ndata={10,25,50,75,100};
    power=zeros(101,rows(ndata));
    j=1;
    do while j<=rows(ndata);
        n=ndata[j];
        se=1/sqrt(n);
        zval=cdfni(1-alpha/2);
        low=myu-zval*se;
        up=myu+zval*se;
        i=0;
        do while i<=100;
            mean=myu-2+0.04*i;
            lowerz=(low-mean)/se;
            upperz=(up-mean)/se;
            beta=cdfn(upperz)-cdfn(lowerz);
            power[i+1,j]=1-beta;
            i=i+1;
        endo;
        j=j+1;
    endo;
        library pgraph;
        graphset;
        title("H0: myu=0    H1: myu=myu1");
        ylabel("power"); xlabel("myu1");
        _plegctl=1;
        _plegstr="n=10¥000n=25¥000n=50¥000n=75¥000n=100";
        xy(seqa(myu-2,0.04,101),power);
    retp(power);
endp;
```

プログラムでは、ndata={10,25,50,75,100};としてデータ数ｎをあらかじめ設定してやってから、５つのｎの場合についてぞれぞれ計算をしますから、今までは101×１であった領域設定を、power=zeros(101,rows(ndata));として101×５のpowerのデータの入る領域をまず確保しておきます。これにｊループで、ｊ番目のndataの値n=ndata[j]に対して、５つのケースについて前のPower計算と同じことを行わせています。

グラフ表示



上のように、データ数ｎが大きくなるにしたがって、Power のおわん状の形は狭いものになり、すこしずれるだけで、よりはやく Power が１になることがわかります。

　今度は、ｎを固定してやって、有意水準αを変化させてやります。

プログラム

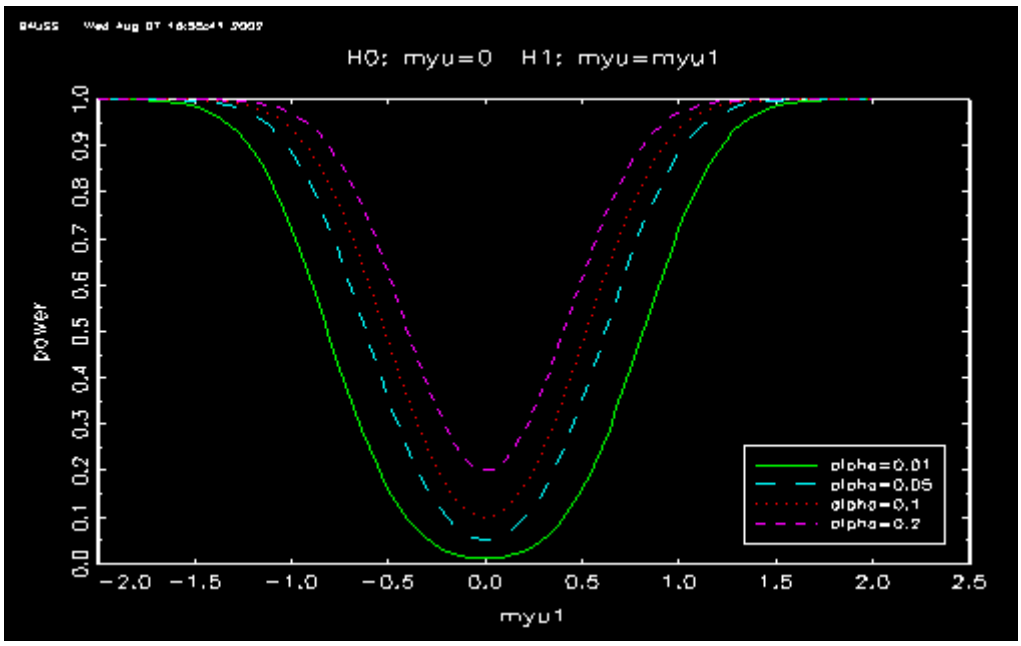
```
new; cls;
call powera(10,0);


proc powera(n,myu);
    local alphadata,se,zval,low,up,power,i,j,n,mean,lowerz,upperz,beta;
    alphadata={0.01,0.05,0.1,0.2};
    power=zeros(101,rows(alphadata));
    j=1;
    do while j<=rows(alphadata);
        se=1/sqrt(n);
        zval=cdfni(1-alphadata[j]/2);
        low=myu-zval*se;
        up=myu+zval*se;
        i=0;
        do while i<=100;
            mean=myu-2+0.04*i;
            lowerz=(low-mean)/se;
```

```
            upperz=(up-mean)/se;
            beta=cdfn(upperz)-cdfn(lowerz);
            power[i+1,j]=1-beta;
            i=i+1;
        endo;
        j=j+1;
    endo;
        library pgraph;
        graphset;
        title("H0: myu=0    H1: myu=myu1");
        ylabel("power"); xlabel("myu1");
        _plegctl=1;
        _plegstr="alpha=0.01¥000alpha=0.05¥000alpha=0.1¥000alpha=0.2";
        xy(seqa(myu-2,0.04,101),power);
    retp(power);
endp;
```

ここでは、データ数ｎが変化するのと全く同じ方法で、alphadata={0.01,0.05,0.1,0.2};として分析するαの組を設定してやっておいて、power=zeros(101,rows(alphadata));としてやって、101×４の領域を確保した上で、ｊループでそれぞれのαのPowerを計算しています。

グラフ表示



グラフからわかるように、Power は、偽のパラメータμ１やデータ数ｎに依存するばかりで

なく、有意水準αにも当然のことながら依存します。有意水準αが小さいとそれだけ外側に深くおわん状の形がなります。有意水準αは、この底の部分の最小値に一致することは言うまでもありません。αが小さいということは、真の分布のテイルの棄却 critical value が大きいことを意味しているので、なかなか１にはならないということを表しています。

正規分布のケース

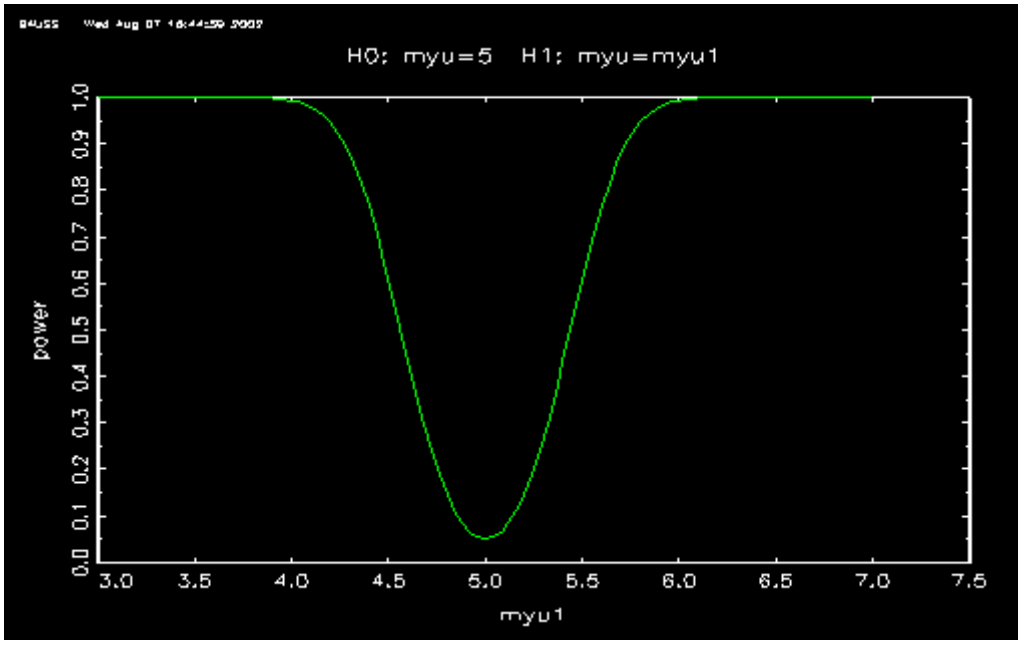
今度は、分散が必ずしも１ではない正規分布の場合について、これまでのことと同じことをしてみましょう。変更点は、分子が１ではなくて se=sqrt(s2)/sqrt(n);とするだけです。

プログラム

```
new; cls;
call powertest(10,5,0.5,0.05);

proc powertest(n,myu,s2,alpha);
    local se,zval,low,up,power,i,mean,lowerz,upperz,beta;
    se=sqrt(s2)/sqrt(n);
    zval=cdfni(1-alpha/2);
    low=myu-zval*se;
    up=myu+zval*se;
    power=zeros(101,1);
    i=0;
    do while i<=100;
        mean=myu-2+0.04*i;
        lowerz=(low-mean)/se;
        upperz=(up-mean)/se;
        beta=cdfn(upperz)-cdfn(lowerz);
        power[i+1]=1-beta;
        i=i+1;
    endo;
        library pgraph;
        graphset;
        title("H0: myu=5    H1: myu=myu1");
        ylabel("power"); xlabel("myu1");
        xy(seqa(myu-2,0.04,101),power);
    retp(power);
endp;
```

グラフ表示(n＝10, μ＝5, σ^2＝0.5)



上のグラフでは、μ＝５にしたものに対して、σ^2＝0.5 となるより分散が小さいケースについて計算したものです。μは０ではなくても仮説間の相対距離を計算する上のプログラムでは同様になります。グラフは、分散が１のものよりも若干狭い形になっています。

　このことをグラフで表すために、分散を変えてやると次のようになります。

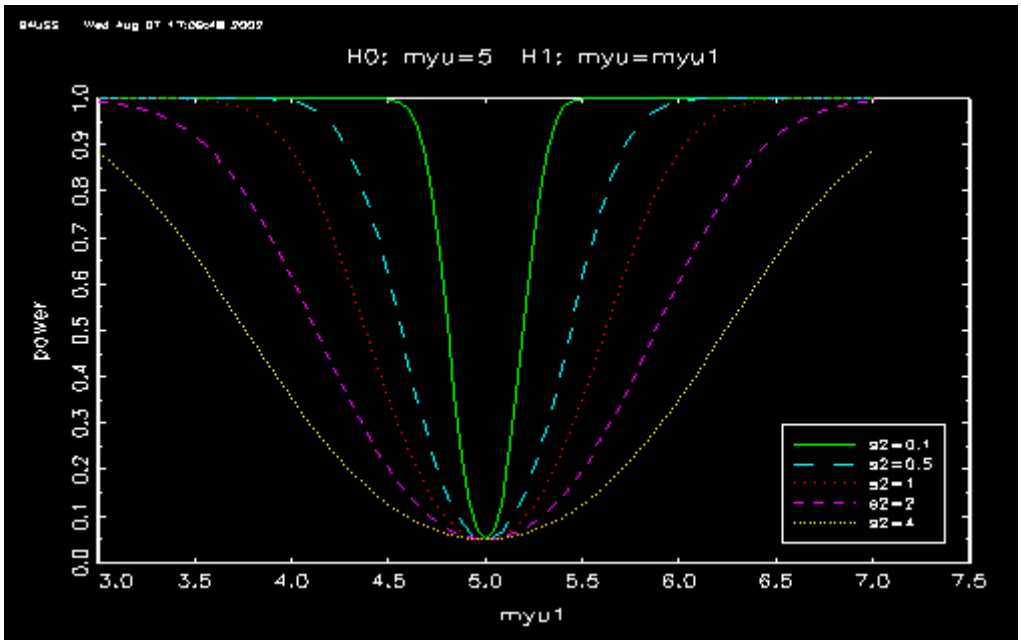
プログラム

```
new; cls;
call powers2(10,5,0.05);

proc powers2(n,myu,alpha);
    local s2data,s2,se,zval,low,up,power,i,j,mean,lowerz,upperz,beta;
    s2data={0.1,0.5,1,2,4};
    power=zeros(101,rows(s2data));
    j=1;
    do while j<=rows(s2data);
        s2=s2data[j];
        se=sqrt(s2)/sqrt(n);
        zval=cdfni(1-alpha/2);
        low=myu-zval*se;
        up=myu+zval*se;
        i=0;
```

```
        do while i<=100;
            mean=myu-2+0.04*i;
            lowerz=(low-mean)/se;
            upperz=(up-mean)/se;
            beta=cdfn(upperz)-cdfn(lowerz);
            power[i+1,j]=1-beta;
            i=i+1;
        endo;
        j=j+1;
    endo;
        library pgraph;
        graphset;
        title("H0: myu=5    H1: myu=myu1");
        ylabel("power"); xlabel("myu1");
        _plegctl=1;
        _plegstr="s2=0.1¥000s2=0.5¥000s2=1¥000s2=2¥000s2=4";
        xy(seqa(myu-2,0.04,101),power);
    retp(power);
endp;
```

グラフ表示



分散が大きければ、おわん状の形もより開いた形になり、１になかなか到達しないようになります。分散が大きい場合、critical value も大きくなることからしても明らかです。

シミュレーションによる Power 検出力

以下では、理論上の決まった Power の値ではなく、シミュレーションによって乱数過程によって若干変ってくるシミュレーションによる Power の計算方法を示します。まずは、一番簡単な場合の真と偽の仮説が一致している場合について、シミュレーションを何度か繰り返すことによって、Power が有意水準の値にほぼ等しくなることを確認します。すなわち、以下のプログラムでは、毎回n＝10 個のデータに対して、標準正規分布にしたがうとして、仮説

H0: μ=0

H1: μ=μ1=0

に対して、有意水準α＝0.05 に対して、毎回 meanc(x)/sqrt(1/n)を計算してやってから、zval=cdfni(1-alpha/2)で計算される Z 値のプラスマイナスの値を超えるケースをカウントしてやって、それがどれだけの割合で発生するのかをプログラムで求めています。すなわち、これがシミュレーションによる Power の計算になります。これを以下の計算では１万回繰り返してやって、critical value を超える割合を計算しています。

プログラム

```
new; cls;
print/rz powernsim(10,0,0,0.05,10000);

proc powernsim(n,myu0,myu1,alpha,times);
    local zval,z,i,x,count,power;
    zval=cdfni(1-alpha/2);
    z=zeros(times,1);
    i=1;
    do while i<=times;
        x=rndn(n,1)+(myu1-myu0);
        z[i]=meanc(x)/sqrt(1/n);
        i=i+1;
    endo;
    count=( (z.>zval) .or (z.<-zval) );
    power=sumc(count)/times;
    retp(power);
endp;
```

画面表示

```
          0.0504
```

上のように、この場合の Power の値は、設定された有意水準 0.05 に非常に近い値になっています。シミュレーションの回数を増やせば増やすほど、両者は一致してきます。

今度は、偽の方のμ1の値を理論値の計算と同じようにずらしていくことでその地点でのPower の値をシミュレーションで求めます。上の真と偽の仮説が一致する場合を少し変えて、x=rndn(n,1)+(myu1-myu0);によって、相対的位置関係を計算する方法で簡易的に計算してみます。これを z[i]=meanc(x)/sqrt(1/n);として、プラスマイナスZ値を超えるケースをカウントする方法で、上の場合と同じように計算します。同様に、理論値の計算も以前と同じように計算できるように procedure にしておきます。これを冒頭で、‐2から2までの 0.1 目盛の始点を含んだ 41 区間について、powernsim(10,0,j,0.05,100)およびpowern(10,0,j,0.05)をμ1の値をｊとして変更することで計算してやって、これらを水平方向にマージして、ｙという行列のｉ行目の値として計算結果を代入していきます。これを１から 41 まで 41 区間についてそれぞれ計算し、最後に横軸のμ1の値とともに、縦軸にこの理論値とシミュレーション値の両者を同時にプロットさせます。

プログラム

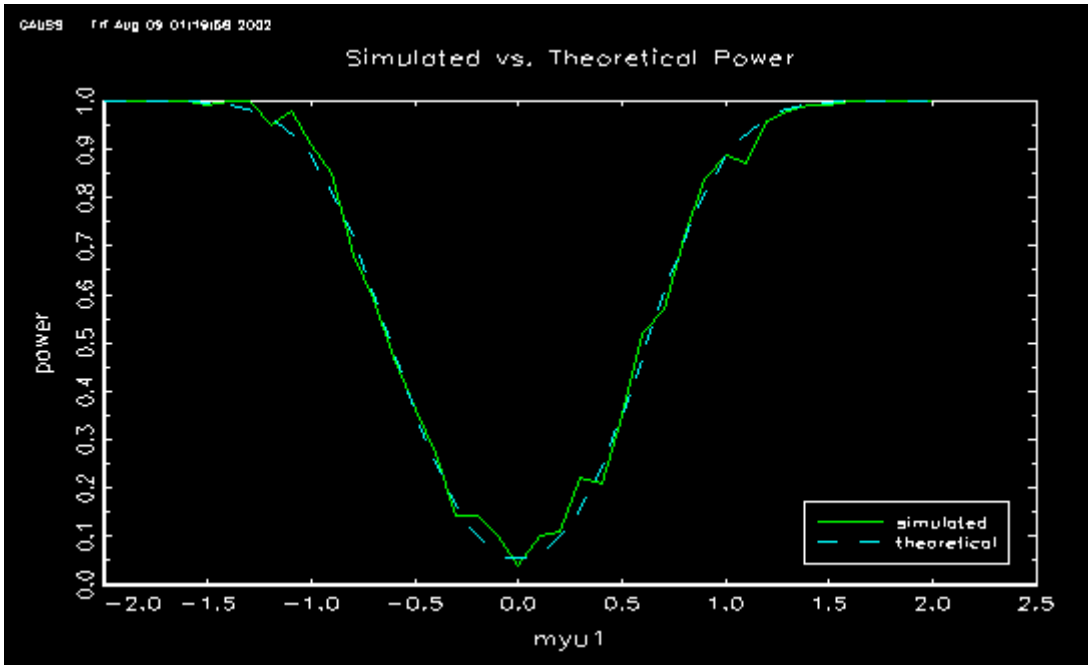
```
new; cls;
rndseed 911;
library pgraph;
graphset;
y=zeros(41,2); start=-2;
i=1; j=start;
do while i<=41;
    y[i,.]=powernsim(10,0,j,0.05,100)~powern(10,0,j,0.05);
    i=i+1; j=j+0.1;
endo;
print/rd seqa(start,0.1,41)~y;
title("Simulated vs. Theoretical Power");
ylabel("power"); xlabel("myu1");
_plegctl=1;
_plegstr="simulated¥000theoretical";
xy(seqa(start,0.1,41),y);

proc powern(n,myu0,myu1,alpha);
    local se,zval,low,up,lowerz,upperz,beta,power;
    se=1/sqrt(n);
    zval=cdfni(1-alpha/2);
    low=myu0-zval*se;
    up=myu0+zval*se;
    lowerz=(low-myu1)/se;
```

```
    upperz=(up-myu1)/se;
    beta=cdfn(upperz)-cdfn(lowerz);
    power=1-beta;
    retp(power);
endp;

proc powernsim(n,myu0,myu1,alpha,times);
    local zval,z,i,x,count,power;
    zval=cdfni(1-alpha/2);
    z=zeros(times,1);
    i=1;
    do while i<=times;
        x=rndn(n,1)+(myu1-myu0);
        z[i]=meanc(x)/sqrt(1/n);
        i=i+1;
    endo;
    count=( (z.>zval) .or (z.<-zval) );
    power=sumc(count)/times;
    retp(power);
endp;
```

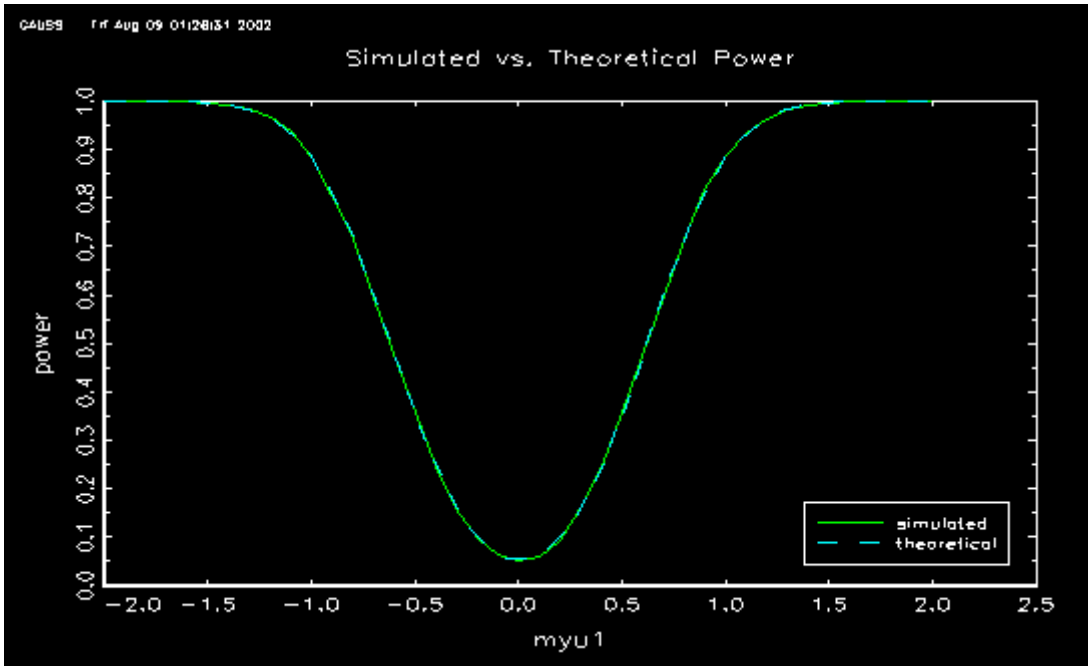
グラフ表示(n＝10,　H0: μ=0　H1: μ=myu1　100 times)



上のように 100 回程度のシミュレーションでは理論値よりもジグザグした形になります。

上のプログラムの y[i,.]=powernsim(10,0,j,0.05,100)~powern(10,0,j,0.05);のところの 100 を 10000 に変更して同じ計算をやると次のようなグラフが得られます。

グラフ表示(10000 times)



より大きな領域を必要とするので Light 版では環境によって実行できないかもしれませんが、繰り返しの数を 10000 回にしてやると、上のグラフのように、理論上の Power とシミュレーション値はほぼ一致します。

同じことを分散が１ではないケースについて計算してみます。以下の例では、分散は４で、プログラムでは x=rndn(n,1)+(myu1-myu0)/sqrt(s2);として簡易的に、真と偽の仮説の相対的位置を standardized する方法をとることにします。シミュレーションのほかの部分は上の分散１のケースと同じになります。

プログラム

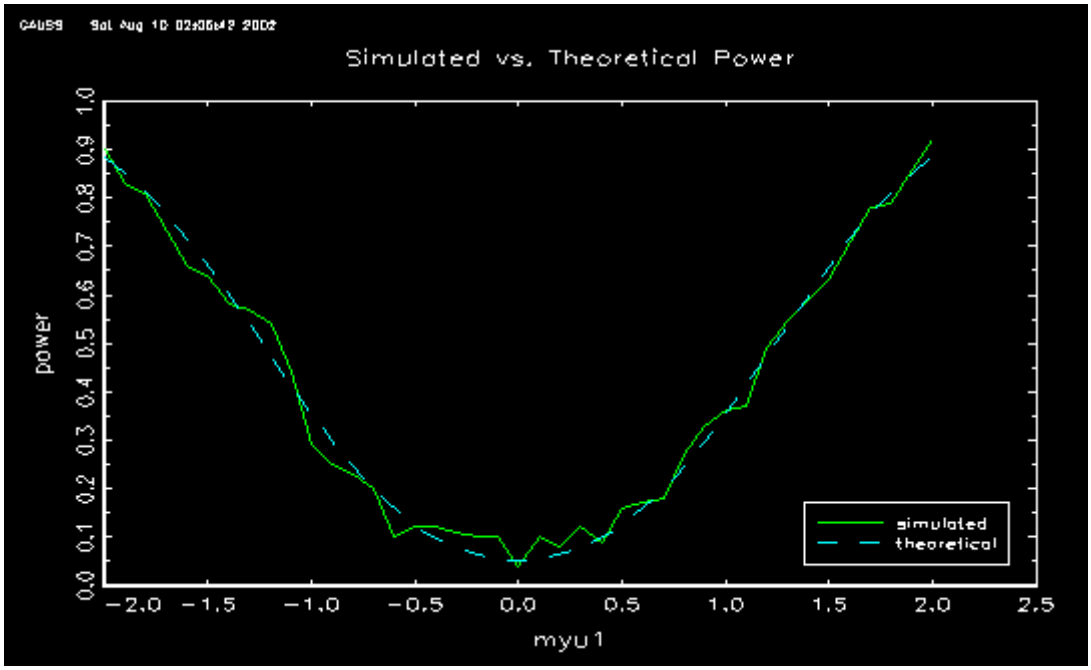
```
new; cls;
rndseed 911;
library pgraph;
graphset;
y=zeros(41,2); start=-2;
i=1; j=start;
do while i<=41;
    y[i,.]=powernsim(10,0,j,4,0.05,100)~powern(10,0,j,4,0.05);
    i=i+1; j=j+0.1;
endo;
print/rd seqa(start,0.1,41)~y;
```

```
title("Simulated vs. Theoretical Power");
ylabel("power"); xlabel("myu1");
_plegctl=1;
_plegstr="simulated¥000theoretical";
xy(seqa(start,0.1,41),y);

proc powern(n,myu0,myu1,s2,alpha);
    local se,zval,low,up,lowerz,upperz,beta,power;
    se=sqrt(s2)/sqrt(n);
    zval=cdfni(1-alpha/2);
    low=myu0-zval*se;
    up=myu0+zval*se;
    lowerz=(low-myu1)/se;
    upperz=(up-myu1)/se;
    beta=cdfn(upperz)-cdfn(lowerz);
    power=1-beta;
    retp(power);
endp;

proc powernsim(n,myu0,myu1,s2,alpha,times);
    local zval,z,i,x,count,power;
    zval=cdfni(1-alpha/2);
    z=zeros(times,1);
    i=1;
    do while i<=times;
        x=rndn(n,1)+(myu1-myu0)/sqrt(s2);
        z[i]=meanc(x)/sqrt(1/n);
        i=i+1;
    endo;
    count=( (z.>zval) .or (z.<-zval) );
    power=sumc(count)/times;
    retp(power);
endp;
```

グラフ表示(分散４のより広がった Power グラフ　n=100)



上のように、分散４の場合は、分散１の以前のケースよりも広がってなかなか１にはならないグラフになります。なお、もう少し分析の区間を広げてやると１になります。100 回のシミュレーションでは以前と同様に理論値とは違ってジグザグな形が見られます。作業領域が許す限り times の数を増してやって、例えば、10000 回などにすると理論値とシミュレーション値の両者は一致してきます。

OLS 回帰のｔ-テストの Power

これまでは、統計の分野で使われる正規分布にもとづくテストをstandardizedして理論値とシミュレーション値を計算しましたが、今度は、OLS回帰で使われるｔ分布にもとずく係数が有意であるかをみるｔテストを考えます。すなわち、係数βi＝0を真の帰無仮説を固定しておいて、βiの偽の仮説をその値をずらしていくことによってPowerの計算をしようというものです。具体的に下の例では、定数項と１説明変数のOLSモデルを考えてやって、冒頭で、ｙ＝Ｘβ＋uとなるように乱数設定します。これをもとに、tpowerというprocedure内で、定数項も含んだith番目の変数について、有意水準αのもと、times回シミュレーションをして、理論値とともにグラフ表示します。真の帰無仮説μ＝0はこの場合固定されていますから動かす必要はありません。偽の方のμ1をここでは適当に－５から５までの間を考え、0.1刻みにします。区間は101個ありますが、作業領域の関係から100個にしてあります。まず、外側のｉループでtimesのその回ごとの乱数項uを変化させ設定します。そのそれぞれの乱数項に対して、内側のｊループでは、係数βのith番目の係数をprocedure内部の冒頭で計算されたｂの値とその番目の係数だけ－5から５まで動かすそれぞれの偽の方の値に置き換えます。そうした上の再びOLS計算をしてith番目のｔ値を求めます。こ

れを自由度n-kで与えられるcrit=cdftci(pp/2,n-k)の計算で得られたｔ分布のcritical valueと比べて、両側テストで、それを超えるｔ値のものをカウントして、変数countのｉ行ｊ列目に１を代入します。そうでなければ、領域設定の０の入ったままになります。一方、そのプログラムの間に挿入された、d=myu1[j]/se[ith];ではnoncentralityパラメータを偽の仮説のパラメータをstandardizedすることによって求めた上で、critical valueを所与として

power[i,j]=cdftnc(-crit,n-k,d)+(1-cdftnc(crit,n-k,d));

によって、自由度n-kのｔ分布にもとづく両側テストを念頭に置いたPowerを計算します。なお、後半の項だけにすると片側テストになります。その場合はcritical valueを計算するのにppを２で割らない数になります。この場合は、両側テストなので、上の２項になります。これにより、ｊループの内側で、理論値とシミュレーション値の両方を求めて、ｉループの外側で、spower=meanc(count);とすることで、カウントが１になった割合を列ごとに（ｊごとにずらしたものに対応する形で）シミュレーションPowerを計算すると同時に、power=meanc(power);で列ごとの理論Powerの平均をとってやっています。これらを横軸の偽のパラメータμ1とともに、縦軸に同時にプロットさせます。

プログラム

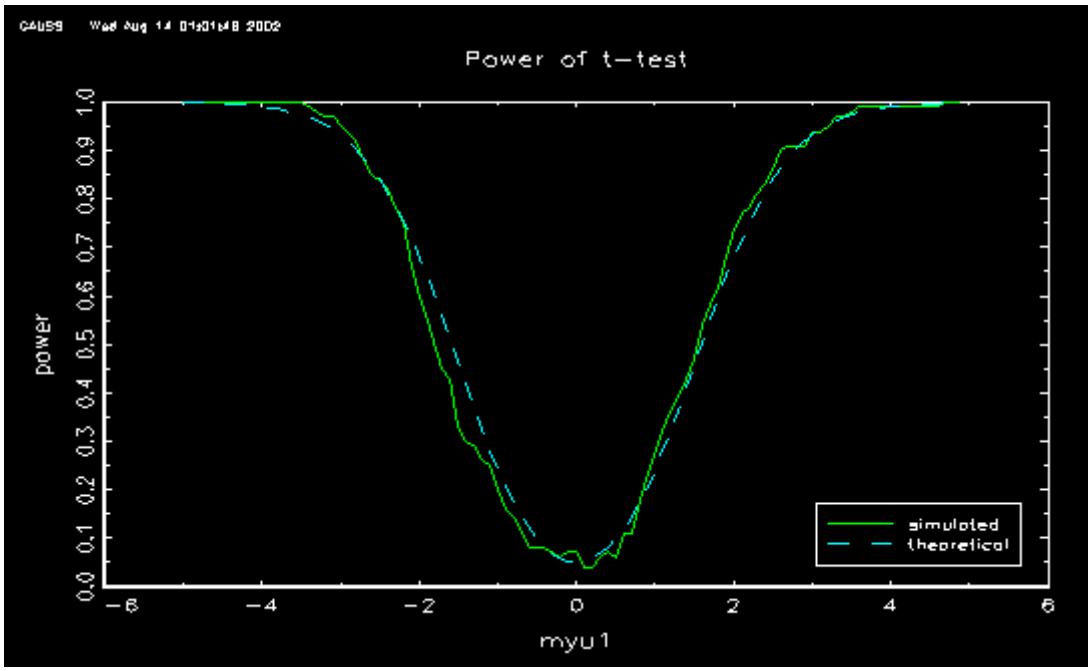
```
new; cls;
b={0.5,2}; x=ones(20,1)~rndu(20,1); u=rndn(20,1);
y=x*b+u;
call tpower(y,x,2,0.05,100);

proc(0)=tpower(y,x,ith,pp,times);
local n,k,crit,power,count,spower,myu1,i,j,d,u,b,bhat,e,s2hat,varb,se,t;
n=rows(x); k=cols(x);
b=inv(x'x)*x'y;
crit=cdftci(pp/2,n-k);
power=zeros(times,100); count=zeros(times,100); myu1=seqa(-5,0.1,100);
i=1;
do while i<=times;
    u=rndn(n,1);
    j=1;
    do while j<=100;
        b[ith]=myu1[j];
        y=x*b+u;
        bhat=inv(x'x)*x'y;
        e=y-x*bhat;
        s2hat=e'e/(n-k);
```

```
            varb=s2hat*inv(x'x);
            se=sqrt(diag(varb));
            d=myu1[j]/se[ith];
            power[i,j]=cdftnc(-crit,n-k,d)+(1-cdftnc(crit,n-k,d));
            t=(bhat[ith]-0)/se[ith];
            if (t<-crit) or (t>crit);
                count[i,j]= 1;
            endif;
        j=j+1;
        endo;
    i=i+1;
    endo;
    spower=meanc(count);
    power=meanc(power);
        library pgraph;
        graphset;
        title("Power of t-test");
        ylabel("power"); xlabel("myu1");
        _plegctl=1;
        _plegstr="simulated¥000theoretical";
    xy(myu1,spower~power);
endp;
```

なお、上のプログラムでは、noncentrality パラメータを計算するのに一定した SE を使うのではなくて、その都度 OLS 計算された SE を使っています。また、グラフでは、これまでどおり実際の偽のパラメータのシフトを横軸にとっています。

グラフ表示



グラフでは、シミュレーション値がずいぶんとジグザグした形になります。乱数設定とシードによっては、凹凸が激しくなったり理論値から離れたりします。

F-type Test の Power

ｔテストと同じことをＦ分布にしたがう制限式１個のテストで行なってみると次のようになります。原理的には、ｔテストと同じ真の帰無仮説があるとして、これを１本の制限式と考えて、プログラミングの節の線形制約とテストの章で扱ったように、

$$F = \frac{(RSS_R - RSS_U)/(K-1)}{RSS_U/(N-K)}$$

を制約した場合の RSS とそうでない元の RSS から計算して、これをもとにＦ分布にもとづく計算をしてやります。なお、仮説が同じである限りは、ｔテストの Power と形は同じになります。計算に使う分布が異なることになります。ループの内側で上のＦ値の計算が加わるだけで基本的にはｔテストのプログラムと同じになります。また、この場合の noncentrality パラメータは同じく偽の係数を standardized した形を使うことにします。

プログラム

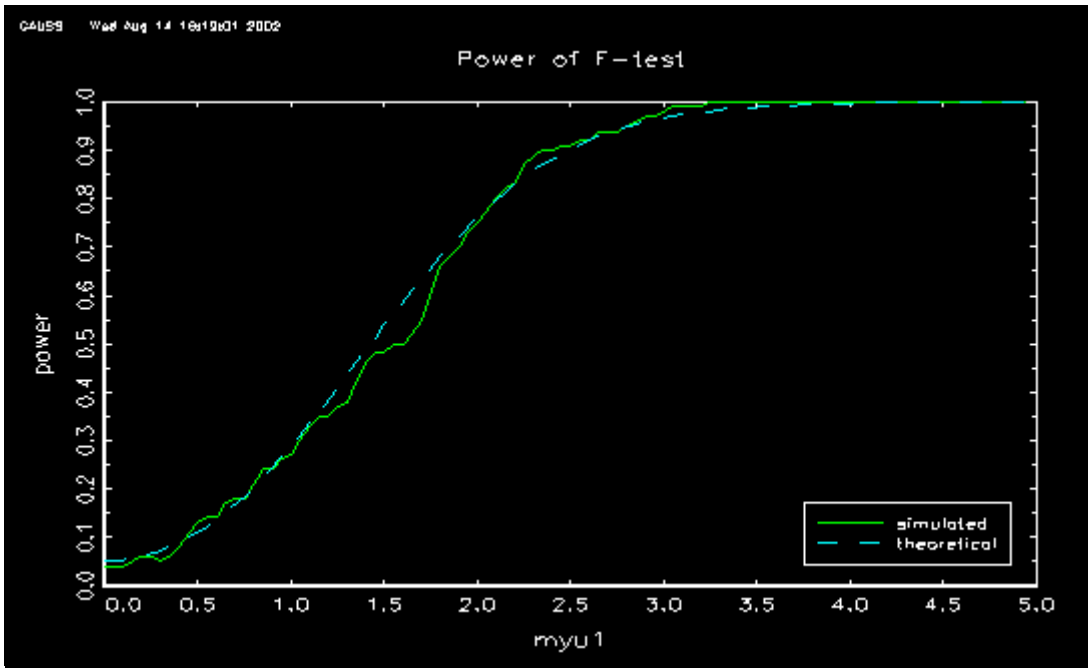
```
new; cls;
b={0.5,2};
x=ones(20,1)~rndu(20,1); u=rndn(20,1);
y=x*b+u;
call fpower(y,x,2,0.05,100);
```

```
proc(0)=fpower(y,x,ith,pp,times);
    local n,k,crit,power,count,spower,myu1,i,j,d,u,b,bhat,e,s2hat,varb,se;
    local R,q,br,rssr,rssu,f;
    n=rows(x); k=cols(x);
    b=inv(x'x)*x'y;
    crit=cdffci(pp,1,n-k);
    power=zeros(times,100); count=zeros(times,100); myu1=seqa(0,0.05,100);
    i=1;
    do while i<=times;
        u=rndn(n,1);
        j=1;
        do while j<=100;
            b[ith]=myu1[j];
            y=x*b+u;
            bhat=inv(x'x)*x'y;
            e=y-x*bhat;
            s2hat=e'e/(n-k);
            varb=s2hat*inv(x'x);
            se=sqrt(diag(varb));
            R=zeros(1,k); R[ith]=1; q=0;
            br=bhat+inv(x'x)*R'inv(R*inv(x'x)*R')*(q-R*bhat);
            rssr=(y-x*br)'(y-x*br);
            rssu=(y-x*bhat)'(y-x*bhat);
            f= ((rssr-rssu)/1) / (rssu/(n-k));
            d=myu1[j]/se[ith];
            power[i,j]=1-cdffnc(crit,1,n-k,d);
            if f>crit;
                count[i,j]= 1;
            endif;
        j=j+1;
        endo;
    i=i+1;
    endo;
    spower=meanc(count);
    power=meanc(power);
```

```
        library pgraph;
        graphset;
        title("Power of F-test");
        ylabel("power"); xlabel("myu1");
        _plegctl=1;
        _plegstr="simulated¥000theoretical";
    xy(myu1,spower~power);
endp;

proc cdffci(alpha, df1, df2);
    local crit,epsilon;
    crit=1;
    epsilon=1;
    do while epsilon>1e-6;
        epsilon=alpha-cdffc(crit,df1,df2);
        crit=crit-epsilon;
        epsilon=abs(epsilon);
    endo;
    retp(crit);
endp;
```

グラフ表示



なお、上の計算では、F分布の CDF の critical value の計算に GAUSS には cdfnci に相当

するものがないので、末尾に簡単な procedure を付け加えています。αと分子の自由度および分母の自由度の３つのインプットからＦ分布の critical value をリターンで返すようにしてあります。上では、この場合Ｆ分布にしたがう片側テストなので

power[i,j]=1-cdffnc(crit,1,n-k,d);

を用いて Power を計算しています。また、Ｆ分布の片側右方向だけを考えるようにするため便宜上、偽の仮説μ１は０からスタートさせています。

Type Ⅰ Error と Empirical Power(Non-Nested Tests の例)

最初から真の仮説と偽の仮説の２つを立てやすいテストである非入れ子テストの場合の Type I Error の確率と Power (すなわち、１－Type II Error の確率)を計算してみます。ここでは、プログラミングの節のモデル選択とテストの章で扱ったプログラムをベースにして、若干変更して Type I Error の確率と Power の値を返すようにした上で、Ｊテスト、JA テスト、それに COX テストの３つを同一乱数設定およびシードのもとに比較します。この場合、２つある仮説

ｙ＝Ｘβ＋u

ｙ＝Ｚβ＋u　　(ただし、ＸおよびＺは定数項を含む)

のうち、前者を真に設定するために、技術的に、所与のＸのもとで乱数項 u から計算してＸβ＋u から得られる値をｙとして、後者のｙ＝Ｚβ＋u を偽の仮説とします。これによりそれぞれのテストにおいて、第 1 のモデル（Ｚのモデルなにがしか考えた推定値を加えたもの）が通常通り棄却されるケースを Type Ⅰ Error と考えて、これまでと同じようにそうなる回数をシミュレーションで求め割合を出します。また、第２のモデルを偽としていますからこれを棄却しない領域が Type Ⅱ Errror ですから、この場合、その反対のこれを棄却する領域に入る確率が Power と見ることができます。すなわち、第２のモデルが棄却される回数を同一乱数設定の同一シードのシミュレーションで求め割合を出します。これが、この場合の Power になります。

プログラム

```
new; cls;
rndseed 1000;
b={0.5,1,2};
x=ones(20,1)~rndu(20,2);
z=ones(20,1)~rndu(20,2);
call modelpower(b,x,z,0.05,1000);

proc(0)=modelpower(b,x,z,pp,times);
    local j,ja,cox,i,u,y,type1,power;
    j=zeros(times,2); ja=zeros(times,2); cox=zeros(times,2);
```

```
    i=1;
    do while i<=times;
        u=rndn(rows(x),1);
        y=x*b+u; /* Suppose y=Xb+u is true and y=Zb+u is false. */
        {type1,power}=jtest(y,x,z,0.05);
            j[i,.]=type1~power;
        {type1,power}=jatest(y,x,z,0.05);
            ja[i,.]=type1~power;
        {type1,power}=coxtest(y,x,z,0.05);
            cox[i,.]=type1~power;
        i=i+1;
    endo;
    print /rz times "times simulation";
    print "                          Type I                  Power";
    print/rz "    J Test" meanc(j)';
    print/rz " JA Test" meanc(ja)';
    print/rz "COX Test" meanc(cox)';
    print "(Suppose y=Xb+u is true and y=Zb+u is false)";
endp;

proc(2)=jtest(y,x,z,pp);
    local n,bx,bz,yhatx,yhatz,k1,k2,b1,b2,e1,e2,type1,power;
    local s2hat1,s2hat2,varb1,varb2,se1,se2,t1,t2;
    if rows(x)/=rows(z);
        errorlog "ERROR: Each row does not match.";
        retp;
    endif;
    n=rows(x);
    bx=inv(x'x)*x'y; yhatx=x*bx;
    bz=inv(z'z)*z'y; yhatz=z*bz;
    x=x~yhatz; z=z~yhatx;
    k1=cols(x); k2=cols(z);
/* Equation 1 */
    b1=inv(x'x)*x'y;
    e1=y-x*b1;
    s2hat1=e1'e1/(n-k1);
```

```
    varb1=s2hat1*inv(x'x);
    se1=diag(sqrt(varb1));
    t1=b1./se1;
    if t1[k1]<cdftci(1-pp/2,n-k1) or t1[k1]>cdftci(pp/2,n-k1);
        type1=1;
    else;
        type1=0;
    endif;
/* Equation2 */
    b2=inv(z'z)*z'y;
    e2=y-z*b2;
    s2hat2=e2'e2/(n-k2);
    varb2=s2hat2*inv(z'z);
    se2=diag(sqrt(varb2));
    t2=b2./se2;
    if t2[k2]<cdftci(1-pp/2,n-k1) or t2[k2]>cdftci(pp/2,n-k1);
        power=1;
    else;
        power=0;
    endif;
    retp(type1,power);
endp;

proc(2)=jatest(y,x,z,pp);
    local n,bx,bz,yhatx,yhatz,k1,k2,b1,b2,e1,e2,type1,power;
    local s2hat1,s2hat2,varb1,varb2,se1,se2,t1,t2,b,bb;
    if rows(x)/=rows(z);
        errorlog "ERROR: Each row does not match.";
        retp;
    endif;
    n=rows(x);
    b=inv(x'x)*x'y; bb=inv(z'z)*z'y;
    bx=inv(x'x)*x'(z*bb); yhatx=x*bx;
    bz=inv(z'z)*z'(x*b); yhatz=z*bz;
    x=x~yhatz; z=z~yhatx;
    k1=cols(x); k2=cols(z);
```

```
/* Equation 1 */
    b1=inv(x'x)*x'y;
    e1=y-x*b1;
    s2hat1=e1'e1/(n-k1);
    varb1=s2hat1*inv(x'x);
    se1=diag(sqrt(varb1));
    t1=b1./se1;
    if t1[k1]<cdftci(1-pp/2,n-k1) or t1[k1]>cdftci(pp/2,n-k1);
        type1=1;
    else;
        type1=0;
    endif;
/* Equation2 */
    b2=inv(z'z)*z'y;
    e2=y-z*b2;
    s2hat2=e2'e2/(n-k2);
    varb2=s2hat2*inv(z'z);
    se2=diag(sqrt(varb2));
    t2=b2./se2;
    if t2[k2]<cdftci(1-pp/2,n-k1) or t2[k2]>cdftci(pp/2,n-k1);
        power=1;
    else;
        power=0;
    endif;
    retp(type1,power);
endp;

proc(2)=coxtest(y,x,z,pp);
    local n,bx,bz,Mx,Mz,s2x,s2z,s2xz,q1,q2,temp,type1,power;
    n=rows(y);
/* Equation 1 */
    bx=inv(x'x)*x'y; Mx=eye(n)-x*inv(x'x)*x'; s2x=(y-x*bx)'(y-x*bx)/n;
    bz=inv(z'z)*z'y; Mz=eye(n)-z*inv(z'z)*z'; s2z=(y-z*bz)'(y-z*bz)/n;
    s2xz=s2x+bx'x'Mz*x*bx/n;
    q1=(n/2)*ln(s2z/s2xz)/sqrt((s2x/s2xz^2)*bx'x'Mz*Mx*Mz*x*bx);
    if q1<cdfni(pp/2) or q1>cdfni(1-pp/2);
```

```
        type1=1;
    else;
        type1=0;
    endif;
/* Equation 2 */
    temp=x; x=z; z=temp;
    bx=inv(x'x)*x'y; Mx=eye(n)-x*inv(x'x)*x'; s2x=(y-x*bx)'(y-x*bx)/n;
    bz=inv(z'z)*z'y; Mz=eye(n)-z*inv(z'z)*z'; s2z=(y-z*bz)'(y-z*bz)/n;
    s2xz=s2x+bx'x'Mz*x*bx/n;
    q2=(n/2)*ln(s2z/s2xz)/sqrt((s2x/s2xz^2)*bx'x'Mz*Mx*Mz*x*bx);
    if q2<cdfni(pp/2) or q2>cdfni(1-pp/2);
        power=1;
    else;
        power=0;
    endif;
    retp(type1,power);
endp;
```

上のプログラムの末尾に付け加えられているＪおよび JA、それに COX の procedure は、それぞれ t1 や t2 または q1 や q2 の統計量の計算の後に条件分岐を入れて Type I Error となる数および Power に相当するものとなる数をそれぞれ計算してリターンになるように、以前に作った Procedure を一部改造したものです。これを冒頭の procedure で１と０からなるこれらの数を合計して平均をとることで一括計算させています。

画面表示

1000 times simulation

	Type I	Power
J Test	0.108	0.774
JA Test	0.046	0.493
COX Test	0.286	0.976

(Suppose y=Xb+u is true and y=Zb+u is false)

上の設定の場合には、ほぼ常に COX テストの Power が一番大きく１に近い。次は、Ｊテストが大きく最後に JA テストの順になる。一方、Type I Error の確率は JA テストが、ほぼ常に有意水準の設定値 0.05 に近い値をとり最小値をとり、J テスト、COX テストの順で大きくなっていく。もちろん、テストの結果は設定方法や乱数シードに大きく依存します。