5.4　モンテカルロシミュレーション Maximum Likelihood ver.0.1

ここでは、時系列以外に関する Maximum Likelihood で推定をする際の乱数設定についていろいろな例を取り上げます。設定は基本的には OLS を用いてセットアップします。

Poisson 分布の設定と推定

ここでは、Likelihood 計算でもって、ポワソン分布の乱数ｙの平均は、そのパラメータλに等しいことを示します。この Log-Likelihood はベクトル形で、

LL=－λ＋ｙｌｎλ－ｌｎ(ｙ!)

となります。ただし、第１項は同じ－λの数の列とします。ほかの足し合わせる項が列であるならば、GAUSS では ones(n,1)をかけてもかけなくても同じ計算をします。なお、通常の Log-Likelihood は、上の列の合計をとって、sumc(LL)とすればできます。以下では

n	λ	→	y
1000	5		rndp(n,1,λ)

という乱数設定をします。GAUSS で直に計算する場合には Log-Likelihood を ll(b)という１値のものにします。これを例えば、grid search で最大化します。ライブラリ maxlik で Log-Likelihood を最大化する場合には、ll(b,dataset)という２値のものを最大化します。なお、ここでは列を合計する前のベクトル形（この方がうまくいく確率が経験上高まる）を最大化します。直に計算する１値のケースでは必ず列を合計した通常の Log-Likelihood でなくてはなりません。その場合、dataset の部分は、インプットとは関係なくグローバルに通します。

プログラム

```
new; cls;
n=1000; lambda=5;
dataset=rndp(n,1,lambda);
call linegrid(&ll,1,10);

proc linegrid(&ll,start,finish);
    local length,maxiter,tol,i,j,step,grid,b,temp,bmax,ll:proc;
        length=100;
        maxiter=100;
        tol=1e-8;
    bmax=0;
```

```
    i=1;
    do while i<=maxiter;
        step=(finish-start)/length;
        grid=(-1e256).*ones(length+1,1);
        b=start;
        j=1;
        do while j<=length+1;
            grid[j]=ll(b);
            j=j+1;
            b=b+step;
         endo;
         temp=start+(maxindc(grid)-1)*step;
         if abs(bmax-temp)<tol;
            break;
         endif;
         bmax=temp;
         print/rd bmax;
         start=bmax-step; finish=bmax+step;
         i=i+1;
    endo;
    print/rz "# of iterations=" i-1;
    print/rd bmax;
    retp(bmax);
endp;

proc ll(b);
    local y,logl;
    y=dataset;
    logl=-ones(n,1)*b+y*ln(b)-ln(y!);
    retp(sumc(logl));
endp;
```

画面表示

```
        4.96000000
        5.00140000
        5.00100400
        5.00099968
```

```
        5.00100000
        5.00100001
# of iterations=                6
        5.00100001
```

上のケースは、1 値の grid search で log-Likelihood を最大化したものです。これは、3.8 の最適化アルゴリズムで扱った 2 値の通常の grid search のプログラムを 1 値の 1 重ループにしたものに、

```
    temp=start+(maxindc(grid)-1)*step;
    if abs(bmax-temp)<tol;
        break;
    endif;
    bmax=temp;
```

という打ち切りのアルゴリズムを入れたものです。すなわち、start+(maxindc(grid)-1)*step という grid search で得られたその回の最高値をとりあえず temp という一時変数に格納しておき、それが前回の最高値が入っている bmax という変数と比べて、その差の絶対値があらかじめ設定しておいた tol という許容桁数を表す変数よりも小さくなったときにループから break で出ます。そうでなければ、temp の内容を bmax に格納して、次の回の最高値を求めるためにもう一度プログラムのループを回します。

プログラム[要ライブラリ Maxlik の方法]

```
new; cls;
library maxlik;
maxset;

    n=1000; lambda=5;
    dataset=rndp(n,1,lambda);
    _max_Algorithm = 5;
    start={1};
    call maxprt(maxlik(dataset,0,&ll,start));

proc ll(b);
  local y,logl;
  y=dataset;
  logl=-ones(n,1)*b+y*ln(b)-ln(y!);
  retp(logl);
endp;
```

画面表示

Mean log-likelihood -2.20623

Number of cases 1000

Covariance matrix of the parameters computed by the following method:

Inverse of computed Hessian

Parameters	Estimates	Std. err.	Est./s.e.	Prob.	Gradient
P01	5.0040	0.0707	70.739	0.0000	-0.0000

Correlation matrix of the parameters

1.000

Number of iterations 12

Minutes to convergence 0.01567

こちらは、maxlik ですっきりとした形で計算しています。要するに、どのような最適化の計算法をとるにしても、ll の部分の proc を最大化しているのです。上の結果は、前回のものと同様に、ほぼ設定したスケラーλの値５に近い推定結果が得られます。すなわち、ｙの平均値は poisson パラメータλにほぼ等しいことが示されているのです。

<u>従属変数がカウントイベントの Poisson Regression の設定と推定</u>

上の Poisson 分布の応用で、ここでは、λ（常に正の数）のところにＸβを指数変換した値を持ってきます。すなわち、

$$\lambda = \exp(X\beta)$$

を Poisson 分布の Log-Likelihood に代入します。ただし、ベクトル形で

$$LL = -\lambda + y.*\ln(\lambda) - \ln(y!)$$

のように第２項を要素対要素のかけ算にする必要があります。ベクトル形ではない一般形の Log-Likelihood は、この LL を列の合計をとって、sumc(LL)とすればできます。上のλは今度はベクトルであって、各要素は同じではありません。ここでは、そのパラメータと乱数の設定を

n	β_0	β_1	β_2	X_1	X_2
1000	0.1	0.2	-0.1	[0,2]	[0,2]

なお、上の[0,2]は０から２までの範囲の一様分布を表します。指数変換しますからβの符号は任意なのですが、！の階乗項が Log-Likelihood にありますから、その絶対値での大きさはあまり大きくできません。撹乱項 u は、この場合、ベクトル exp(Ｘβ)の各要素によっ

てその都度 Poisson 乱数を考えますから、そこから誤差が生じていて、特に必要ではありません。（なお、一般の Maximum Likelihood 計算には撹乱項 u は必須です。）これまでと同じように、log-Likelihood の proc ll を作成して、これを最大化します。

プログラム

```
new; cls;
n=1000;
x=2*rndu(n,2);
x=ones(n,1)~x;
b={0.1,0.2,-0.1};
lambda=exp(x*b);
dataset=rndp(n,1,lambda)~x;
start={0,0,0};
print newton(&ll,start);

proc newton(&ll,beta);
    local step,maxiter,tol,i,gvector,hmatrix,ll:proc;
        step=1;
        maxiter=1e+3;
        tol=1e-3;
    i=1;
    do while i<=maxiter;
        print ll(beta)~beta';
        gvector=gradp(&ll,beta);
        if abs(gvector) < tol;
            break;
        endif;
        hmatrix = hessp(&ll,beta);
        beta = beta - step*gvector'/hmatrix;
    i=i+1;
    endo;
    print; print /rz "# of iterations=" i-1;
    retp(beta);
endp;

proc ll(b);
```

```
    local y,x,lambda,logl;
    y=dataset[.,1];
    x=dataset[.,2:cols(dataset)];
    lambda=exp(x*b);
    logl=-lambda+y.*ln(lambda)-ln(y!);
    retp(-sumc(logl));
endp;
```

画面表示

```
# of iterations=                    9


        0.10098193
        0.23001671
       -0.14941521
```

上のプログラムでは、dataset というｙとｘをマージしたデータをグローバルに通しおいてから、proc ll(b)の内部で、ｙとｘに分解して、その引数ｂとともに Log-Likelihood を作成します。λ=exp(Ｘβ)を先に設定しておいてから、LL＝－λ＋ｙ.*ｌｎ(λ)－ｌｎ(ｙ!)に代入してやっています。なお、grid search 以外の自分で作成したプログラムは基本的に最小値（極小値）をもとめるものですから、厳密さをきすために、最後にマイナスの符号をつけてリターンとしています。つまり、Log-Likelihood そのもののマイナス値を最小化することにより、Log-Likelihood を最大化しています。なお、実際的には、マイナスはつけなくても通常、経路が逆になるだけで、どうにか最大化はしてくれます。

プログラム

```
    new; cls;
    library maxlik;
    maxset;
    n=1000;
    x=2*rndu(n,2);
    x=ones(n,1)~x;
    b={0.1,0.2,-0.1};
    lambda=exp(x*b);
    dataset=rndp(n,1,lambda)~x;
    start={0,0,0};
    _max_Algorithm=5;
    _max_ParNames={"CONST","X1","X2"};
    call maxprt(maxlik(dataset,0,&ll,start));
```

```
proc ll(b,dataset);
    local y,x,lambda,logl;
    y=dataset[.,1];
    x=dataset[.,2:cols(dataset)];
    lambda=exp(x*b);
    logl=-lambda+y.*ln(lambda)-ln(y!);
    retp(logl);
endp;
```

画面表示

```
Mean log-likelihood         -1.39133
Number of cases      1000

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------------
CONST            0.1010        0.0780     1.295   0.0977     -0.0000
X1               0.2300        0.0501     4.590   0.0000     -0.0000
X2              -0.1494        0.0491    -3.045   0.0012      0.0000

Correlation matrix of the parameters
   1.000  -0.707  -0.624
  -0.707   1.000   0.029
  -0.624   0.029   1.000

Number of iterations    4
Minutes to convergence      0.00833
```

上では、ライブラリ maxlik を用いて ll(b,dataset)を最大化させています。こちらは、dataset に基づいた LL (β｜data)の最大化をしていることになります。２値が必要です。リターンはベクトル形のまま、そのまま返します。なぜなら、maxlik や cml などのライブラリでは通常、内部では最小化（極小化）を計算していますが、みかけは最大化をしており、入力は関数そのものを呼び出すのです。したがって、こちらにはマイナスはつけません。結果は、GAUSS そのもので解いたものもライブラリをつかったものも共に、もともとのβのベクトル設定{0.1,0.2,-0.1}に近い結果になっています。上の計算は１回かぎりのシミュレーションですが、上の計算を繰り返して平均をとってやれば、さらにもとのβの値に近くなっ

ていくはずです。

Possion Regression と Negative Binomial Regression の比較

ここでは、これまでの Possion Regression の推定の乱数設定をそのまま用いて、シードを一定にすることで、Poisson と Negative Binomial の推定の両方を同時に行ない係数比較をしてみます。

プログラム

```
new; cls;
n=1000;
x=2*rndu(n,2);
x=ones(n,1)~x;
b={0.1,0.2,-0.1};
lambda=exp(x*b);
dataset=rndp(n,1,lambda)~x;
rndseed 911;
start={0,0,0};
print "Poisson Regression";;
print newton(&ll,start);
start=start|1;
print;
print "Negative Binomial Regression";;
print newton(&nbll,start);

proc newton(&ll,beta);
    local step,maxiter,tol,i,gvector,hmatrix,ll:proc;
        step=1;
        maxiter=1e+3;
        tol=1e-3;
    i=1;
    do while i<=maxiter;
        /* print ll(beta)~beta'; */
        gvector=gradp(&ll,beta);
        if abs(gvector) < tol;
            break;
        endif;
        hmatrix = hessp(&ll,beta);
```

```
        beta = beta - step*gvector'/hmatrix;
    i=i+1;
    endo;
    print; print /rz "# of iterations=" i-1;
    retp(beta);
endp;

proc ll(b);
    local y,x,lambda,logl;
    y=dataset[.,1];
    x=dataset[.,2:cols(dataset)];
    lambda=exp(x*b);
    logl=-lambda+y.*ln(lambda)-ln(y!);
    retp(-sumc(logl));
endp;

proc nbll(b);
    local y,x,a,lambda,logl;
    y=dataset[.,1];
    x=dataset[.,2:cols(dataset)];
    a=(b[cols(x)+1])^2;
    lambda=exp(x*b[1:cols(x)]);
    logl=ln(gamma((1./a)+y))-ln(y!)-ln(gamma(a))-
                        (1./a).*ln(1+a.*lambda)+y.*ln(a.*lambda)-y.*ln(1+a.*lambda);
    retp(-sumc(logl));
endp;
```

画面表示

```
Poisson Regression
# of iterations=          10

        0.13258664
        0.21581520
       -0.12331363

Negative Binomial Regression
# of iterations=           8
```

0.13494198
0.21619450
-0.12615203
1.3915564

上の係数で、Negative Binomial の方にもう１つ付け加えられている第４パラメータは、その係数ａに相当する部分です。大雑把に言って、Poisson は平均と分散を一致させるようにしたもの、一方、Negative Binomial は平均と分散は一致させないものと言えます。上の推定では、同じシード設定で両者の比較を試みています。平均と分散を一致させない分だけ Negative Binomial の係数は若干ずれていることがわかると思います。何か定数ｋを考えて

Poisson: Mean=μ Variance=μ

Negative Binomial: Mean=μ Variance=$\mu + k\mu^2$

というふうに Negative Binomial の方は定数ｋの分だけ平均と分散がずれているとしたものを推定することになっています。

LOGIT と PROBIT の設定と推定

LOGIT と PROBIT はもととなる分布が異なるだけで、Likelihood の基本的な部分とその設定方法は共通です。OLS で言うところの従属変数側が０より大きければ従属変数には１を、それ以外ならば０となる。すなわち、

$y^*>0$ ならば $y=1$

$y^*\leqq 0$ ならば $y=0$

とします。例えば、定数項と２説明変数の場合には、

$y^*=\beta_0+\beta_1X_1+\beta_2X_2+u$

を右辺から設定してやって、その計算結果である左辺のｙ*をもとに１か０の判断をしてやることになる。その際、ある一定割合で１と０が混ざっている必要があるので（すなわちあるｙ*がある一定割合で正か非正である必要がある）、その列合計を見てサンプル数ｎと比べてやると正の個数がわかる。下の例では、

n	β_0	β_1	β_2	X_1	X_2	u
1000	10	-4	5	[0,20]	[0,20]	N(0,50)

についてシミュレーションをします。ここで [0,20]とは０から 20 までの一様乱数。以下のプログラムの冒頭部分の乱数設定では、

```
ystar=x*b+u;
y=(ystar.>0);
```

のように、計算されたｙ*の値にもとづいて、それが要素対要素で０より大きければ True

で１を括弧内で返して、それをｙに代入します。そうでなければ括弧内で０を返して、それをｙに代入する作業を１行でこなしています。条件分岐でｉ＝1 から n まで繰り返しても同じことになります。LOGIT の Log-Likelihood は、

$F(x_i\beta)=\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}=\frac{1}{1+\exp(-x_i\beta)}$ のもと　LP=yln F(Xβ)+(1-y)(1-F(Xβ))

となります。F(Xβ)の分布設定のところにロジスティック分布を用いることになります。

プログラム

```
new; cls;
n=1000;
b={10,-4,5};
x=20*rndu(n,rows(b)-1); x=ones(n,1)~x;
u=sqrt(50)*rndn(n,1);
ystar=x*b+u;
y=(ystar.>0);          /*    check by print ystar~y; and print "n(ystar>0)=" sumc(y); */

dataset=y~x;
start={1,0,0};
print newton(&ll,start);

proc newton(&ll,beta);
    local step,maxiter,tol,i,gvector,hmatrix,ll:proc;
        step=1;
        maxiter=1e+3;
        tol=1e-5;
    i=1;
    do while i<=maxiter;
        print ll(beta)~beta';
        gvector=gradp(&ll,beta);
        if abs(gvector) < tol;
            break;
        endif;
        hmatrix = hessp(&ll,beta);
        beta = beta – step*gvector'/hmatrix;
    i=i+1;
    endo;
    print; print /rz "# of iterations=" i-1;
```

```
    retp(beta);
endp;

proc ll(b);
    local y,x,cdf,logl;
    y=dataset[.,1]; x=dataset[.,2:cols(dataset)];
    cdf=1./(1+exp(-x*b));                    @ cdf=cdfn(x*b); for PROBIT instead @
    logl= y.*ln(cdf)+(1-y).*ln(1-cdf);    /* Log Likelihood */
    retp(-sumc(logl));
endp;
```

画面表示

```
# of iterations=          9

         2.2080489
        -0.94570597
         1.1985263
```

プログラム[要ライブラリ Maxlik の方法]

```
new; cls;
library maxlik;
maxset;
n=1000;
b={10,-4,5};
x=20*rndu(n,rows(b)-1); x=ones(n,1)~x;
u=sqrt(50)*rndn(n,1);
ystar=x*b+u;
y=(ystar.>0);

dataset=y~x;
start={0,0,0};
_max_Algorithm=5;
_max_ParNames={"CONST","X1","X2"};
{x,f,g,cov,retcode}=maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
```

```
    local y,x,cdf,logl;
    y=dataset[.,1]; x=dataset[.,2:cols(dataset)];
    cdf=1./(1+exp(-x*b));                  /* cdf=cdfn(x*b) for PROBIT instead. */
    logl=y.*ln(cdf)+(1-y).*ln(1-cdf);      /* Vector-component of Log-Likelihood. */
    retp(logl);
endp;
```

画面表示

```
Mean log-likelihood        -0.121925
Number of cases      1000

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------------
CONST            2.2080        0.4046     5.457   0.0000     -0.0000
X1              -0.9457        0.0928   -10.191   0.0000     -0.0000
X2               1.1985        0.1171    10.238   0.0000     -0.0000

Correlation matrix of the parameters
   1.000  -0.615   0.352
  -0.615   1.000  -0.942
   0.352  -0.942   1.000

Number of iterations    12
Minutes to convergence      0.02550
```

「rndseed 数字;」の行を乱数設定の前のどこかに挿入してシードを一定にしてやれば、上の結果はどちらもほぼ同じになるはずです。（バージョンが異なると結果は違うことはあります。また最初に run したときにはシードの設定がうまくいかないことがあります。２回以上 run してみてください。）

今度は、乱数設定も Log-Likelihood も同じですが、分布設定が違う CDFN にする Probit の場合です。すなわち、GAUSS では cdf=cdfn(x*b)と設定します。これをもとに、Log-Likelihood のベクトル形 LP=yln F(Xβ)+(1-y)(1-F(Xβ))を最大化することになります。で

すから、LOGIT のプログラムの１行を cdf=cdfn(x*b);のように変更するだけです。

プログラム

```
new; cls;
n=1000;
b={10,-4,5};
rndseed 777;
x=20*rndu(n,rows(b)-1); x=ones(n,1)~x;
u=sqrt(50)*rndn(n,1);
ystar=x*b+u;
y=(ystar.>0);

dataset=y~x;
start={0,0,0};
print newton(&ll,start);

proc newton(&ll,beta);
    local step,maxiter,tol,i,gvector,hmatrix,ll:proc;
        step=1;
        maxiter=1e+3;
        tol=1e-5;
    i=1;
    do while i<=maxiter;
        print ll(beta)~beta';
        gvector=gradp(&ll,beta);
        if abs(gvector) < tol;
            break;
        endif;
        hmatrix = hessp(&ll,beta);
        beta = beta – step*gvector'/hmatrix;
    i=i+1;
    endo;
    print; print /rz "# of iterations=" i-1;
    retp(beta);
endp;
```

```
proc ll(b);
    local y,x,cdf,logl;
    y=dataset[.,1]; x=dataset[.,2:cols(dataset)];
    cdf=cdfn(x*b);
    logl= y.*ln(cdf)+(1-y).*ln(1-cdf);    /* Log Likelihood */
    retp(-sumc(logl));
endp;
```

画面表示

```
# of iterations=                20

        1.2497693
       -0.53314075
        0.67718825
```

プログラム[要ライブラリ Maxlik の方法]

```
new; cls;
library maxlik;
maxset;
n=1000;
b={10,-4,5};
rndseed 777;
x=20*rndu(n,rows(b)-1); x=ones(n,1)~x;
u=sqrt(50)*rndn(n,1);
ystar=x*b+u;
y=(ystar.>0);

dataset=y~x;
start={0,0,0};
_max_Algorithm=5;
_max_ParNames={"CONST","X1","X2"};
{x,f,g,cov,retcode}=maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
    local y,x,cdf,logl;
    y=dataset[.,1]; x=dataset[.,2:cols(dataset)];
    cdf=cdfn(x*b);
```

```
    logl=y.*ln(cdf)+(1-y).*ln(1-cdf);      /* Vector-component of Log-Likelihood. */
    retp(logl);
endp;
```

画面表示

```
Mean log-likelihood        -0.120745
Number of cases      1000

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------------
CONST            1.2498        0.2203     5.673   0.0000       0.0000
X1              -0.5331        0.0486   -10.959   0.0000       0.0000
X2               0.6772        0.0617    10.983   0.0000       0.0000

Correlation matrix of the parameters
   1.000  -0.602   0.321
  -0.602   1.000  -0.935
   0.321  -0.935   1.000

Number of iterations     13
Minutes to convergence      0.04033
```

通常、LOGIT と PROBIT の係数の間には 1.6 倍前後の関係があるとされます。乱数生成ごとに若干の変化はありますが、1.6 倍から 2 倍弱の関係が認められます。

<u>TOBIT の設定と推定</u>

ここでは、LOGIT と PROBIT と同じ設定を用いますが、ｙ*が正のときだけ従属変数ｙとして用います。すなわち、

ｙ*＞0 ならば　ｙ＝ｙ*

ｙ*≦0 ならば　ｙ＝0

という条件にもとづいてｙベクトルを作成して、これを用いて TOBIT 推定をします。以下では、冒頭の乱数設定部分で、ｉ＝１からｎまでループを回してｙ*の値にもとづいて実際の値なのかそれとも０なのかを判断して代入しています。この場合の Log-Likelihood は、

$$LL= \Sigma_{yi=0} \ln(1-F_i)+ \Sigma_{yi>0} [-1/2 \ln(2\pi)-1/2 \ln\sigma^2-(y-x\beta)^2/2\sigma^2]$$

となります。第１項は、PROBIT の第２項のものと基本的には同じです。第２項は、線形（および一般形の）Maximum Likelihood の計算のものと同じです。これを 4.4 の章でやったのと同じくσを用いた表記法で組みたててやれば、ベクトル形で、

```
logl=(y.==0).*ln(1-cdfn(x*beta/s))+ (y.>0).*lnpdfmvn(y-x*beta,s^2);
```

となります。なお、第１項の 1-cdfn(x*beta/s)は Compliment を用いて cdfnc(x*beta/s)としても同じことです。第２項は、multivariate normal の自然対数の値を求める組込み関数 lnpdfmvn の１変数の場合を利用しています。以下では、収束しやすいライブラリ maxlik のケースについてだけ示しておきます。収束に困難を極める場合には、_max_Algorithm=2; のところの数字を変更して最適アルゴリズムをかえるか、start ベクトルを値を変更してみてください。なお、s は正の数ですからライブラリ CML で正の制約をかけてもできます。

プログラム[要ライブラリ Maxlik の方法]

```
new; cls;
library maxlik;
maxset;
n=1000;
b={10,-4,5};
x=20*rndu(n,rows(b)-1); x=ones(n,1)~x;
rndseed 777;
u=sqrt(50)*rndn(n,1);
ystar=x*b+u;
y=zeros(n,1);
i=1;
do while i<=n;
    if ystar[i]>0;
        y[i]=ystar[i];
    else;
        y[i]=0;
    endif;
    i=i+1;
endo;
/*   check by print ystar~y; print "n(ystar>0)=" sumc(y); */

dataset=y~x;
```

```
start={0,0,0,1};
_max_Algorithm=2;
_max_ParNames={"CONST","X1","X2","s"};
{x,f,g,cov,retcode}=maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
    local y,x,beta,s,logl;
    y=dataset[.,1]; x=dataset[.,2:cols(dataset)];
    beta=b[1:3];
    s=b[4];
    logl=(y.==0).*ln(1-cdfn(x*beta/s))+ (y.>0).*lnpdfmvn(y-x*beta,s^2);
    retp(logl);
endp;
```

画面表示

Mean log-likelihood -2.29944
Number of cases 1000

Covariance matrix of the parameters computed by the following method:
Inverse of computed Hessian

Parameters	Estimates	Std. err.	Est./s.e.	Prob.	Gradient
CONST	10.8451	0.6382	16.993	0.0000	0.0000
X1	-4.0785	0.0514	-79.416	0.0000	-0.0000
X2	4.9784	0.0523	95.245	0.0000	-0.0000
s	6.8934	0.1866	36.934	0.0000	-0.0000

Correlation matrix of the parameters

1.000	-0.279	-0.693	-0.070
-0.279	1.000	-0.392	-0.091
-0.693	-0.392	1.000	0.102
-0.070	-0.091	0.102	1.000

Number of iterations 44
Minutes to convergence 0.11350

BOX-COX 変換の設定

ここでは、乱数設定のみについて扱います。（推定については収束に困難が伴うので後日アップすることにします。）ここでは、

$$Y^{(\theta)}=\beta_0+\beta_1 X_1^{(\lambda)}+\beta_2 X_2^{(\lambda)}+u$$

を考えます。ここで、丸括弧内の添字は

$$Z^{(\alpha)}=\ln(Z) \quad \text{if } \alpha=0$$
$$=(Z^{\alpha}-1)/\alpha \quad \text{otherwise}$$

という変換を表します。以下では乱数設定を

n	β_0	β_1	β_2	X_1	X_2	θ	λ	u
1000	1	1	1	[5,15]	[5,15]	1	1	N(0,1)

として、これにもとづいて$\beta_0+\beta_1 X_1^{(\lambda)}+\beta_2 X_2^{(\lambda)}+u$を計算してやってから、$Y^{(\theta)}$をYにθの設定値（この場合は１）にもとづいたアルゴリズムでもどしてやります。

プログラム

```
n=1000;
b={1,1,1};
theta=1;
lambda=1;
sigma2=1;
x=ones(n,1)~(10*rndu(n,2)+5);
u=sqrt(sigma2)*rndn(n,1);
if lambda==0;
    ytheta=ones(n,1)*b[1]+ln(x[.,2:cols(x)]*b[2:rows(b)])+u;
else;
    ytheta=ones(n,1)*b[1]+((x[.,2:cols(x)]*b[2:rows(b)])^lambda-1)./lambda+u;
endif;
if theta==0;
    y=exp(ytheta);
else;
    y=(theta*ytheta+1)^(1/theta);
endif;
dataset=y~x;
```