4.4 特殊関数とライブラリ MAXLIK(2)[要 full version]　ver. 0.2

引き続きLog-Likelihood関数のプログラムの作り方とそれを最大化させるライブラリであるMAXLIKの操作方法について説明をします。今、単純な例として線形の回帰分析（もちろん、MAXLIKの活躍の場は非線形の場合についてなのですが）をいろいろな方法で行なってみます。少なくとも３つの方法で推定できるはずです。１つは逆行列を用いて正確に計算する方法、もう１つはSSEを最低にする方法、そして、本章のトッピクスであるLog-Likelihood関数を最大化する方法があります。今、データはdatafile1.txtというｙとｘについてのシリーズがDドライブにあるものとします。まずは、行列をinverseして求める通常の線形推定の方法を再掲します。

プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
y=data[2:29,1]; x=ones(28,1)~data[2:29,2];
{b,se,t,shat,df,r2}=lse(y,x);
print "y-intercept:" b[1] "          slope for x:" b[2];
print "           se:" se';
print "            t:" t';
print "Std of est :" shat;
print /rz "           df=" df;
print /rz "           R2=" r2;
proc(6)=lse(y,x);
   local b,uhat,n,k,df,s2hat,shat,varb,se,t,r2;
   b=inv(x'x)*x'y;
   uhat=y-x*b;
   n=rows(x);
   k=cols(x);
   df=n-k;
   s2hat=uhat'uhat/df;
   shat=sqrt(s2hat);
   varb=s2hat*inv(x'x);
   se=diag(sqrt(varb));
   t=b./se;
   r2=1-uhat'uhat/(y'(eye(n)-1/n*ones(n,1)*ones(n,1)')*y);
   retp(b,se,t,shat,df,r2);
endp;
```

画面表示

y-intercept:	77.795198	slope for x:	52.009829
se:	22.037400		3.8317273
t:	3.5301442		13.573468
Std of est :	52.331001		
df=	26		
R2=	0.87633124		

これは、proc(n)の章で取り上げましたので説明の繰り返しは省略します。自作関数lse(y,x)を使うのではなくて、olsという関数を使ってパッケージのような計算をすることもできますが、アメリカの大学院ではこれを使わずに計算せよと言われることが常ですから、ここではとりあげません。（もっと厳しいところでは、私が教育を受けたように、MAXLIKやCMLはおろか、QNewtonなどの高度なライブラリやコマンドを使ってはいけないと指導されることもあります。）さらに、これと同じ切片と係数を求めるのであれば、SSEを最小にするようにしても得られます。この場合にはQNewtonで計算してみましょう。関係統計量の計算はここでは省略します。今、わかりやすいように、各変数のラベルを_qn_ParNamesというグローバル変数で設定します。

プログラム

```
new; cls;
load data[29,2]=d:datafile1.txt;
data=data[2:29,.];
y=data[.,1];
x=ones(28,1)~data[.,2];

fn sse(b)=(y-x*b)'(y-x*b);
start={0,0};
_qn_ParNames={"CONST","X"};
{b,f,g,retcode}=QNewton(&sse,start);
```

画面表示

```
return code =    0
normal convergence

Value of objective function    71201.875467

Parameters    Estimates    Gradient
```

```
----------------------------------------
CONST          77.7952       -0.0001
X              52.0098       -0.0003

Number of iterations      12
Minutes to convergence    0.00000
```

なお、上のように、SSEを求めるにはe'eを計算すれば行列形式ですぐに計算されますが、ベクトル方式で計算させるのであれば、sumc(e^2)としても全く同じ結果になります。

ポイント　SSEの計算には、e'e　または　sumc(e^2)

なお、QNewtonなどの一般的な関数の最小値（最大値）を求める関数では、列を合計した値が用いられるのに対して、以下のLog-Likelihoodを最大化するMAXLIKやCMLでは、列を合計する前の段階の列のままのベクトルの形が関数として設定されることの違いを十分に理解してください。また、通常Qnewtonで呼び出されて最小化される関数は、上の例であれば、ｂというベクトル１値の関数になっているのに対して、MAXLIKやCMLではｂというベクトルのほかに、全体のdataも与えて、合計２値の関数にしなければなりません。これらの点について、しばしば、誤解が生じます。

さて、同じデータと関数形を推定するのに、Maximum Likelihoodの概念を用いて推定します。具体的には、

$$LL = -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln(\sigma^2) - \frac{(y - x\beta)'(y - x\beta)}{2\sigma^2}$$

であるから、これを前の２項をN回足し合わせて最終項を足し合わせる前の形を表せば、

$$-\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\sigma^2) - \frac{(y - x\beta)^2}{2\sigma^2}$$

という28×１の列ベクトルになります。これを用いて、MAXLIKでLLを最大化します。前の例との比較から、ここでは列が合計された本当の意味でのLog-Likelihoodが用いられているのではなくて、その前段階の各項を列ベクトルにした形がMAXLIK（およびCML）の受けつける最大化すべき関数形に設定します。２πの項はコンスタントなので、しばしば省略されて、その部分がないLog-Likelihoodを最大化するようなパラメータを求めることもありますが、基本的に答えはMaximum Log-Likelihoodの値以外は同じになるはずです。

プログラム

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

```
load data[29,2]=d:datafile1.txt;
data=data[2:29,.];
start={0,0,1};
_max_Algorithm = 2;
_max_ParNames = {"CONST","X","s"};
call maxprt(maxlik(data,0,&ll,start));

proc ll(b,data);
    local beta, s, y, x, e;
    y=data[.,1];
    x=ones(28,1)~data[.,2];
    beta=b[1:2];
    s=b[3];
    e=y-x*beta;
    retp(-1/2*ln(2*pi)-1/2*ln(s^2)-1/2*e^2/s^2);
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood         -5.33947
Number of cases      28

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

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
-------------------------------------------------------------------
CONST           77.7945       21.2356     3.663   0.0001     -0.0000
X               52.0099        3.6923    14.086   0.0000     -0.0000
s               50.4275        6.7388     7.483   0.0000      0.0000

Correlation matrix of the parameters
   1.000  -0.894  -0.000
```

```
  -0.894    1.000    0.000
  -0.000    0.000    1.000

Number of iterations     91
Minutes to convergence      0.11800
```

上のように、切片および係数ほぼ同じですが、推定式のstandard errorであるｓは行列をinverseして正確に計算した結果の52よりも小さい50となっていて、その分だけ若干の切片および係数の違いが表れています。なお、最大化のアルゴリズムに　_max_Algorithm = 2;のBFGSを採用していますが、もしこれを_max_Algorithm = 4;　のニュートン法にするとiterationは通常の回数では収束しません。ここでは注意点が２つあります。１つ目は、上の計算ではスタートベクトルを{0,0,1}としていますが、これは、最後の１に相当する部分は、Log-Likelihoodを最高にするｓを推定しているので、関数llのリターンのところの割る数にｓがあるので０がくると計算できないので、正の任意の数、たとえば、１から始めていることです。２つ目は、_max_ParNames = {"CONST","X","s"};でパラメータのラベルをつけているのですが、それぞれの名前が８文字を超えるとエラーを出すので、それ以下の文字にする必要があります。上の例は極めて単純な例ですが、計算上はかなりたいへんなLog-Likelihoodを最大化する作業をしていることが実感できると思います。また、いくつかあるオプションのアルゴリズムのうち、どれがどれだけ速いか、またiterationの数が少ないかを議論するのは無意味なことで、それらのことはベクトル形式のLog-Likelihoodの関数形およびスタート値に依存して決まってくるものです。関数形によっては、ニュートン法だけではスタート値をいろいろと変えて調整しないと最大値に到達しないケースが多く存在するので、このように_max_Algorithmで指定できるアルゴリズムがたくさんあるのです。また、関数形が複雑である場合には、グローバルに最大値であるのかを確認するために、いろいろなアルゴリズムでやってみることも必要になってきます。ただし、アルゴリズムによっては、永遠にConvergeしないケースもありますから、

```
_max_MaxTime=1e+5;
```

というiterationをする最大分数のデフォルト値を、たとえば、２とか３とかの数分にしてみてもよいかもしれません。もちろん、これまでやったように、

```
_max_MaxIters=1e+5;
```

という最大iteration数のデフォルトを1000回程度に変更しても構いません。ここでは、かなりの分数を要することから、最大分数で強制終了させる方法も用意されているのです。

特にUNIX系でプログラムをやっている方や、大人数で１つのシステムにぶら下がってGAUSS等を動かしている場合には、上のiterationの時間または回数の制限を加えるグローバル変数の設定行を加えておくことをおすすめします。

TOBITモデル

いろいろなバージョンの TOBIT モデルがあるのですが、従属変数がある条件（多くの場合、０またはそれ以上）を満たした場合のみ観測されて、それをもとに推定します。分布には、standard normal が用いられます。いま、D ドライブに datafile6.txt というファイル（応用計量経済学 II[1997]表 4-4 を利用）があるものとします。通常 TOBIT と呼ばれるものをプログラムするには、Log-Likelihood 関数を、足し合わせる前の列ベクトルで、

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

として計算します。なお、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。lnpdfmvn は multivariate のケースの ln normal の PDF を計算するもので、その第２要素に、Covariance 行列がきます。１つの normal のケースには、ここにｓを２乗したスケラーがくることになります。これを利用しています。なお、最初からｓをｓの２乗とおいて計算することもできます。そうした方がｓがマイナスの値を取らないために、計算上、0 収束しやすくなります。その場合、前の項の beta/s のところのｓは sqrt(s)に置きかえることになります。なお、上の LL の後半πやσ^2を用いて書くとすれば、この場合の Log-Likelihood 関数は、

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

となります。この後半のｙ＞０ついての項を 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(y.>0).*lnpdfmvn(y-x*beta,s^2)としているのです。上の Log-Likelihood の第１項は、PROBIT の第２項のものと同じです。第２項は、線形（および一般形の）Maximum Likelihood の計算で行なったものと同じです。この２つの場合を統合したものが、TOBIT ということになります。
ここで推定するものは、

$$HOUR = b_1+b_2C18+b_3AGE+b_4AGE2+b_5ED+b_6HI+\varepsilon$$

これにσに相当するものをｓとして、計７つの値を推定します。ただし、GAUSS の内部定義の事情から ed という変数は使えないので、下のプログラムでは ed1 としています。

プログラム

```
new; cls;
library maxlik;
maxset;
load data[51,7]=d:datafile6.txt;
data=data[2:51,.];

hour=data[.,2];
one=ones(50,1);
c18=data[.,4];
age=data[.,5];
age2=age^2;
ed1=data[.,6];
hi=data[.,7];
data=hour~one~c18~age~age2~ed1~hi;

_max_ParNames = {"CONST","C18","AGE","AGE2","ED","HI","SIGMA"};
_max_Algorithm = 5;
start={-1,0,0,0,0,0,1};
call maxprt(maxlik(data,0,&ll,start));

proc ll(b,data);
    local y,x,s,beta;
    y=data[.,1];
    x=data[.,2:7];
    s=b[7];
    beta=b[1:6];
    retp( (y.==0).*ln(1-cdfn(x*beta/s))+ (y.>0).*lnpdfmvn(y-x*beta,s^2) );
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood          -3.16591
Number of cases      50
```

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

Parameters	Estimates	Std. err.	Est./s.e.	Prob.	Gradient
CONST	-47.4325	45.8356	-1.035	0.1504	-0.0000
C18	-7.1220	3.1962	-2.228	0.0129	0.0000
AGE	2.8161	2.3687	1.189	0.1172	-0.0000
AGE2	-0.0432	0.0273	-1.579	0.0571	-0.0000
ED	2.5322	1.3753	1.841	0.0328	-0.0000
HI	0.0001	0.0002	0.580	0.2810	0.0000
SIGMA	22.4303	3.0445	7.368	0.0000	0.0000

Correlation matrix of the parameters

1.000	0.025	-0.923	0.894	-0.065	0.173	-0.122
0.025	1.000	-0.141	0.212	-0.093	0.057	-0.139
-0.923	-0.141	1.000	-0.990	-0.253	-0.159	0.135
0.894	0.212	-0.990	1.000	0.226	0.177	-0.162
-0.065	-0.093	-0.253	0.226	1.000	-0.300	0.016
0.173	0.057	-0.159	0.177	-0.300	1.000	-0.015
-0.122	-0.139	0.135	-0.162	0.016	-0.015	1.000

Number of iterations 1059
Minutes to convergence 1.62300

上のプログラムの今までのプログラムと違う点は、推定するｂの列ベクトルの最終項にｓを加えている点にあります。Log-Likelihood関数をつくって、それを列を足し合わせる前の列ベクトル形式を用いて、各係数に加えてｓもあわせて、Log-Likelihood関数が最大になるようにiterationをさせます。上の計算では、かなりのiterationを要することと、スタートベクトルを調整してやらないとiterationがうまくいかないことが多いこと、それに、アルゴリズムによっては全くiterationをしないことなど、かなりの大作業です。パッケージソフトでは、プログラムをコンパイルしていることと、かなりの近似計算をしているため、TOBITモデルの計算速度は、さほど遅くはありませんが、このようにLog-Likelihood関数を正式にたててiterationをさせるとかなりの時間と手間を要します。上のプログラムでは変数ｂにβとｓの両方が入っているわけで、それを区別するためにｂの１番目から６番目

をbetaとするためにbeta=b[1:6]としていて、ｓをｂの７番目とするためにs=b[7]としています。こうしたインデックスの書き方は、ｆｎの関数定義でｘについてｘ１はx[1]、ｘ２はx[2]としたのと同様です。あえて言うならば、これらは列ベクトルの１番上の要素、２番目の要素、...というふうに言えます。

さらに、上で示した、Maximum Likelihoodの一般形を第２項に用いたLog-Likelihood関数で同じように係数βとｓを推定してみると、

プログラム（変更箇所のみ）

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

画面表示

```
return code =    0
normal convergence

Mean log-likelihood         -3.16591
Number of cases      50

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

Parameters    Estimates     Std. err.  Est./s.e.  Prob.    Gradient
------------------------------------------------------------------
CONST          -47.4325       45.8169   -1.035   0.1503     -0.0000
C18             -7.1220        3.1963   -2.228   0.0129     -0.0000
AGE              2.8161        2.3677    1.189   0.1171     -0.0000
AGE2            -0.0432        0.0273   -1.580   0.0571      0.0000
ED               2.5322        1.3753    1.841   0.0328     -0.0000
HI               0.0001        0.0002    0.580   0.2810     -0.0000
SIGMA           22.4303        3.0445    7.367   0.0000      0.0000

Correlation matrix of the parameters
   1.000    0.025   -0.923    0.894   -0.065    0.173   -0.122
   0.025    1.000   -0.141    0.212   -0.093    0.057   -0.139
  -0.923   -0.141    1.000   -0.990   -0.253   -0.159    0.135
   0.894    0.212   -0.990    1.000    0.226    0.177   -0.162
  -0.065   -0.093   -0.253    0.226    1.000   -0.300    0.016
   0.173    0.057   -0.159    0.177   -0.300    1.000   -0.015
```

```
  -0.122   -0.139    0.135   -0.162    0.016   -0.015    1.000

Number of iterations     1059
Minutes to convergence      2.69867
```

以上のように、細かいinferenceの細かい相違を除くと、だいたいのところ、前のlnpdfmvnを用いた推定と同じ結果になります。

一方、TruncatedバージョンのTOBITの方は、yに相当する変数が正の数である場合のみ推定を行ないます。CDFをF、PDFをｆとするとLog-Likelihoodを導く合計する前の列ベクトルは

$$LL=-\ln F(x\beta/s)-\ln\sigma+\ln f((y-x\beta)/s)$$

実際にはこれを用いて、βとｓについて、上のLLの列ベクトルを足し合わせたものを最大化することになります。

プログラム

```
new; cls;
library maxlik;
maxset;
load data[51,7]=d:datafile6.txt;
data=data[2:51,.];

hour=data[.,2];
one=ones(50,1);
c18=data[.,4];
age=data[.,5];
age2=age^2;
ed1=data[.,6];
hi=data[.,7];
data=hour~one~c18~age~age2~ed1~hi;
data=delif(data,data[.,1].<=0);

_max_ParNames = {"CONST","C18","AGE","AGE2","ED","HI","SIGMA"};
_max_Algorithm = 5;
start={0,0,0,0,0,0,1};
call maxprt(maxlik(data,0,&ll,start));
```

```
    proc ll(b,data);
        local y,x,s,beta,cdf,pdf;
        y=data[.,1];
        x=data[.,2:7];
        s=b[7];
        beta=b[1:6];
        retp( -ln(cdfn(x*beta/s))-ln(s)+ln(pdfn((y-x*beta)/s)) );
    endp;
```

画面表示

return code = 0
normal convergence

Mean log-likelihood -3.88125
Number of cases 32

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

Parameters	Estimates	Std. err.	Est./s.e.	Prob.	Gradient
CONST	-0.8565	40.5103	-0.021	0.4916	0.0000
C18	0.3380	2.4274	0.139	0.4446	-0.0000
AGE	-0.3381	2.2577	-0.150	0.4405	-0.0000
AGE2	0.0024	0.0276	0.085	0.4660	-0.0000
ED	2.6673	0.9791	2.724	0.0032	-0.0000
HI	0.0002	0.0001	1.861	0.0314	0.0000
SIGMA	12.7419	1.8528	6.877	0.0000	0.0000

Correlation matrix of the parameters

1.000	0.162	-0.945	0.926	0.038	0.019	-0.072
0.162	1.000	-0.297	0.343	0.094	0.099	0.017
-0.945	-0.297	1.000	-0.994	-0.313	0.012	0.020
0.926	0.343	-0.994	1.000	0.301	-0.015	-0.022
0.038	0.094	-0.313	0.301	1.000	-0.316	0.114
0.019	0.099	0.012	-0.015	-0.316	1.000	0.058

```
  -0.072    0.017    0.020   -0.022    0.114    0.058    1.000

Number of iterations     934
Minutes to convergence      0.98867
```

上の計算では、組みこみ関数delifを用いて、その第１要素のデータについて、その第２要素の論理式で真である行をまるごと消去しています。そのｙに相当するものが正であるものだけのデータについて、与えられたLog-Likelihoodを最大化するbetaおよびｓを求めています。

このように、iterationをともなう計算はGAUSSプログラムの中心を占めますが、理論的に正確であるLog-Likelihoodを書いたからといって、スタートベクトルやアルゴリズム、そしてパラメーターの変域などの諸要因によって、必ずしもiterationが収束して推定値が得られるとは限らないのです。また、収束したからといって、そのLog-Likelihoodがグローバルに最大であるのかどうかは保証されてもいないのです。しかしながら、このことがあるといっても少しもGAUSSの価値を低めるものではありません。計算のすぐ出てくるパッケージソフトは近似計算をしていたり、断わりもなく変数をスケーリングをしていることが多く、その点でGAUSSは忠実に自分の組んだLog-Likelihoodどおりに最大化をしてくれます。計算機のスピードが発達した現在でも、iterationにはかなりの時間を要するもので、普通の学生研究者はコンピュータールームのモニターに「計算中。さわるな！」の張り紙をしたり、自分のコンピュータで計算している場合でも、iterationを回しておいて睡眠をとったりすることになります。（ただし、スタートベクトルによって、「極値」がGAUSSでたくさん見つかるからといって、そのことを騒ぎ立てることは自重すべきです。きっちりとGAUSS以外の数学系のソフトでLog-Likelihoodの形状と極値の数を調べるべきです。）また、SASと違って（私はSASはUNIXの上で教わりましたし、アメリカの大学院のUNIXサーバの中にはSASやFortranが標準で入っていてTelnetなどで遠隔操作できます）、GAUSSのUNIXの上でのプログラムは適してはいません。非能率的です。途中でオペレータにJOBを切られてしまいかねません。むしろ、高速な自分だけが使うパソコンのWINDOWS上での計算に適しています。WINDOWSの上であれば、メモリはパソコンの使用可能なメモリに依存して（バーチャルメモリも含む）いくらでも大きな計算ができます。そういう理由から、GAUSSが授業で多用される研究機関や大学院には、WINDOWSの上にGAUSSが組みこまれているはずです。