4.3 特殊関数とライブラリ MAXLIK [要 full version]　　ver. 0.2

いままでの関数群は、ライブラリーではなくて、いつでもGAUSSの環境のもとで自由に使えるものでした。ライブラリとは、厳密にはlibrary命令で呼び出す関数群を指します。一方、コンパイル済みのパッケージはuse命令で呼び出されます。（一部のプログラムではMAXLIKをlibrary命令とuse命令の両方で呼び出しているものもありますが、library命令だけにしても通常は動きます。一度だけ文頭でlibrary命令で呼び出せば、その後は、その中に入っている関数を、GAUSS上の組込み関数とまったく同じように操作できます。グラフィックのライブラリであるpgraphのように、graphset;としてそのグローバル変数初期設定値をリセットする必要があったように、ここでもmaxlikには、maxset;また、cmlには、cmlset;などの命令と組にして使い、そのグローバル変数をリセットする必要があります。

ポイント　use **名称;**　　**コンパイルされたパッケージを呼び出す**
library **名称;**　**通常のオープンソースのライブラリを呼び出す**
（例）use gpe2;　library grapgh;　library maxlik;　library cml;

ポイント　library maxlik;
maxset;
ライブラリMAXLIKを呼び出して、そのグローバル変数をリセットする

基本的に、＃から始まる行はコンパイルに関連した命令を記述してあるところで、コンパイルをしないかぎり、その部分を取り除いても通常のRUN操作には影響を与えません。また、AutoloderとAutodeleteがONの設定になっているかぎりは、通常のRUNでの使用では、external命令は不用です。コンパイル用と考えてください。（ただし、AutoloderとAutodeleteがONとなっているとコンパイルが遅くなるため、それらの設定をはずしてコンパイルすることが多いので、コンパイルしてプログラムをする「開発者」とコンピューターを共有している際にはONになっていることを確認してプログラムをしてください。）

ポイント　**＃から始まる命令はコンパイル関連のもので通常使用には無関係**
ポイント　external**命令もAutoloderとAutodeleteがONの場合は不用**

MAXLINKは制約条件なしの最大化を行なうもので、Maximum Likelihoodに特化した計算をして、そのVarianceCovariance行列や、Asymptotic Standard Error、Asymptotic t-ratioなどもあわせて計算してくれます。一方、CMLは制約条件つき（なしも含めて）の最大化を行なうもので、同様にMaximum Likelihoodに特化した計算をします。どちらの場合

もprocedure形式で、データが所与の場合のLog-Likelihood関数（係数とデータの２つに依存する２値関数）を作成してから、その関数形を呼び出して計算することになります。このMaximum Likelihoodを求めるアルゴリズムは、Newton法をはじめ、６種類の方法を選択できるほか、ステップに相当するところを固定値１だけではなくて、ラインサーチの様々オプションに設定にできます。いずれにしても、Log-Likelihood関数のprocedureでの作成までは、少なくとも、ユーザーサイドでプログラムする必要があります。このユーザーが作成した関数をどのようにして制約条件なしで最大化するのかといった機能が豊富にあるのが、このMAXLIKライブラリというものだと捉えてください。

ここで初心者も上級者も注意しなければならない点がいくつかあります。１つは、このMaxlikを含めほとんどのGAUSSライブラリおよびパッケージが、内部でもうすでに関数の全体にマイナスをつけたものを「最小化」しているということです。3.8の最適化アルゴリズムで説明しましたように、ニュートン系のアルゴリズムはすべて厳密には最小化をするプログラムでした。Maxlik/CMLまたGPE2などのGAUSS上のライブラリパッケージは、インプットとして関数をそのまま受けてつけて、内部で関数の全体にマイナスの符号をつけて、このマイナス関数を最小化させています。ただし、我々がGAUSSのライブラリで関数を呼び出す場合には、マイナスをつけないもともとの関数形をインプットします。そういう意味で、ニュートン法やBHHH法を自作した際には、本来ならばマイナス関数を呼び出すインプットに採用すべきなのですが、基本的にHessianをチェックするアルゴリズムにはなっていないので、最小値でなくて最大値が求まったわけです。もう１つには、logというのは、自然対数lnのことを計算上はさします。慣習上、Log-Likelihoodと書きますが、これはプログラムする際に自然対数のlnを使わなくてはいけません。特にプログラムの初心者には繰り返し指摘して注意を喚起する必要があります。

MAXLIKが最大化するLog-Likelihoodを考える上で、非線形の推定式を考えるわけですが、その中でもLog-Likelihoodが比較的簡単で扱いやすい形をしているものとして、労働経済学や教育などの分野をはじめとしてよく使われるLOGITとPROBITがあります。従属変数側に１と０から成る質的変数が入っている場合の推定です。以下の例では、教育の分野について、gradeが改善されたかされなかったによって、１と０を従属変数側に入れて推定するものです。線形の確率モデルでは、推定されたそれぞれの確率値が確率本来の０と１の間にこないという欠点があって、これを改善するために、それぞれの推定された確率値が、Ｘβの大きさに依存して、０と１におさまるように作られたものがLOGITとPROBITです。LOGITはロジスティック分布を関数を使ったもので、Ｘβが大きければそれぞれの推定確率値は１に近づき、小さければ０に近づくように、０と１の間にうまくおさまるように考えられたモデルです。PROBITはstandard normal分布を使って、Ｘβが大きければそれぞれの推定確率値は１に近づき、小さければ０に近づくように、０と１の間になるように考えられたモデルです。

LOGITモデル

今、Lin[2001]の8.2のGPE２を用いたlogitと同じ計算をしてみます。データは、ｄドライブにdatafile5.txt（Greene table19.1）というファイルがあるものとします。このモデルを選んだわけは、特にこのあと述べますPROBITモデルと同じようにgradientとHessianの手計算が楽であるからです。まずは、最も簡易なgradientとHessianを与えない方法で、Log-Likelihoodを最大化してみます。今、ロジスティック分布関数は

$$F(x_i\beta)=\frac{\exp(x_i\beta)}{1+\exp(x_i\beta)}=\frac{1}{1+\exp(-x_i\beta)} \quad \text{なぜなら} \quad \frac{1}{\exp(x_i\beta)}=\exp(-x_i\beta)$$

ですから、これを用いて、Likelihood関数はｙが１のときF(Xβ)、ｙが０のとき1-F(xβ)となるような２項確率の積になっていて、

$$L=\Pi F(x_i\beta)^{yi}\,(1-F(x_i\beta))^{1-yi}$$

となります。この場合のLog-Probabilityをベクトル表示で１行で書くと

$$LP=y\ln F(X\beta)+(1-y)(1-F(X\beta))$$

となります。それぞれのデータの個数をｎとすると、LPはｎ×１の列ベクトルです。ロジスティック分布のCDFであるF(Xβ)をこのLog-Probability関数に代入したものを最大化することになります。なお、MAXLIKでは、上のLog-Probabilityの列の合計をしたLog-Likelihoodではなくてこのような列ベクトルのLog-Probabilityを最大化すればよいようになっています。

GAUSSのプログラムでは、このロジスティック分布のCDFであるF(Xβ)を代入した形のLog-Likelihood（またはLog-Probabilityの列ベクトル）関数がβとデータ全体の２つに依存するようにprocを書いてやることがまず必要になります。それをstartベクトルとともにライブラリMAXLIKで呼び出します。

ポイント　MAXLIKの書式

{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));

または、

call maxprt(maxlik(dataset,0,&ll,start));

Log - Likelihood Functionのprocの作り方

proc ll(b,dataset);

ここで、bは独立変数のベクトル、datasetは使用するデータ全体

今、推定する式は下のような従属変数GRADE（ｙとする）が１と０といった値しかとらない離散モデルを考えます。（Prof.Lin[2001]の設定と同じにして比較するため定数項b_4は最終列にもってきています。以下、この章では便宜上この順番で通します。）

$$GRADE_i = b_1GPA_i + b_2TUCE_i + b_3PSIi + b_4 + \varepsilon_i$$

ただし、GRADE＝１　または　０

b_4　は切片定数項

プログラム

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

load data[33,4]=d:datafile5.txt;
data=data[2:33,.];
dataset=data[.,4]~data[.,1:3]~ones(rows(data),1);

start={0,0,0,0};
_max_Algorithm = 4;
{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
    local y,x,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=1./(1+exp(-x*b));
    retp(y.*ln(cdf)+(1-y).*ln(1-cdf));
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood        -0.402801
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
------------------------------------------------------------------
P01             2.8261        1.2632     2.237   0.0126      0.0000
P02             0.0952        0.1416     0.672   0.2507      0.0000
P03             2.3787        1.0647     2.234   0.0127      0.0000
P04           -13.0213        4.9326    -2.640   0.0041     -0.0000

Correlation matrix of the parameters
   1.000  -0.206   0.318  -0.734
  -0.206   1.000   0.099  -0.496
   0.318   0.099   1.000  -0.450
  -0.734  -0.496  -0.450   1.000

Number of iterations    6
Minutes to convergence     0.00917
```

上のように、まず33行４列のdataをload文で読みこんで、ラベルが入っている１行目を切り落とすために、２行目から33行目までをあらためてdataとしています。そして、便宜上（GPE２を使ったLIN[2001]8.2の変数の順番と一致させるため）１列目に元のデータの４列目のGRADEに相当する部分を、そして、最後の列にconstantの１ばかりの項がくるように加工したものをdatasetと定義します。MAXLIKを呼び出す前にまずやることは、procでLog-Likelihood functionを書くことです。（もちろんｆｎで１行で書いてもかまいません。しかしながら、この場合は読みにくくなりますので、procの書き方でいきます。）MAXLIKにおいてもCMLにおいてもこのLog-Likelihood Functionを書くという作業は必須となります。この関数の名前は任意ですが、英語の頭文字をとって、ll（エルエル）とします。MAXLIKやCMLで呼び出すLog-Likelihood functionには書式があって、必ず２つの引数が必要です。1番目の引数は、係数に相当する列ベクトル。２番目は、対象となるデータ行列（従属変数および独立変数をマージしたもの。切片項およびダミー変数等すべてを含む)。［Prof LinのGPE2では、これを巧妙狡猾に引数の左右を逆にして互換性をなくしていますから注意が必要。］まず、１と０からなる従属変数の列をｙとし、そのほかの列をｘとします。そして、logitのロジスティック分布に従うCDFを計算します。それを用いて最終的にLog-Likelihood（またはLog-Probability）をリターンとして返します。

そして、いよいよ、独立変数（切片がある場合にはそれを含む）と同じ数のstartベクトルを設定して、{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));としてMAXLIK

を呼び出してMaximum Likelihoodとそれに関連した統計量を計算させます。その際に、NEWTON法で計算させるのなら、_max_Algorithm = 4;として、その部分のグローバル変数の変更が必要となります。引数の１番目は、dataset、その次の２番目にはセレクトする変数ラベルベクトルが入りますが、通常は０としていることが多く、この意味は、与えられたdatasetのすべての列を使うことです。３番目の引数には、作成したLog-Likelihood functionの関数名にポインタ＆のしるしをつけて呼び出します。最後に４番目の引数には、startベクトルが入ります。

なお、この{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));という箇所は慣用的に入れ子にして一括計算させることがほとんどなのですが、具体的には、

iterationをして最大化をする　{x,f,g,cov,retcode} = maxlik(dataset,0,&ll,start);
と
結果を統計量とともに表示する　{x,f,g,cov,retcode} = maxprt(x,f,g,cov,retcode);

の２つの関数の呼び出しの連結した形になっています。（もちろん、これら２つの呼び出しを併記してもプログラムは動きます。）前者の複数のリターンが、後者の引数と一致しますから、２つの関数を入れ子にして、引数はdataset,0,&ll,startだけにできるわけです。なお、全体をcall文で呼んでも結果は同じになります。前者のmaklikだけを使うと、各iterationだけの羅列の表示になります。注意しなければならないことは、上の例では、計算アルゴリズムのグローバル変数の設定をしなければ、２のBFGSというメソッドになってしまって、収束しません。通常は、４のNewton法、または５のBHHH（ビートリプルエイチ）法が最もよく使われる方法です。以前自作したproc maxと同じstep間隔を１に設定したいのなら

```
_max_LineSearch = 1;          （デフォルトは２）
```

とすれば、ステップ間隔が２の自動設定になっていたのが、１の１間隔に変更されます。この場合には、まれに収束の際にオーバーシュートを起こします。また、３のHALFにするとステップ間隔は２分の１になって、iterationは細かくなり、関数形によっては計算を繰り返した分だけ誤差を拾って蓄積していくことになります。ちなみに、２のデフォルトでは、そのステップ幅を３次または２次の関数で自動的に最適化した値を使います。また、これまでの最適値問題の関数の計算のように、最大iteration数を変更することも可能です。

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

というデフォルトを、たとえば、もっと短い100回までのiterationに制限するのならば、数

値を1e+2に変更したこのグローバル設定を書き加えます。一般に、デフォルトの1e+5では関数形とスタートベクトルがうまくマッチしていなかったり、最大化のアルゴリズムの設定がふさわしくなかったり、関数形の設計にエラーがあるのに計算させていたりする場合に、コンピュータはiterationを続けることになります。特に、計算アルゴリズムのグローバル変数を２のデフォルトのままにしておくと、計算が止まらなくなる問題になります。

```
_max_Algorithm = 4;　または　_max_Algorithm = 5;
```

に変更して（Newton法またはBHHH法）、まず計算することを通常はおすすめします。ParametersのP01,P02,P03,P04のところを実際の変数名に変更しようと思えば、

```
_max_ParNames = {"GPA","TUCE","PSI","CONST"};
```

などと、実際の従属変数の変数名を文字の縦ベクトルとして引用符にくるんで代入します。この場合は、切片も含めて４変数分ありますから、４×１の列ベクトルになります。そういう意味で、カンマで区切ってあります。

また、最適値問題の関数と同様に、gradientベクトルとHessianの両方または一方が手計算可能であったり形がわかっていたりすれば、これらをグローバル変数としてMAKLIKを呼ぶ前に設定しておけば、数値的にgradientとHessianを計算するよりも計算が正確になり有効桁数が増えることになります。（このAnalytical gradient/Hessianと呼ばれる手計算方法は専門家の間でしばしば行なわれることで、GAUSSの大きな魅力の１つでもあります。）なお、このままの設定でもよいのですが、手計算で精度を上げたのなら、収束基準の桁数をさらに小さくすることも可能となります。

```
MAX_GradTol=1e-5;
```

となっている桁数をさらに小さな値、たとえば、1e-7とか1e-8とかにすることも可能で、逆に、gradientなどを与えずに計算させる場合に関数形が複雑で数値計算によるgradientに大幅な誤差が予想される場合には、この値を1e-3とかの大きな値にする必要もあります。

プログラム

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

```
load data[33,4]=d:datafile5.txt;
data=data[2:33,.];
dataset=data[.,4]~data[.,1:3]~ones(rows(data),1);

start={0,0,0,0};
_max_Algorithm = 4;
_max_MaxIters = 1e+3;
_max_ParNames = {"GPA","TUCE","PSI","CONST"};
_max_GradProc=&gradient;
_max_HessProc=&hessian;
{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
    local y,x,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=1./(1+exp(-x*b));
    retp(y.*ln(cdf)+(1-y).*ln(1-cdf));
endp;

proc gradient(b,dataset);
    local y,x,pdf,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=1./(1+exp(-x*b));
    pdf= exp(-x*b)./(1+exp(-x*b))^2;
    retp(y.*(pdf./cdf).*x-(1-y).*(pdf./(1-cdf)).*x);
endp;

proc hessian(b,dataset);
    local y,x,pdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    pdf=exp(-x*b)./(1+exp(-x*b))^2;
    retp(-(x.*pdf)'*x);
endp;
```

画面表示

```
return code =    0
normal convergence
```

```
Mean log-likelihood        -0.402801
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
------------------------------------------------------------------
GPA              2.8261        1.2629     2.238   0.0126      0.0000
TUCE             0.0952        0.1416     0.672   0.2507      0.0000
PSI              2.3787        1.0646     2.234   0.0127      0.0000
CONST          -13.0213        4.9313    -2.641   0.0041      0.0000

Correlation matrix of the parameters
    1.000   -0.207    0.318   -0.734
   -0.207    1.000    0.099   -0.496
    0.318    0.099    1.000   -0.449
   -0.734   -0.496   -0.449    1.000

Number of iterations    5
Minutes to convergence      0.00650
```

上のように、gradientとHessianをグローバル変数として設定すると、この場合、若干ではありますが、iterationの回数も６回から５回に減っています。太字で示しているように

_max_GradProc=&gradient;

_max_HessProc=&hessian;

としてgradientとHessianのグローバル変数を呼ぶのには、Log-Likelihood関数とまったく同じ形式のprocで作った（ｆｎで１行で作ってもかまいません）関数を、関数名にポインタのしるしの＆をつけて指し示して呼び出します。gradientの計算の仕方は、この場合、関数の関数の関数の微分のルールで計算をします。CDFに相当する部分の微分は、cdf=1./(1+exp(-x*b))と分数の形になっていますから微分のquotientルールを使って－exp(-x*b)./(1+exp(-x*b))^2に－ｘをかけたものになるはずです。マイナスとマイナスをキャンセルアウトしてｘを取り除いたものをPDFとおきます。大きな関数全体の方は、別に関数の関数の微分で解いて、lnFのところは、F'/Fの形、すなわち、PDF/CDFの形になってさらに関数の中のｘをかけます。既にマイナスはキャンセルアウトさせています。同様に

して、第２項目もマイナスに注意して微分してPDFとCDFを使って書き表して、その関数中の微分のマイナスをキャンセルアウトしたｘを忘れずにつけています。その計算の結果を直接リターンで返して、gradientのprocの作成は完了です。このgradientも元の関数であるLog-Probabilityと同様に、列ベクトルのままでかまいません。Hessianの計算計算の方は、この場合は、上で定義したものと同じPDFを使って、Hessian行列が

$$H=-\Sigma F_i(1-F_i)x_i'x_i$$

と与えられているとすれば、上のロジスティック分布の場合には、計算上、$F_i(1-F_i)$=PDFとなりますから、これをcomformableになるように行列ベクトル表示になるように加工して-(x.*pdf)'*x　を計算させてgradientのprocのリターンとして返しています。なお、ディメンションを確認しますと、((ｎ×ｋ).(ｎ×１))'ｎ×ｋですから、最終的には括弧の中を転置して、ｋ×ｎとｎ×ｋをかけて、Hはｋ×ｋになります。なお、括弧の中は要素対要素の計算ですから、後ろのｎ×１のそれぞれが前のそれぞれの行にかかっていきますから、計算後のディメンションは前の行列のｎ×ｋが保たれます。

PROBITモデル

今、CDFがロジステック分布ではなくて、standard normal分布の場合のprobitモデルについての計算を同様にやらせてみます。同じ２項確率を用いて、分布に相当するCDFとPDFの部分を変更するだけで手軽に変更できます。なお、standard normal分布のCDFとPDFを計算する組込み関数であるcdfnとpnfnを使います。その引数は、ｘｂとなります。まずは、gradientやHessianを指定しない簡易な方法で計算させます。

プログラム

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

load data[33,4]=d:datafile5.txt;
data=data[2:33,.];
dataset=data[.,4]~data[.,1:3]~ones(rows(data),1);

start={0,0,0,0};
_max_Algorithm = 4;
_max_MaxIters = 1e+3;
_max_ParNames = {"GPA","TUCE","PSI","CONST"};
{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));
```

```
proc ll(b,dataset);
    local y,x,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=cdfn(x*b);
    retp(y.*ln(cdf)+(1-y).*ln(1-cdf));
endp;
```

なお、gradientがそこから動かず、このプログラムのままでは動きません。そこで、計算方法アルゴリズムをBHHH法に変更して、

```
_max_Algorithm = 5;
```

としてみましょう。今度はかなりのiterationをしますが、正常に動くはずです。

画面表示

```
return code =    0
normal convergence

Mean log-likelihood        -0.400588
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
------------------------------------------------------------------
GPA              1.6258        0.6939     2.343   0.0096      0.0000
TUCE             0.0517        0.0839     0.617   0.2687      0.0000
PSI              1.4263        0.5951     2.397   0.0083      0.0000
CONST           -7.4523        2.5429    -2.931   0.0017      0.0000

Correlation matrix of the parameters
   1.000   -0.325    0.255   -0.663
  -0.325    1.000    0.050   -0.474
   0.255    0.050    1.000   -0.393
```

```
  -0.663   -0.474   -0.393    1.000

Number of iterations     67
Minutes to convergence      0.10450
```

今度は、元の４の設定にアルゴリズムの変更をを戻して、gradientだけを手計算でしたものをグローバル変数で設定して計算させてみましょう。なお、standard normalのCDFとPDFは正確に計算できる組込み関数を直接用います。全体のLog-Probabilityは、Logitのときと同じですが、CDFの中には－のサインはないのとquotientルールの－のサインは分数ではないので最初からありませんので、結果的にマイナスのサインがキャンセルアウトした場合のLOGITと同様な形になります。

プログラム

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

load data[33,4]=d:datafile5.txt;
data=data[2:33,.];
dataset=data[.,4]~data[.,1:3]~ones(rows(data),1);

start={0,0,0,0};
_max_Algorithm = 4;
_max_MaxIters = 1e+3;
_max_ParNames = {"GPA","TUCE","PSI","CONST"};
_max_GradProc=&gradient;
{x,f,g,cov,retcode} = maxprt(maxlik(dataset,0,&ll,start));

proc ll(b,dataset);
    local y,x,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=cdfn(x*b);
    retp(y.*ln(cdf)+(1-y).*ln(1-cdf));
endp;

proc gradient(b,dataset);
```

```
        local y,x,pdf,cdf;
        y=dataset[.,1]; x=dataset[.,2:5];
        cdf=cdfn(x*b);
        pdf=pdfn(x*b);
        retp(y.*(pdf./cdf).*x-(1-y).*(pdf./(1-cdf)).*x);
    endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood        -0.400588
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
-------------------------------------------------------------------
GPA              1.6258        0.6939    2.343   0.0096      0.0000
TUCE             0.0517        0.0839    0.617   0.2687      0.0000
PSI              1.4263        0.5950    2.397   0.0083     -0.0000
CONST           -7.4523        2.5425   -2.931   0.0017      0.0000

Correlation matrix of the parameters
   1.000  -0.325   0.255  -0.663
  -0.325   1.000   0.050  -0.474
   0.255   0.050   1.000  -0.393
  -0.663  -0.474  -0.393   1.000

Number of iterations    5
Minutes to convergence     0.00733
```

上のように、係数はほぼ同じになるはずですが、iterationの数は５回と格段に減少して、すぐに答えが得られます。なお、HessianはPDFとCDFに依存した形になるはずですがLOGITの時とは当然のことながら違った形になります。PROBITの場合のHessianは、

$$H = -\sum \frac{f_i^{\,2} x_i' x_i}{F_i(1-F_i)}$$

となりますから、F＝CDF, f＝PDFであるから、それをLOGITのときと同じように、行列ベクトル表示に書き換えて、最終的にｋ×ｋのディメンションになるように転置の括弧の計算の中に挟み込むようにｆとFの項を埋めこんで計算すると、

```
-(x.*(pdf^2./(cdf.*(1-cdf))))'*x
```

とHessianのリターンはなるはずである。（左右の括弧の数に注意して入力してください。）これをまとめて、

```
proc hessian(b,dataset);
    local y,x,pdf,cdf;
    y=dataset[.,1]; x=dataset[.,2:5];
    cdf=cdfn(x*b);
    pdf=pdfn(x*b);
    retp(-(x.*(pdf^2./(cdf.*(1-cdf))))'*x);
endp;
```

のHessianのprocを前のgradientだけのプログラムの一番最後に付け加えた上で、さらに

```
_max_HessProc=&hessian;
```

のHessianを呼び出すグローバル変数の設定文をMAKLIKの本文の直前におけば、完璧なgradientとHessianが両方揃った計算ができます。結果は、以前よりもiterationが多くなるはずですが、Hessianの形が正しければ、桁数の精度は上がっているはずです。

このように、MAXLIKなどのライブラリにおいてもgradientとHessianの両方または、少なくともgradientだけを設定してMaximum Log-Likelihoodを計算させることがベストでありかつ、GAUSSの最もGAUSSたるプログラムの姿です。

なお、gradientとHessianの手計算をanalytical gradient/Hessianと言い、GAUSSが計算で出してくるものの方は、numerical gradient/Hessianと言いますが、これらを比べて確かめてくれる機能もMAXLIKにはあります。

```
_max_GradChekTol=1e-3;
```

などと0.001までのanalyticalとnumericalの結果に違いがあれば、そこで計算をやめて違いを表示します。これはHessianにも使えて共用です。もっとも、Hessian自体の手計算が間違っている場合には、このオプションがきくまでもなく、行列をinvertできないでそこで計算がストップする場合が多いようです。